FLTK 1.5.0 Programming Manual

By F. Costantini, M. Melcher,
A. Schlosser, B. Spitzak, and M. Sweet.

Copyright © 1998 - 2026 by Bill Spitzak and others.

This software and manual are provided under the terms of the GNU Library General Public License.
Permission is granted to reproduce this manual or any portion for any purpose,
provided this copyright and permission notice are preserved.

Generated by Doxygen 1.9.8

October 4, 2026

Git revision 0bbf02e76300f9ac4970d6d4666a56d56e409fd3

1 FLTK Programming Manual 1
1.1 Preface . . . 2
1.1.1 Organization . . . 2
1.1.2 Conventions . . . 3
1.1.3 Abbreviations . . . 4
1.1.4 Copyrights and Trademarks . . . 4
1.2 Introduction to FLTK . . . 4
1.2.1 History of FLTK . . . 5
1.2.2 Features . . . 6
1.2.3 Licensing . . . 7
1.2.4 What Does "FLTK" Mean? . . . 7
1.2.5 FLUID . . . 7
1.2.6 Building and Installing FLTK with CMake . . . 7
1.2.7 Building FLTK Under Microsoft Windows . . . 8
1.2.7.1 Free and Commercial Microsoft Visual Studio Versions . . . 8
1.2.7.2 Using the Visual C++ DLL Library . . . 8
1.2.7.3 GNU toolsets (Cygwin or MinGW) hosted on Windows . . . 9
1.2.8 Internet Resources . . . 9
1.2.9 Reporting Bugs . . . 10
1.3 FLTK Basics . . . 10
1.3.1 Writing Your First FLTK Program . . . 10
1.3.1.1 Creating the Widgets . . . 11
1.3.1.2 Creating Widget Hierarchies . . . 11
1.3.1.3 Get/Set Methods . . . 12
1.3.1.4 Redrawing After Changing Attributes . . . 12
1.3.1.5 Labels . . . 12
1.3.1.6 Showing the Window . . . 12
1.3.1.7 The Main Event Loop . . . 12
1.3.2 Naming Conventions . . . 13
1.3.3 Header Files . . . 13
1.3.4 Compiling Programs that Use FLTK . . . 13
1.3.4.1 Compiling Programs with Standard Compilers . . . 13
1.3.4.2 Compiling Programs with the 'fltk-config' Script . . . 14
1.3.4.3 Compiling Multiple Source Files with 'fltk-config' . . . 15
1.3.4.4 Compiling Programs with Makefiles . . . 15
1.3.4.5 Compiling Programs with Microsoft Visual C++ . . . 16
1.4 Common Widgets and Attributes . . . 16
1.4.1 Buttons . . . 16
1.4.2 Text . . . 17
1.4.3 Valuators . . . 18
1.4.4 Groups . . . 19
1.4.5 Setting the Size and Position of Widgets . . . 19

1.4.6 Colors . . . 20
1.4.7 Box Types . . . 21
1.4.7.1 Making Your Own Boxtypes . . . 21
1.4.8 Labels and Label Types . . . 22
1.4.9 Callbacks . . . 27
1.4.10 When and Reason . . . 28
1.4.11 Shortcuts . . . 28
1.5 Coordinates and Layout Widgets . . . 28
1.5.1 The Widget Coordinate System . . . 28
1.5.2 Layout and Container Widgets . . . 29
1.5.2.1 The Fl_Flex Layout Widget . . . 29
1.5.2.2 The Fl_Grid Layout Widget . . . 30
1.5.2.3 The Fl_Pack Layout Widget . . . 31
1.5.2.4 The Fl_Scroll Container Widget . . . 31
1.5.2.5 The Fl_Tabs Container Widget . . . 32
1.5.2.6 The Fl_Tile Layout Widget . . . 32
1.5.2.7 The Fl_Wizard Container Widget . . . 32
1.6 How Does Resizing Work? . . . 33
1.6.1 Resizing can be disabled . . . 33
1.6.2 Resizing can be simple . . . 33
1.6.3 Resizing can be complex . . . 34
1.6.4 Practical examples . . . 35
1.7 Designing a Simple Text Editor . . . 38
1.7.1 Determining the Goals of the Text Editor . . . 39
1.7.2 Chapter 1: A Minimal App . . . 39
1.7.3 Chapter 2: Adding a Menu Bar . . . 40
1.7.4 Chapter 3: Adding a Text Editor widget . . . 41
1.7.5 Chapter 4: Reading and Writing Files . . . 42
1.7.6 Chapter 5: Cut, Copy, and Paste . . . 44
1.7.7 Chapter 6: Find and Find Next . . . 45
1.7.8 Chapter 7: Replace and Replace Next . . . 46
1.7.9 Chapter 8: Editor Features . . . 48
1.7.10 Chapter 9: Split Editor . . . 48
1.7.11 Chapter 10: Syntax Highlighting . . . 49
1.8 Drawing Things in FLTK . . . 53
1.8.1 When Can You Draw Things in FLTK? . . . 53
1.8.2 What Units Do FLTK Functions Use? . . . 53
1.8.3 Drawing Functions . . . 54
1.8.3.1 Boxes . . . 55
1.8.3.2 Clipping . . . 56
1.8.3.3 Colors . . . 57
1.8.3.4 Color Contrast . . . 59

1.8.3.5 Line Dashes and Thickness . . . 60
1.8.3.6 Drawing Fast Shapes . . . 61
1.8.3.7 Drawing Complex Shapes . . . 64
1.8.3.8 Drawing Text . . . 66
1.8.3.9 Fonts . . . 69
1.8.3.10 Character Encoding . . . 69
1.8.3.11 Drawing Overlays . . . 70
1.8.4 Drawing Images . . . 70
1.8.4.1 Direct Image Drawing . . . 70
1.8.4.2 Direct Image Reading . . . 72
1.8.4.3 Direct Image Saving . . . 72
1.8.4.4 Image Classes . . . 72
1.8.5 Offscreen Drawing . . . 74
1.9 Handling Events . . . 75
1.9.1 The FLTK Event Model . . . 75
1.9.2 Mouse Events . . . 75
1.9.2.1 FL_PUSH . . . 76
1.9.2.2 FL_DRAG . . . 76
1.9.2.3 FL_RELEASE . . . 76
1.9.2.4 FL_MOVE . . . 76
1.9.2.5 FL_MOUSEWHEEL . . . 76
1.9.3 Focus Events . . . 76
1.9.3.1 FL_ENTER . . . 76
1.9.3.2 FL_LEAVE . . . 77
1.9.3.3 FL_FOCUS . . . 77
1.9.3.4 FL_UNFOCUS . . . 77
1.9.4 Keyboard Events . . . 77
1.9.4.1 FL_KEYBOARD, FL_KEYDOWN, FL_KEYUP . . . 77
1.9.4.2 FL_SHORTCUT . . . 78
1.9.5 Widget Events . . . 78
1.9.5.1 FL_DEACTIVATE . . . 78
1.9.5.2 FL_ACTIVATE . . . 78
1.9.5.3 FL_HIDE . . . 78
1.9.5.4 FL_SHOW . . . 79
1.9.6 Clipboard Events . . . 79
1.9.6.1 FL_PASTE . . . 79
1.9.6.2 FL_SELECTIONCLEAR . . . 79
1.9.7 Drag and Drop Events . . . 79
1.9.7.1 Dropped filenames . . . 79
1.9.7.2 FL_DND_ENTER . . . 80
1.9.7.3 FL_DND_DRAG . . . 80
1.9.7.4 FL_DND_LEAVE . . . 80

1.9.7.5 FL_DND_RELEASE . . . 80
1.9.8 Other events . . . 80
1.9.8.1 FL_SCREEN_CONFIGURATION_CHANGED . . . 80
1.9.8.2 FL_FULLSCREEN . . . 80
1.9.9 Fl::event_∗() methods . . . 81
1.9.10 Event Propagation . . . 81
1.9.11 FLTK Compose-Character Sequences . . . 82
1.10 Adding and Extending Widgets . . . 83
1.10.1 Subclassing . . . 83
1.10.2 Making a Subclass of Fl_Widget . . . 83
1.10.3 The Constructor . . . 83
1.10.4 Protected Methods of Fl_Widget . . . 84
1.10.5 Handling Events . . . 86
1.10.6 Drawing the Widget . . . 87
1.10.7 Resizing the Widget . . . 87
1.10.8 Making a Composite Widget . . . 88
1.10.9 Cut and Paste Support . . . 89
1.10.10 Drag And Drop Support . . . 90
1.10.11 Making a subclass of Fl_Window . . . 90
1.11 Using OpenGL . . . 90
1.11.1 Using OpenGL in FLTK . . . 90
1.11.2 Making a Subclass of Fl_Gl_Window . . . 91
1.11.2.1 Defining the Subclass . . . 91
1.11.2.2 The draw() Method . . . 91
1.11.2.3 The handle() Method . . . 91
1.11.3 OpenGL and support of HighDPI displays . . . 92
1.11.4 Using OpenGL in Normal FLTK Windows . . . 93
1.11.5 Using FLTK widgets in OpenGL Windows . . . 93
1.11.6 OpenGL Drawing Functions . . . 94
1.11.7 Speeding up OpenGL . . . 95
1.11.8 Using OpenGL Optimizer with FLTK . . . 96
1.11.9 Using OpenGL 3.0 (or higher versions) . . . 97
1.12 FLTK Runtime Options . . . 99
1.12.1 Runtime Options . . . 99
1.12.2 Obtaining Current Settings . . . 99
1.12.3 Administrative Tool . . . 99
1.12.4 List of Options . . . 100
1.13 Advanced FLTK . . . 100
1.13.1 Multithreading . . . 100
1.13.2 FLTK multithread locking - Fl::lock() and Fl::unlock() . . . 101
1.13.3 Simple multithreaded examples using Fl::lock . . . 102
1.13.4 FLTK multithreaded "lockless programming" . . . 103

1.13.5 FLTK multithreaded Constraints . . . 104
1.14 Unicode and UTF-8 Support . . . 105
1.14.1 Overview . . . 105
1.14.2 Quick Start . . . 105
1.14.3 What is Unicode and UTF-8? . . . 105
1.14.4 Unicode Functions in FLTK . . . 106
1.14.4.1 Text Validation and Analysis . . . 106
1.14.4.2 Text Conversion . . . 106
1.14.4.3 Text Navigation . . . 107
1.14.4.4 String Operations . . . 107
1.14.4.5 File Operations . . . 108
1.14.5 Best Practices . . . 108
1.14.5.1 File Handling . . . 108
1.14.5.2 String Processing . . . 108
1.14.5.3 Display and UI . . . 108
1.14.5.4 Performance Notes . . . 109
1.14.6 Common Issues and Solutions . . . 109
1.14.6.1 "My international text shows up as question marks" . . . 109
1.14.6.2 "File operations fail with international names" . . . 109
1.14.6.3 "String length calculations are wrong" . . . 109
1.14.6.4 "Text gets corrupted when I truncate it" . . . 109
1.14.7 Error Handling . . . 110
1.14.8 Current Limitations . . . 110
1.15 Constants and Enumerations . . . 111
1.15.1 Version Numbers . . . 111
1.15.2 Events . . . 111
1.15.3 Callback "When" Conditions . . . 112
1.15.4 Fl::event_button() Values . . . 113
1.15.5 Fl::event_key() Values . . . 113
1.15.6 Fl::event_state() Values . . . 114
1.15.7 Alignment Values . . . 115
1.15.8 Fonts . . . 115
1.15.9 Colors . . . 116
1.15.9.1 Color Constants . . . 116
1.15.10 Cursors . . . 117
1.15.11 FD "When" Conditions . . . 118
1.15.12 Damage Masks . . . 118
1.16 GLUT Compatibility . . . 118
1.16.1 Using the GLUT Compatibility Header File . . . 118
1.16.2 Known Problems . . . 119
1.16.3 Mixing GLUT and FLTK Code . . . 120
1.16.4 class Fl_Glut_Window . . . 120

1.16.4.1 Class Hierarchy . . . 120
1.16.4.2 Include Files . . . 120
1.16.4.3 Description . . . 120
1.16.4.4 Members . . . 120
1.16.4.5 Methods . . . 121
1.17 Forms Compatibility . . . 121
1.17.1 Importing Forms Layout Files . . . 122
1.17.2 Using the Compatibility Header File . . . 122
1.17.3 Problems You Will Encounter . . . 122
1.17.4 Additional Notes . . . 124
1.18 Operating System Issues . . . 126
1.18.1 Accessing the OS Interfaces . . . 126
1.18.2 The Wayland/X11 hybrid library . . . 126
1.18.3 The UNIX (X11) Interface . . . 127
1.18.3.1 Handling Other X Events . . . 127
1.18.3.2 Drawing using Xlib . . . 129
1.18.3.3 Changing the Display, Screen, or X Visual . . . 129
1.18.3.4 Using a Subclass of Fl_Window for Special X Stuff . . . 131
1.18.3.5 Setting the Icon of a Window . . . 133
1.18.3.6 X Resources . . . 133
1.18.3.7 Display Scaling Factor . . . 134
1.18.4 The Windows Interface . . . 134
1.18.4.1 Using filenames with non-ASCII characters . . . 134
1.18.4.2 Responding to WM_QUIT . . . 134
1.18.4.3 Handling Other Windows API Messages . . . 135
1.18.4.4 Drawing Things Using the Windows GDI . . . 135
1.18.4.5 HighDPI support . . . 136
1.18.4.6 Display Scaling Factor . . . 136
1.18.4.7 Setting the Icon of a Window . . . 136
1.18.4.8 How to Not Get a MSDOS Console Window . . . 136
1.18.4.9 Known Windows Bugs and Problems . . . 137
1.18.5 The Apple OS X Interface . . . 137
1.18.5.1 Setting the icon of an application . . . 139
1.18.5.2 Drawing Things Using Quartz . . . 139
1.18.5.3 Internationalization . . . 139
1.18.5.4 Fl_Double_Window . . . 140
1.18.5.5 Mac File System Specifics . . . 140
1.18.6 The Wayland Interface . . . 140
1.18.6.1 HiDPI display support . . . 141
1.18.6.2 Window icons . . . 141
1.18.6.3 Window titlebars . . . 141
1.18.6.4 Fl_Double_Window . . . 141

1.19 Migrating Code from FLTK 1.3 to 1.4 . . . 142
1.19.1 Changes in Header Files . . . 142
1.19.2 Fl_Preferences . . . 143
1.19.3 Fl::add_timeout and friends . . . 143
1.19.4 New FL_OVERRIDE Macro . . . 143
1.19.5 Fl_Image::copy() 'const' . . . 144
1.19.6 Using X11 specific code with a "hybrid" FLTK library . . . 144
1.19.7 Modern CMake . . . 145
1.19.8 New FL_HELVETICA Font on Windows . . . 147
1.20 Migrating Code from FLTK 1.4 to 1.5 . . . 148
1.20.1 CMake is required to build FLTK . . . 148
1.20.2 FLTK requires C++11 or higher . . . 148
1.20.3 Default for FORMS compatibility is now OFF . . . 148
1.20.4 Simplified FL_OVERRIDE Macro . . . 148
1.20.5 Removed Methods and Functions . . . 148
1.20.6 More Deprecated Methods and Functions . . . 149
1.21 Software License . . . 149
1.22 Example Source Code . . . 154
1.22.1 Example Applications: Overview . . . 154
1.22.1.1 adjuster . . . 155
1.22.1.2 animated . . . 155
1.22.1.3 arc . . . 155
1.22.1.4 ask . . . 155
1.22.1.5 bitmap . . . 155
1.22.1.6 blocks . . . 155
1.22.1.7 boxtype . . . 155
1.22.1.8 browser . . . 155
1.22.1.9 button . . . 156
1.22.1.10 buttons . . . 156
1.22.1.11 cairo_test . . . 156
1.22.1.12 checkers . . . 156
1.22.1.13 clipboard . . . 156
1.22.1.14 clock . . . 156
1.22.1.15 colbrowser . . . 156
1.22.1.16 color_chooser . . . 156
1.22.1.17 cube . . . 156
1.22.1.18 CubeView . . . 156
1.22.1.19 cursor . . . 157
1.22.1.20 curve . . . 157
1.22.1.21 demo . . . 157
1.22.1.22 device . . . 157
1.22.1.23 doublebuffer . . . 157

1.22.1.24 editor . . . 157
1.22.1.25 fast_slow . . . 157
1.22.1.26 file_chooser . . . 157
1.22.1.27 fonts . . . 157
1.22.1.28 forms . . . 158
1.22.1.29 fractals . . . 158
1.22.1.30 fullscreen . . . 158
1.22.1.31 gl_image . . . 158
1.22.1.32 gl_overlay . . . 158
1.22.1.33 glpuzzle . . . 158
1.22.1.34 hello . . . 158
1.22.1.35 help_dialog . . . 158
1.22.1.36 icon . . . 158
1.22.1.37 iconize . . . 158
1.22.1.38 image . . . 159
1.22.1.39 inactive . . . 159
1.22.1.40 input . . . 159
1.22.1.41 input_choice . . . 159
1.22.1.42 keyboard . . . 159
1.22.1.43 label . . . 159
1.22.1.44 line_style . . . 159
1.22.1.45 list_visuals . . . 159
1.22.1.46 mandelbrot . . . 159
1.22.1.47 menubar . . . 159
1.22.1.48 message . . . 160
1.22.1.49 minimum . . . 160
1.22.1.50 native-filechooser . . . 160
1.22.1.51 navigation . . . 160
1.22.1.52 offscreen . . . 160
1.22.1.53 output . . . 160
1.22.1.54 overlay . . . 160
1.22.1.55 pack . . . 160
1.22.1.56 pixmap . . . 160
1.22.1.57 pixmap_browser . . . 160
1.22.1.58 preferences . . . 161
1.22.1.59 radio . . . 161
1.22.1.60 resizebox . . . 161
1.22.1.61 rotated_text . . . 161
1.22.1.62 resize . . . 161
1.22.1.63 scroll . . . 161
1.22.1.64 shape . . . 161
1.22.1.65 subwindow . . . 161

1.22.1.66 sudoku . . . 161
1.22.1.67 symbols . . . 161
1.22.1.68 table . . . 162
1.22.1.69 tabs . . . 162
1.22.1.70 threads . . . 162
1.22.1.71 tile . . . 162
1.22.1.72 tiled_image . . . 162
1.22.1.73 tree . . . 162
1.22.1.74 twowin . . . 162
1.22.1.75 unittests . . . 162
1.22.1.76 utf8 . . . 162
1.22.1.77 valuators . . . 162
1.22.1.78 windowfocus . . . 163
1.22.1.79 FLUID . . . 163
1.22.2 Example Applications: Images . . . 163
1.22.2.1 cairo_test . . . 163
1.22.2.2 icon . . . 163
1.22.2.3 unittests . . . 164
1.23 FAQ (Frequently Asked Questions) . . . 164
1.23.1 Where do I start learning FLTK? . . . 165
1.23.2 How do I make a box with text? . . . 165
1.23.3 Can I use FLTK to make closed-source commercial applications? . . . 165
1.23.4 Hitting the 'Escape' key closes windows - how do I prevent this? . . . 165

2 Fl_Terminal Technical Documentation **167**
2.1 The Escape Codes Fl_Terminal Supports . . . 167
2.2 Useful Terminal Escape Code Documentation . . . 168
2.3 Fl_Terminal Design Document . . . 169

3 Development of the FLTK library **175**
3.1 FLTK Code Management Plan (CMP) . . . 175
3.1.1 CMP Coding Intro . . . 175
3.1.2 Communication . . . 175
3.1.3 Specific Goals . . . 176
3.1.4 Documentation . . . 176
3.1.5 Coding Standards . . . 176
3.1.6 General Coding Style . . . 176
3.1.7 Source File Documentation . . . 177
3.1.8 Doxygen Documentation (Comments) . . . 178
3.1.9 General Developer Recommendations . . . 178
3.1.10 Documenting Temporary Code or Issues . . . 179
3.1.11 Documenting Classes and Structures . . . 180
3.1.12 Documenting Enumerations . . . 180

3.1.13 Documenting Functions and Methods . . . 180
3.1.14 Documenting Members and Variables . . . 181
3.1.15 Methodology, Algorithms, Etc. . . . 181
3.1.16 C++ Portability . . . 182
3.1.16.1 FLTK 1.1.x Restrictions . . . 182
3.1.16.2 FLTK 1.3.x Restrictions . . . 182
3.1.16.3 FLTK 1.4.x Restrictions . . . 182
3.1.16.4 FLTK 1.5.x Restrictions . . . 183
3.1.16.5 Source File Naming . . . 183
3.1.17 Structure/Class Naming . . . 183
3.1.18 Constant/Enumeration Naming . . . 183
3.1.19 Preprocessor Variables . . . 183
3.1.20 Miscellaneous Coding Practices . . . 185
3.1.20.1 Using switch() and 'FALLTHROUGH' Comments . . . 185
3.1.20.2 Useful Visual Studio C++ Macros (Windows) . . . 185
3.1.20.3 Useful macOS C++ Macros . . . 186
3.1.20.4 Useful GNU C++ Macros . . . 187
3.1.21 Source Files . . . 187
3.1.22 Build System (CMake) . . . 188
3.2 The Wayland backend for its developer . . . 188
3.2.1 Introduction to Wayland . . . 188
3.2.2 Building libfltk as a Wayland client . . . 189
3.2.3 The hybrid Wayland/X11 platform . . . 190
3.2.4 Listeners . . . 190
3.2.5 Opening a Wayland connection . . . 191
3.2.6 Wayland windows and surfaces . . . 193
3.2.7 Menu windows and other popups . . . 194
3.2.8 Fl_Wayland_Graphics_Driver and Fl_Cairo_Graphics_Driver . . . 195
3.2.9 Wayland buffers . . . 195
3.2.10 Throttling window redraws . . . 196
3.2.11 Buffer factory . . . 197
3.2.12 Displays and HighDPI support . . . 199
3.2.13 Mouse and trackpad handling . . . 200
3.2.14 Tablet and pen handling . . . 201
3.2.15 Wayland cursors . . . 201
3.2.16 Keyboard support . . . 202
3.2.17 Support of text input methods . . . 202
3.2.18 Interface with libdecor . . . 203
3.2.19 Copy/Paste/Drag-n-Drop . . . 204
3.2.20 EGL as support for OpenGL . . . 204
3.2.21 FLTK-defined, Wayland-specific types . . . 205
3.2.22 Documentation resources . . . 207

3.3 Developer info for bundled libs . . . 207
3.3.1 Introduction . . . 207
3.3.2 Current status . . . 208
3.3.3 How to update the bundled libraries . . . 208
3.3.4 zlib: . . . 209
3.3.5 png: . . . 209
3.3.6 jpeg: . . . 210
3.3.7 nanosvg: . . . 210
3.3.8 libdecor: . . . 211
3.4 Developer Information . . . 211
3.4.1 Non-ASCII Characters . . . 213
3.4.2 Document Structure . . . 213
3.4.3 Creating Links . . . 214
3.4.4 Paragraph Layout . . . 214
3.4.5 Navigation Elements . . . 215

4 Todo List **217**

5 Deprecated List **221**

6 Topic Index **225**
6.1 Topics . . . 225

7 Namespace Index **227**
7.1 Namespace List . . . 227

8 Hierarchical Index **229**
8.1 Class Hierarchy . . . 229

9 Class Index **233**
9.1 Class List . . . 233

10 File Index **241**
10.1 File List . . . 241

11 Topic Documentation **247**
11.1 Version Numbers . . . 247
11.1.1 Detailed Description . . . 247
11.1.2 Macro Definition Documentation . . . 247
11.1.2.1 FL_ABI_VERSION . . . 247
11.1.2.2 FL_API_VERSION . . . 248
11.1.2.3 FL_MAJOR_VERSION . . . 248
11.1.2.4 FL_MINOR_VERSION . . . 248
11.1.2.5 FL_PATCH_VERSION . . . 248
11.1.2.6 FL_VERSION . . . 248

11.2 Events handling functions . . . 249
11.2.1 Detailed Description . . . 252
11.2.2 Function Documentation . . . 252
11.2.2.1 add_handler() [1/2] . . . 252
11.2.2.2 add_handler() [2/2] . . . 252
11.2.2.3 add_system_handler() . . . 253
11.2.2.4 belowmouse() [1/2] . . . 253
11.2.2.5 belowmouse() [2/2] . . . 253
11.2.2.6 callback_reason() . . . 253
11.2.2.7 compose() . . . 254
11.2.2.8 compose_reset() . . . 254
11.2.2.9 disable_im() . . . 254
11.2.2.10 enable_im() . . . 255
11.2.2.11 event() . . . 255
11.2.2.12 event_button() . . . 255
11.2.2.13 event_button1() . . . 255
11.2.2.14 event_button2() . . . 255
11.2.2.15 event_button3() . . . 255
11.2.2.16 event_button4() . . . 255
11.2.2.17 event_button5() . . . 256
11.2.2.18 event_buttons() . . . 256
11.2.2.19 event_clicks() [1/2] . . . 256
11.2.2.20 event_clicks() [2/2] . . . 256
11.2.2.21 event_clipboard() . . . 256
11.2.2.22 event_clipboard_type() . . . 256
11.2.2.23 event_dispatch() . . . 256
11.2.2.24 event_dx() . . . 257
11.2.2.25 event_dx_f() . . . 257
11.2.2.26 event_dy() . . . 257
11.2.2.27 event_dy_f() . . . 257
11.2.2.28 event_inside() [1/2] . . . 257
11.2.2.29 event_inside() [2/2] . . . 258
11.2.2.30 event_is_click() [1/2] . . . 258
11.2.2.31 event_is_click() [2/2] . . . 258
11.2.2.32 event_key() [1/2] . . . 259
11.2.2.33 event_key() [2/2] . . . 259
11.2.2.34 event_length() . . . 259
11.2.2.35 event_original_key() . . . 259
11.2.2.36 event_state() [1/2] . . . 260
11.2.2.37 event_state() [2/2] . . . 260
11.2.2.38 event_text() . . . 260
11.2.2.39 event_x_root() . . . 261

11.2.2.40 event_y_root() . . . 261
11.2.2.41 fl_callback_reason_str() . . . 261
11.2.2.42 fl_eventname_str() . . . 261
11.2.2.43 fl_fontname_str() . . . 261
11.2.2.44 focus() [1/2] . . . 262
11.2.2.45 focus() [2/2] . . . 262
11.2.2.46 get_key() . . . 262
11.2.2.47 get_mouse() . . . 262
11.2.2.48 handle() . . . 263
11.2.2.49 handle_() . . . 263
11.2.2.50 last_handler() . . . 264
11.2.2.51 pushed() [1/2] . . . 264
11.2.2.52 pushed() [2/2] . . . 264
11.2.2.53 remove_handler() . . . 264
11.2.2.54 remove_system_handler() . . . 264
11.2.2.55 test_shortcut() . . . 264
11.2.3 Variable Documentation . . . 265
11.2.3.1 fl_callback_reason_names . . . 265
11.2.3.2 fl_fontnames . . . 265
11.2.4 Pen and tablet event handling . . . 265
11.2.4.1 Detailed Description . . . 267
11.2.4.2 Enumeration Type Documentation . . . 267
11.2.4.3 Function Documentation . . . 269
11.3 Callback Function Typedefs . . . 276
11.3.1 Detailed Description . . . 277
11.3.2 Typedef Documentation . . . 277
11.3.2.1 Fl_Event_Dispatch . . . 277
11.3.2.2 Fl_Idle_Handler . . . 277
11.3.2.3 Fl_Old_Idle_Handler . . . 277
11.3.2.4 Fl_Timeout_Handler . . . 277
11.4 Windows handling functions . . . 278
11.4.1 Detailed Description . . . 278
11.4.2 Function Documentation . . . 278
11.4.2.1 default_atclose() . . . 278
11.4.2.2 first_window() [1/2] . . . 278
11.4.2.3 first_window() [2/2] . . . 279
11.4.2.4 grab() [1/2] . . . 279
11.4.2.5 grab() [2/2] . . . 279
11.4.2.6 modal() . . . 279
11.4.2.7 next_window() . . . 279
11.4.2.8 set_atclose() . . . 279
11.4.3 Variable Documentation . . . 280

11.4.3.1 atclose . . . 280
11.5 Selection & Clipboard functions . . . 280
11.5.1 Detailed Description . . . 280
11.5.2 Function Documentation . . . 281
11.5.2.1 add_clipboard_notify() . . . 281
11.5.2.2 clipboard_contains() . . . 281
11.5.2.3 copy() . . . 281
11.5.2.4 dnd() . . . 282
11.5.2.5 paste() [1/2] . . . 282
11.5.2.6 paste() [2/2] . . . 282
11.5.2.7 selection() . . . 283
11.5.2.8 selection_owner() [1/2] . . . 283
11.5.2.9 selection_owner() [2/2] . . . 283
11.5.2.10 selection_to_clipboard() [1/2] . . . 283
11.5.2.11 selection_to_clipboard() [2/2] . . . 284
11.6 Screen functions . . . 284
11.6.1 Detailed Description . . . 285
11.6.2 Function Documentation . . . 285
11.6.2.1 keyboard_screen_scaling() . . . 285
11.6.2.2 normalized_screen_scale() [1/2] . . . 286
11.6.2.3 normalized_screen_scale() [2/2] . . . 286
11.6.2.4 screen_count() . . . 286
11.6.2.5 screen_dpi() . . . 287
11.6.2.6 screen_num() [1/2] . . . 287
11.6.2.7 screen_num() [2/2] . . . 287
11.6.2.8 screen_scale() [1/2] . . . 288
11.6.2.9 screen_scale() [2/2] . . . 288
11.6.2.10 screen_scaling_supported() . . . 288
11.6.2.11 screen_work_area() [1/3] . . . 289
11.6.2.12 screen_work_area() [2/3] . . . 289
11.6.2.13 screen_work_area() [3/3] . . . 289
11.6.2.14 screen_xywh() [1/4] . . . 290
11.6.2.15 screen_xywh() [2/4] . . . 290
11.6.2.16 screen_xywh() [3/4] . . . 290
11.6.2.17 screen_xywh() [4/4] . . . 291
11.7 Color & Font functions . . . 291
11.7.1 Detailed Description . . . 293
11.7.2 Function Documentation . . . 293
11.7.2.1 fl_color() [1/3] . . . 293
11.7.2.2 fl_color() [2/3] . . . 293
11.7.2.3 fl_color() [3/3] . . . 294
11.7.2.4 fl_color_average() . . . 294

11.7.2.5 fl_contrast() . . . 294
11.7.2.6 fl_contrast_function() . . . 296
11.7.2.7 fl_contrast_level() [1/2] . . . 296
11.7.2.8 fl_contrast_level() [2/2] . . . 297
11.7.2.9 fl_contrast_mode() [1/2] . . . 297
11.7.2.10 fl_contrast_mode() [2/2] . . . 298
11.7.2.11 fl_descent() . . . 298
11.7.2.12 fl_font() [1/2] . . . 299
11.7.2.13 fl_font() [2/2] . . . 299
11.7.2.14 fl_height() [1/2] . . . 299
11.7.2.15 fl_height() [2/2] . . . 299
11.7.2.16 fl_latin1_to_local() . . . 299
11.7.2.17 fl_lightness() . . . 300
11.7.2.18 fl_local_to_latin1() . . . 300
11.7.2.19 fl_local_to_mac_roman() . . . 300
11.7.2.20 fl_luminance() . . . 301
11.7.2.21 fl_mac_roman_to_local() . . . 301
11.7.2.22 fl_show_colormap() . . . 301
11.7.2.23 fl_size() . . . 302
11.7.2.24 fl_text_extents() [1/2] . . . 302
11.7.2.25 fl_text_extents() [2/2] . . . 303
11.7.2.26 fl_width() [1/3] . . . 303
11.7.2.27 fl_width() [2/3] . . . 303
11.7.2.28 fl_width() [3/3] . . . 304
11.7.2.29 free_color() . . . 304
11.7.2.30 get_color() [1/3] . . . 304
11.7.2.31 get_color() [2/3] . . . 304
11.7.2.32 get_color() [3/3] . . . 305
11.7.2.33 get_font() . . . 305
11.7.2.34 get_font_name() . . . 305
11.7.2.35 get_font_sizes() . . . 305
11.7.2.36 set_color() [1/3] . . . 305
11.7.2.37 set_color() [2/3] . . . 305
11.7.2.38 set_color() [3/3] . . . 306
11.7.2.39 set_font() . . . 306
11.7.2.40 set_fonts() . . . 306
11.8 Drawing functions . . . 307
11.8.1 Detailed Description . . . 312
11.8.2 Enumeration Type Documentation . . . 312
11.8.2.1 anonymous enum . . . 312
11.8.3 Function Documentation . . . 313
11.8.3.1 fl_add_symbol() [1/2] . . . 313

11.8.3.2 fl_add_symbol() [2/2] . . . 313
11.8.3.3 fl_antialias() . . . 314
11.8.3.4 fl_arc() [1/2] . . . 314
11.8.3.5 fl_arc() [2/2] . . . 315
11.8.3.6 fl_begin_complex_polygon() . . . 315
11.8.3.7 fl_begin_offscreen() . . . 315
11.8.3.8 fl_begin_points() . . . 316
11.8.3.9 fl_can_do_alpha_blending() . . . 316
11.8.3.10 fl_capture_window() . . . 316
11.8.3.11 fl_circle() . . . 316
11.8.3.12 fl_clip_box() . . . 317
11.8.3.13 fl_clip_region() [1/2] . . . 318
11.8.3.14 fl_clip_region() [2/2] . . . 318
11.8.3.15 fl_copy_offscreen() . . . 318
11.8.3.16 fl_create_offscreen() . . . 319
11.8.3.17 fl_cursor() . . . 319
11.8.3.18 fl_curve() . . . 319
11.8.3.19 fl_delete_offscreen() . . . 320
11.8.3.20 fl_draw() [1/5] . . . 320
11.8.3.21 fl_draw() [2/5] . . . 320
11.8.3.22 fl_draw() [3/5] . . . 321
11.8.3.23 fl_draw() [4/5] . . . 321
11.8.3.24 fl_draw() [5/5] . . . 321
11.8.3.25 fl_draw_arrow() . . . 321
11.8.3.26 fl_draw_box() . . . 322
11.8.3.27 fl_draw_box_focus() . . . 322
11.8.3.28 fl_draw_check() . . . 323
11.8.3.29 fl_draw_circle() . . . 323
11.8.3.30 fl_draw_image() [1/2] . . . 323
11.8.3.31 fl_draw_image() [2/2] . . . 324
11.8.3.32 fl_draw_image_mono() [1/2] . . . 325
11.8.3.33 fl_draw_image_mono() [2/2] . . . 325
11.8.3.34 fl_draw_pixmap() [1/2] . . . 325
11.8.3.35 fl_draw_pixmap() [2/2] . . . 325
11.8.3.36 fl_draw_radio() . . . 326
11.8.3.37 fl_draw_symbol() . . . 326
11.8.3.38 fl_expand_text() . . . 327
11.8.3.39 fl_focus_rect() . . . 327
11.8.3.40 fl_frame() . . . 327
11.8.3.41 fl_frame2() . . . 328
11.8.3.42 fl_gap() . . . 328
11.8.3.43 fl_line_style() . . . 328

11.8.3.44 fl_load_matrix() . . . 329
11.8.3.45 fl_measure() . . . 330
11.8.3.46 fl_measure_pixmap() [1/2] . . . 330
11.8.3.47 fl_measure_pixmap() [2/2] . . . 330
11.8.3.48 fl_mult_matrix() . . . 331
11.8.3.49 fl_not_clipped() . . . 331
11.8.3.50 fl_old_shortcut() . . . 331
11.8.3.51 fl_overlay_clear() . . . 332
11.8.3.52 fl_overlay_rect() . . . 332
11.8.3.53 fl_override_scale() . . . 333
11.8.3.54 fl_pie() . . . 333
11.8.3.55 fl_polygon() . . . 334
11.8.3.56 fl_pop_clip() . . . 334
11.8.3.57 fl_push_clip() . . . 334
11.8.3.58 fl_push_matrix() . . . 334
11.8.3.59 fl_read_image() . . . 335
11.8.3.60 fl_rect() [1/3] . . . 335
11.8.3.61 fl_rect() [2/3] . . . 335
11.8.3.62 fl_rect() [3/3] . . . 335
11.8.3.63 fl_rectf() [1/4] . . . 336
11.8.3.64 fl_rectf() [2/4] . . . 336
11.8.3.65 fl_rectf() [3/4] . . . 336
11.8.3.66 fl_rectf() [4/4] . . . 336
11.8.3.67 fl_remove_symbol() . . . 337
11.8.3.68 fl_rescale_offscreen() . . . 337
11.8.3.69 fl_reset_spot() . . . 337
11.8.3.70 fl_restore_scale() . . . 337
11.8.3.71 fl_rotate() . . . 337
11.8.3.72 fl_rounded_rect() . . . 338
11.8.3.73 fl_rounded_rectf() . . . 338
11.8.3.74 fl_scale() [1/2] . . . 338
11.8.3.75 fl_scale() [2/2] . . . 338
11.8.3.76 fl_scroll() . . . 338
11.8.3.77 fl_set_spot() . . . 339
11.8.3.78 fl_set_status() . . . 339
11.8.3.79 fl_shortcut_label() [1/2] . . . 339
11.8.3.80 fl_shortcut_label() [2/2] . . . 340
11.8.3.81 fl_transform_dx() . . . 341
11.8.3.82 fl_transform_dy() . . . 341
11.8.3.83 fl_transform_x() . . . 341
11.8.3.84 fl_transform_y() . . . 341
11.8.3.85 fl_transformed_vertex() . . . 342

11.8.3.86 fl_translate() . . . 342
11.8.3.87 fl_vertex() . . . 342
11.9 Multithreading support functions . . . 342
11.9.1 Detailed Description . . . 343
11.9.2 Function Documentation . . . 343
11.9.2.1 awake() [1/3] . . . 343
11.9.2.2 awake() [2/3] . . . 343
11.9.2.3 awake() [3/3] . . . 343
11.9.2.4 awake_once() . . . 344
11.9.2.5 lock() . . . 344
11.9.2.6 thread_message() . . . 345
11.9.2.7 unlock() . . . 345
11.10 Cairo Support Functions and Classes . . . 345
11.10.1 Detailed Description . . . 346
11.10.2 Function Documentation . . . 346
11.10.2.1 cairo_autolink_context() [1/2] . . . 346
11.10.2.2 cairo_autolink_context() [2/2] . . . 346
11.10.2.3 cairo_cc() . . . 346
11.10.2.4 cairo_flush() . . . 346
11.10.2.5 cairo_make_current() . . . 347
11.11 Unicode and UTF-8 functions . . . 347
11.11.1 Detailed Description . . . 350
11.11.2 Macro Definition Documentation . . . 350
11.11.2.1 ERRORS_TO_CP1252 . . . 350
11.11.2.2 ERRORS_TO_ISO8859_1 . . . 350
11.11.2.3 STRICT_RFC3629 . . . 350
11.11.3 Function Documentation . . . 350
11.11.3.1 fl_access() . . . 350
11.11.3.2 fl_chdir() . . . 351
11.11.3.3 fl_chmod() . . . 351
11.11.3.4 fl_close_fd() . . . 351
11.11.3.5 fl_execvp() . . . 352
11.11.3.6 fl_fopen() . . . 352
11.11.3.7 fl_getcwd() . . . 352
11.11.3.8 fl_getenv() . . . 353
11.11.3.9 fl_make_path() . . . 353
11.11.3.10 fl_make_path_for_file() . . . 354
11.11.3.11 fl_mkdir() . . . 354
11.11.3.12 fl_nonspacing() . . . 354
11.11.3.13 fl_open() . . . 355
11.11.3.14 fl_open_ext() . . . 355
11.11.3.15 fl_putenv() . . . 356

11.11.3.16 fl_rename() . . . 356
11.11.3.17 fl_rmdir() . . . 356
11.11.3.18 fl_stat() . . . 357
11.11.3.19 fl_system() . . . 357
11.11.3.20 fl_tolower() . . . 357
11.11.3.21 fl_toupper() . . . 358
11.11.3.22 fl_ucs_to_Utf16() . . . 358
11.11.3.23 fl_unlink() . . . 359
11.11.3.24 fl_utf2mbcs() . . . 359
11.11.3.25 fl_utf8_clean() . . . 359
11.11.3.26 fl_utf8_next_composed_char() . . . 360
11.11.3.27 fl_utf8_previous_composed_char() . . . 360
11.11.3.28 fl_utf8back() . . . 361
11.11.3.29 fl_utf8bytes() . . . 361
11.11.3.30 fl_utf8decode() . . . 361
11.11.3.31 fl_utf8encode() . . . 362
11.11.3.32 fl_utf8from_mb() . . . 362
11.11.3.33 fl_utf8froma() . . . 363
11.11.3.34 fl_utf8fromwc() . . . 363
11.11.3.35 fl_utf8fwd() . . . 364
11.11.3.36 fl_utf8len() . . . 364
11.11.3.37 fl_utf8len1() . . . 365
11.11.3.38 fl_utf8locale() . . . 365
11.11.3.39 fl_utf8strlen() . . . 366
11.11.3.40 fl_utf8test() . . . 366
11.11.3.41 fl_utf8to_mb() . . . 367
11.11.3.42 fl_utf8toa() . . . 367
11.11.3.43 fl_utf8toUtf16() . . . 367
11.11.3.44 fl_utf8towc() . . . 368
11.11.3.45 fl_utf_nb_char() . . . 369
11.11.3.46 fl_utf_strcasecmp() . . . 369
11.11.3.47 fl_utf_strncasecmp() . . . 369
11.11.3.48 fl_utf_tolower() . . . 370
11.11.3.49 fl_utf_toupper() . . . 371
11.11.3.50 fl_wcwidth() . . . 371
11.11.3.51 fl_wcwidth_() . . . 371
11.12 String handling functions . . . 372
11.12.1 Detailed Description . . . 372
11.12.2 Function Documentation . . . 372
11.12.2.1 fl_strdup() . . . 372
11.13 Safe widget deletion support functions . . . 372
11.13.1 Detailed Description . . . 373

11.13.2 Function Documentation . . . 373
11.13.2.1 clear_widget_pointer() . . . 373
11.13.2.2 delete_widget() . . . 374
11.13.2.3 do_widget_deletion() . . . 374
11.13.2.4 release_widget_pointer() . . . 374
11.13.2.5 watch_widget_pointer() . . . 375
11.14 Mac OS X-specific symbols . . . 375
11.14.1 Detailed Description . . . 376
11.14.2 Function Documentation . . . 376
11.14.2.1 fl_mac_set_about() . . . 376
11.14.2.2 fl_open_callback() . . . 376
11.14.3 Variable Documentation . . . 376
11.14.3.1 fl_mac_os_version . . . 376
11.15 Common Dialog Classes and Functions . . . 376
11.15.1 Detailed Description . . . 378
11.15.2 Function Documentation . . . 378
11.15.2.1 fl_alert() . . . 378
11.15.2.2 fl_beep() . . . 379
11.15.2.3 fl_choice() . . . 379
11.15.2.4 fl_choice_n() . . . 380
11.15.2.5 fl_color_chooser() [1/2] . . . 381
11.15.2.6 fl_color_chooser() [2/2] . . . 382
11.15.2.7 fl_dir_chooser() . . . 383
11.15.2.8 fl_file_chooser() . . . 383
11.15.2.9 fl_file_chooser_callback() . . . 384
11.15.2.10 fl_file_chooser_ok_label() . . . 384
11.15.2.11 fl_input() [1/2] . . . 385
11.15.2.12 fl_input() [2/2] . . . 385
11.15.2.13 fl_input_str() [1/2] . . . 386
11.15.2.14 fl_input_str() [2/2] . . . 386
11.15.2.15 fl_message() . . . 387
11.15.2.16 fl_message_hotspot() [1/2] . . . 387
11.15.2.17 fl_message_hotspot() [2/2] . . . 387
11.15.2.18 fl_message_icon() . . . 387
11.15.2.19 fl_message_icon_label() . . . 388
11.15.2.20 fl_message_position() [1/3] . . . 388
11.15.2.21 fl_message_position() [2/3] . . . 389
11.15.2.22 fl_message_position() [3/3] . . . 389
11.15.2.23 fl_message_title() . . . 390
11.15.2.24 fl_message_title_default() . . . 390
11.15.2.25 fl_password() [1/2] . . . 390
11.15.2.26 fl_password() [2/2] . . . 391

11.15.2.27 fl_password_str() [1/2] . . . 391
11.15.2.28 fl_password_str() [2/2] . . . 392
11.15.3 Variable Documentation . . . 392
11.15.3.1 error . . . 392
11.15.3.2 fatal . . . 392
11.15.3.3 warning . . . 393
11.16 File names and URI utility functions . . . 393
11.16.1 Detailed Description . . . 394
11.16.2 Typedef Documentation . . . 394
11.16.2.1 Fl_File_Sort_F . . . 394
11.16.3 Function Documentation . . . 395
11.16.3.1 fl_decode_uri() . . . 395
11.16.3.2 fl_filename_absolute() [1/2] . . . 395
11.16.3.3 fl_filename_absolute() [2/2] . . . 395
11.16.3.4 fl_filename_absolute_str() [1/2] . . . 396
11.16.3.5 fl_filename_absolute_str() [2/2] . . . 396
11.16.3.6 fl_filename_expand() . . . 396
11.16.3.7 fl_filename_expand_str() . . . 397
11.16.3.8 fl_filename_ext() . . . 397
11.16.3.9 fl_filename_ext_str() . . . 397
11.16.3.10 fl_filename_free_list() . . . 398
11.16.3.11 fl_filename_isdir() . . . 398
11.16.3.12 fl_filename_list() . . . 398
11.16.3.13 fl_filename_match() . . . 399
11.16.3.14 fl_filename_name() . . . 400
11.16.3.15 fl_filename_name_str() . . . 400
11.16.3.16 fl_filename_path_str() . . . 400
11.16.3.17 fl_filename_relative() [1/2] . . . 401
11.16.3.18 fl_filename_relative() [2/2] . . . 401
11.16.3.19 fl_filename_relative_str() [1/2] . . . 402
11.16.3.20 fl_filename_relative_str() [2/2] . . . 402
11.16.3.21 fl_filename_setext() . . . 402
11.16.3.22 fl_filename_setext_str() . . . 403
11.16.3.23 fl_getcwd_str() . . . 403
11.16.3.24 fl_open_uri() . . . 403

12 Namespace Documentation **405**
12.1 Fl Namespace Reference . . . 405
12.1.1 Detailed Description . . . 414
12.1.2 Enumeration Type Documentation . . . 415
12.1.2.1 Fl_Option . . . 415
12.1.3 Function Documentation . . . 416

12.1.3.1 abi_check() . . . 416
12.1.3.2 abi_version() . . . 416
12.1.3.3 add_check() . . . 416
12.1.3.4 add_fd() [1/2] . . . 416
12.1.3.5 add_fd() [2/2] . . . 417
12.1.3.6 add_idle() . . . 417
12.1.3.7 add_timeout() . . . 417
12.1.3.8 api_version() . . . 418
12.1.3.9 arg() . . . 418
12.1.3.10 args() [1/2] . . . 419
12.1.3.11 args() [2/2] . . . 419
12.1.3.12 args_to_utf8() . . . 420
12.1.3.13 background() . . . 421
12.1.3.14 background2() . . . 421
12.1.3.15 box_bg() . . . 421
12.1.3.16 box_border_radius_max() [1/2] . . . 422
12.1.3.17 box_border_radius_max() [2/2] . . . 422
12.1.3.18 box_color() . . . 422
12.1.3.19 box_dh() . . . 422
12.1.3.20 box_dw() . . . 422
12.1.3.21 box_dx() . . . 423
12.1.3.22 box_dy() . . . 423
12.1.3.23 box_shadow_width() [1/2] . . . 423
12.1.3.24 box_shadow_width() [2/2] . . . 423
12.1.3.25 check() . . . 423
12.1.3.26 display() . . . 424
12.1.3.27 dnd_text_ops() [1/2] . . . 424
12.1.3.28 dnd_text_ops() [2/2] . . . 424
12.1.3.29 draw_box_active() . . . 424
12.1.3.30 draw_GL_text_with_textures() [1/2] . . . 424
12.1.3.31 draw_GL_text_with_textures() [2/2] . . . 424
12.1.3.32 flush() . . . 425
12.1.3.33 get_system_colors() . . . 425
12.1.3.34 gl_visual() . . . 425
12.1.3.35 has_idle() . . . 425
12.1.3.36 has_timeout() . . . 426
12.1.3.37 hide_all_windows() . . . 426
12.1.3.38 idle() . . . 427
12.1.3.39 is_scheme() . . . 427
12.1.3.40 menu_linespacing() [1/2] . . . 427
12.1.3.41 menu_linespacing() [2/2] . . . 428
12.1.3.42 now() . . . 428

12.1.3.43 option() [1/2] . . . 428
12.1.3.44 option() [2/2] . . . 429
12.1.3.45 own_colormap() . . . 430
12.1.3.46 program_should_quit() [1/2] . . . 430
12.1.3.47 program_should_quit() [2/2] . . . 430
12.1.3.48 readqueue() . . . 430
12.1.3.49 ready() . . . 430
12.1.3.50 reload_scheme() . . . 431
12.1.3.51 remove_check() . . . 431
12.1.3.52 remove_idle() . . . 431
12.1.3.53 remove_next_timeout() . . . 431
12.1.3.54 remove_timeout() . . . 432
12.1.3.55 repeat_timeout() . . . 433
12.1.3.56 run() . . . 433
12.1.3.57 scheme() . . . 433
12.1.3.58 scrollbar_size() [1/2] . . . 434
12.1.3.59 scrollbar_size() [2/2] . . . 434
12.1.3.60 seconds_between() . . . 434
12.1.3.61 seconds_since() . . . 435
12.1.3.62 set_box_color() . . . 435
12.1.3.63 set_boxtype() . . . 435
12.1.3.64 ticks_between() . . . 436
12.1.3.65 ticks_since() . . . 436
12.1.3.66 timeout_list() . . . 437
12.1.3.67 use_high_res_GL() [1/2] . . . 437
12.1.3.68 use_high_res_GL() [2/2] . . . 437
12.1.3.69 version() . . . 437
12.1.3.70 visible_focus() [1/2] . . . 437
12.1.3.71 visible_focus() [2/2] . . . 437
12.1.3.72 visual() . . . 438
12.1.3.73 wait() [1/2] . . . 438
12.1.3.74 wait() [2/2] . . . 438
12.1.4 Variable Documentation . . . 438
12.1.4.1 help . . . 438
12.2 Fl::Pen Namespace Reference . . . 438
12.2.1 Detailed Description . . . 440

13 Class Documentation **441**
13.1 Fl_Grid::Cell Class Reference . . . 441
13.1.1 Constructor & Destructor Documentation . . . 441
13.1.1.1 ~Cell() . . . 441
13.1.2 Member Function Documentation . . . 442

13.1.2.1 next() . . . 442
13.2 Fl_Terminal::CharStyle Class Reference . . . 442
13.3 Fl_GIF_Image::GIF_FRAME::CPAL Struct Reference . . . 443
13.4 Fl_Terminal::Cursor Class Reference . . . 443
13.5 Fl_Shared_Image::Deleter Class Reference . . . 443
13.5.1 Detailed Description . . . 444
13.6 Fl_Preferences::Entry Struct Reference . . . 444
13.7 Fl_Terminal::EscapeSeq Class Reference . . . 444
13.8 Fl_Adjuster Class Reference . . . 444
13.8.1 Detailed Description . . . 452
13.8.2 Constructor & Destructor Documentation . . . 453
13.8.2.1 Fl_Adjuster() . . . 453
13.8.3 Member Function Documentation . . . 453
13.8.3.1 draw() . . . 453
13.8.3.2 handle() . . . 453
13.8.3.3 soft() [1/2] . . . 454
13.8.3.4 soft() [2/2] . . . 454
13.8.3.5 value_damage() . . . 454
13.9 Fl_Anim_GIF_Image Class Reference . . . 454
13.9.1 Detailed Description . . . 460
13.9.2 Member Enumeration Documentation . . . 460
13.9.2.1 Flags . . . 460
13.9.3 Constructor & Destructor Documentation . . . 460
13.9.3.1 Fl_Anim_GIF_Image() [1/2] . . . 460
13.9.3.2 Fl_Anim_GIF_Image() [2/2] . . . 461
13.9.3.3 ~Fl_Anim_GIF_Image() . . . 461
13.9.4 Member Function Documentation . . . 461
13.9.4.1 canvas() [1/2] . . . 461
13.9.4.2 canvas() [2/2] . . . 462
13.9.4.3 canvas_h() . . . 462
13.9.4.4 canvas_w() . . . 462
13.9.4.5 color_average() . . . 462
13.9.4.6 copy() . . . 462
13.9.4.7 delay() [1/2] . . . 463
13.9.4.8 delay() [2/2] . . . 463
13.9.4.9 desaturate() . . . 463
13.9.4.10 draw() . . . 463
13.9.4.11 frame() [1/2] . . . 464
13.9.4.12 frame() [2/2] . . . 464
13.9.4.13 frame_count() . . . 464
13.9.4.14 frame_h() . . . 464
13.9.4.15 frame_uncache() [1/2] . . . 465

13.9.4.16 frame_uncache() [2/2] . . . 465
13.9.4.17 frame_w() . . . 465
13.9.4.18 frame_x() . . . 465
13.9.4.19 frame_y() . . . 466
13.9.4.20 frames() . . . 466
13.9.4.21 image() [1/2] . . . 466
13.9.4.22 image() [2/2] . . . 466
13.9.4.23 is_animated() . . . 466
13.9.4.24 load() . . . 467
13.9.4.25 name() . . . 467
13.9.4.26 next() . . . 467
13.9.4.27 on_extension_data() . . . 467
13.9.4.28 on_frame_data() . . . 467
13.9.4.29 playing() . . . 468
13.9.4.30 resize() [1/2] . . . 468
13.9.4.31 resize() [2/2] . . . 468
13.9.4.32 speed() [1/2] . . . 468
13.9.4.33 speed() [2/2] . . . 468
13.9.4.34 start() . . . 468
13.9.4.35 stop() . . . 469
13.9.4.36 uncache() . . . 469
13.9.4.37 valid() . . . 469
13.9.5 Member Data Documentation . . . 469
13.9.5.1 loop . . . 469
13.9.5.2 min_delay . . . 469
13.10 Fl_Bitmap Class Reference . . . 469
13.10.1 Detailed Description . . . 472
13.10.2 Constructor & Destructor Documentation . . . 472
13.10.2.1 Fl_Bitmap() [1/4] . . . 472
13.10.2.2 Fl_Bitmap() [2/4] . . . 472
13.10.2.3 Fl_Bitmap() [3/4] . . . 473
13.10.2.4 Fl_Bitmap() [4/4] . . . 473
13.10.3 Member Function Documentation . . . 473
13.10.3.1 copy() . . . 473
13.10.3.2 draw() . . . 474
13.10.3.3 label() [1/2] . . . 474
13.10.3.4 label() [2/2] . . . 475
13.10.3.5 uncache() . . . 475
13.11 Fl_BMP_Image Class Reference . . . 475
13.11.1 Detailed Description . . . 478
13.11.2 Constructor & Destructor Documentation . . . 478
13.11.2.1 Fl_BMP_Image() [1/2] . . . 478

13.11.2.2 Fl_BMP_Image() [2/2] . . . 478
13.12 Fl_Box Class Reference . . . 479
13.12.1 Detailed Description . . . 486
13.12.2 Constructor & Destructor Documentation . . . 486
13.12.2.1 Fl_Box() [1/2] . . . 486
13.12.2.2 Fl_Box() [2/2] . . . 486
13.12.3 Member Function Documentation . . . 487
13.12.3.1 draw() . . . 487
13.12.3.2 handle() . . . 487
13.13 Fl_Browser Class Reference . . . 488
13.13.1 Detailed Description . . . 501
13.13.2 Constructor & Destructor Documentation . . . 502
13.13.2.1 Fl_Browser() . . . 502
13.13.3 Member Function Documentation . . . 502
13.13.3.1 _remove() . . . 502
13.13.3.2 add() . . . 503
13.13.3.3 bline_length() . . . 503
13.13.3.4 bottomline() . . . 503
13.13.3.5 clear() . . . 503
13.13.3.6 column_char() [1/2] . . . 503
13.13.3.7 column_char() [2/2] . . . 504
13.13.3.8 column_widths() [1/2] . . . 504
13.13.3.9 column_widths() [2/2] . . . 504
13.13.3.10 data() [1/2] . . . 504
13.13.3.11 data() [2/2] . . . 505
13.13.3.12 display() . . . 505
13.13.3.13 displayed() . . . 505
13.13.3.14 find_line() . . . 505
13.13.3.15 format_char() [1/2] . . . 506
13.13.3.16 format_char() [2/2] . . . 507
13.13.3.17 full_height() . . . 507
13.13.3.18 hide() [1/2] . . . 507
13.13.3.19 hide() [2/2] . . . 507
13.13.3.20 icon() [1/2] . . . 507
13.13.3.21 icon() [2/2] . . . 508
13.13.3.22 incr_height() . . . 508
13.13.3.23 insert() [1/2] . . . 508
13.13.3.24 insert() [2/2] . . . 509
13.13.3.25 item_at() . . . 509
13.13.3.26 item_draw() . . . 509
13.13.3.27 item_first() . . . 509
13.13.3.28 item_height() . . . 510

13.13.3.29 item_last() . . . 510
13.13.3.30 item_next() . . . 510
13.13.3.31 item_prev() . . . 511
13.13.3.32 item_select() . . . 511
13.13.3.33 item_selected() . . . 511
13.13.3.34 item_swap() . . . 512
13.13.3.35 item_text() . . . 512
13.13.3.36 item_width() . . . 512
13.13.3.37 lineno() . . . 513
13.13.3.38 lineposition() . . . 513
13.13.3.39 load() . . . 513
13.13.3.40 make_visible() . . . 514
13.13.3.41 middleline() . . . 514
13.13.3.42 move() . . . 514
13.13.3.43 remove() . . . 514
13.13.3.44 remove_icon() . . . 515
13.13.3.45 select() . . . 515
13.13.3.46 selected() . . . 515
13.13.3.47 show() [1/2] . . . 516
13.13.3.48 show() [2/2] . . . 516
13.13.3.49 size() . . . 516
13.13.3.50 swap() [1/2] . . . 516
13.13.3.51 swap() [2/2] . . . 516
13.13.3.52 text() [1/2] . . . 517
13.13.3.53 text() [2/2] . . . 517
13.13.3.54 textsize() . . . 517
13.13.3.55 topline() [1/2] . . . 518
13.13.3.56 topline() [2/2] . . . 518
13.13.3.57 value() [1/2] . . . 518
13.13.3.58 value() [2/2] . . . 518
13.13.3.59 visible() . . . 518
13.14 Fl_Browser_ Class Reference . . . 519
13.14.1 Detailed Description . . . 530
13.14.2 Member Enumeration Documentation . . . 530
13.14.2.1 anonymous enum . . . 530
13.14.3 Constructor & Destructor Documentation . . . 531
13.14.3.1 Fl_Browser_() . . . 531
13.14.4 Member Function Documentation . . . 531
13.14.4.1 bbox() . . . 531
13.14.4.2 deleting() . . . 531
13.14.4.3 deselect() . . . 531
13.14.4.4 display() . . . 532

13.14.4.5 displayed() . . . 532
13.14.4.6 draw() . . . 532
13.14.4.7 find_item() . . . 532
13.14.4.8 full_height() . . . 533
13.14.4.9 full_width() . . . 533
13.14.4.10 handle() . . . 533
13.14.4.11 has_scrollbar() . . . 533
13.14.4.12 hposition() [1/2] . . . 534
13.14.4.13 hposition() [2/2] . . . 534
13.14.4.14 incr_height() . . . 534
13.14.4.15 inserting() . . . 534
13.14.4.16 item_at() . . . 534
13.14.4.17 item_draw() . . . 535
13.14.4.18 item_first() . . . 535
13.14.4.19 item_height() . . . 535
13.14.4.20 item_last() . . . 535
13.14.4.21 item_next() . . . 536
13.14.4.22 item_prev() . . . 536
13.14.4.23 item_quick_height() . . . 536
13.14.4.24 item_select() . . . 536
13.14.4.25 item_selected() . . . 536
13.14.4.26 item_swap() . . . 537
13.14.4.27 item_text() . . . 537
13.14.4.28 item_width() . . . 537
13.14.4.29 leftedge() . . . 538
13.14.4.30 linespacing() [1/2] . . . 538
13.14.4.31 linespacing() [2/2] . . . 538
13.14.4.32 new_list() . . . 538
13.14.4.33 position() [1/2] . . . 538
13.14.4.34 position() [2/2] . . . 538
13.14.4.35 redraw_line() . . . 539
13.14.4.36 redraw_lines() . . . 539
13.14.4.37 replacing() . . . 539
13.14.4.38 resize() . . . 539
13.14.4.39 scrollbar_left() . . . 539
13.14.4.40 scrollbar_right() . . . 540
13.14.4.41 scrollbar_size() [1/2] . . . 540
13.14.4.42 scrollbar_size() [2/2] . . . 540
13.14.4.43 select() . . . 540
13.14.4.44 select_only() . . . 541
13.14.4.45 selection() . . . 541
13.14.4.46 sort() . . . 541

13.14.4.47 swapping() . . . 541
13.14.4.48 textfont() . . . 542
13.14.4.49 vposition() [1/2] . . . 542
13.14.4.50 vposition() [2/2] . . . 542
13.14.5 Member Data Documentation . . . 542
13.14.5.1 hscrollbar . . . 542
13.14.5.2 scrollbar . . . 542
13.15 Fl_Button Class Reference . . . 543
13.15.1 Detailed Description . . . 550
13.15.2 Constructor & Destructor Documentation . . . 550
13.15.2.1 Fl_Button() . . . 550
13.15.3 Member Function Documentation . . . 551
13.15.3.1 clear() . . . 551
13.15.3.2 compact() [1/2] . . . 551
13.15.3.3 compact() [2/2] . . . 551
13.15.3.4 down_box() [1/2] . . . 552
13.15.3.5 down_box() [2/2] . . . 552
13.15.3.6 draw() . . . 553
13.15.3.7 handle() . . . 553
13.15.3.8 set() . . . 553
13.15.3.9 shortcut() [1/2] . . . 554
13.15.3.10 shortcut() [2/2] . . . 554
13.15.3.11 value() . . . 554
13.16 Fl_Cairo_State Class Reference . . . 554
13.16.1 Detailed Description . . . 555
13.16.2 Member Function Documentation . . . 555
13.16.2.1 cc() . . . 555
13.17 Fl_Cairo_Window Class Reference . . . 555
13.17.1 Detailed Description . . . 568
13.17.2 Member Function Documentation . . . 569
13.17.2.1 draw() . . . 569
13.17.2.2 set_draw_cb() . . . 569
13.18 Fl_Callback_User_Data Class Reference . . . 569
13.18.1 Detailed Description . . . 569
13.19 Fl_Chart Class Reference . . . 570
13.19.1 Detailed Description . . . 577
13.19.2 Constructor & Destructor Documentation . . . 578
13.19.2.1 Fl_Chart() . . . 578
13.19.3 Member Function Documentation . . . 578
13.19.3.1 add() . . . 578
13.19.3.2 autosize() [1/2] . . . 578
13.19.3.3 autosize() [2/2] . . . 578

13.19.3.4 bounds() [1/2] . . . 579
13.19.3.5 bounds() [2/2] . . . 579
13.19.3.6 draw() . . . 579
13.19.3.7 draw_barchart() . . . 579
13.19.3.8 draw_horbarchart() . . . 580
13.19.3.9 draw_linechart() . . . 580
13.19.3.10 draw_piechart() . . . 581
13.19.3.11 insert() . . . 581
13.19.3.12 maxsize() . . . 581
13.19.3.13 replace() . . . 582
13.19.3.14 size() . . . 582
13.20 FL_CHART_ENTRY Struct Reference . . . 582
13.20.1 Detailed Description . . . 582
13.21 Fl_Check_Browser Class Reference . . . 583
13.21.1 Detailed Description . . . 594
13.21.2 Member Function Documentation . . . 594
13.21.2.1 add() [1/2] . . . 594
13.21.2.2 add() [2/2] . . . 594
13.21.2.3 handle() . . . 595
13.21.2.4 item_at() . . . 595
13.21.2.5 item_draw() . . . 595
13.21.2.6 item_first() . . . 595
13.21.2.7 item_height() . . . 595
13.21.2.8 item_next() . . . 596
13.21.2.9 item_prev() . . . 596
13.21.2.10 item_select() . . . 596
13.21.2.11 item_selected() . . . 596
13.21.2.12 item_swap() . . . 597
13.21.2.13 item_text() . . . 597
13.21.2.14 item_width() . . . 597
13.21.2.15 nitems() . . . 597
13.21.2.16 remove() . . . 598
13.22 Fl_Check_Button Class Reference . . . 598
13.22.1 Detailed Description . . . 605
13.22.2 Constructor & Destructor Documentation . . . 606
13.22.2.1 Fl_Check_Button() . . . 606
13.23 Fl_Choice Class Reference . . . 606
13.23.1 Detailed Description . . . 615
13.23.2 Constructor & Destructor Documentation . . . 616
13.23.2.1 Fl_Choice() . . . 616
13.23.3 Member Function Documentation . . . 617
13.23.3.1 draw() . . . 617

13.23.3.2 handle() . . . 617
13.23.3.3 value() [1/3] . . . 618
13.23.3.4 value() [2/3] . . . 618
13.23.3.5 value() [3/3] . . . 618
13.24 Fl_Clock Class Reference . . . 618
13.24.1 Detailed Description . . . 625
13.24.2 Constructor & Destructor Documentation . . . 626
13.24.2.1 Fl_Clock() [1/2] . . . 626
13.24.2.2 Fl_Clock() [2/2] . . . 626
13.24.3 Member Function Documentation . . . 627
13.24.3.1 handle() . . . 627
13.25 Fl_Clock_Output Class Reference . . . 627
13.25.1 Detailed Description . . . 635
13.25.2 Constructor & Destructor Documentation . . . 635
13.25.2.1 Fl_Clock_Output() . . . 635
13.25.3 Member Function Documentation . . . 636
13.25.3.1 draw() [1/2] . . . 636
13.25.3.2 draw() [2/2] . . . 636
13.25.3.3 hour() . . . 636
13.25.3.4 minute() . . . 636
13.25.3.5 second() . . . 636
13.25.3.6 shadow() [1/2] . . . 636
13.25.3.7 shadow() [2/2] . . . 637
13.25.3.8 value() [1/3] . . . 637
13.25.3.9 value() [2/3] . . . 637
13.25.3.10 value() [3/3] . . . 637
13.26 Fl_Color_Chooser Class Reference . . . 638
13.26.1 Detailed Description . . . 647
13.26.2 Constructor & Destructor Documentation . . . 647
13.26.2.1 Fl_Color_Chooser() . . . 647
13.26.3 Member Function Documentation . . . 648
13.26.3.1 b() . . . 648
13.26.3.2 g() . . . 648
13.26.3.3 handle() . . . 648
13.26.3.4 hsv() . . . 649
13.26.3.5 hsv2rgb() . . . 649
13.26.3.6 hue() . . . 649
13.26.3.7 mode() [1/2] . . . 649
13.26.3.8 mode() [2/2] . . . 649
13.26.3.9 r() . . . 650
13.26.3.10 rgb() . . . 650
13.26.3.11 rgb2hsv() . . . 650

13.26.3.12 saturation() . . . 650
13.26.3.13 value() . . . 650
13.27 Fl_Copy_Surface Class Reference . . . 651
13.27.1 Detailed Description . . . 652
13.27.2 Constructor & Destructor Documentation . . . 653
13.27.2.1 Fl_Copy_Surface() . . . 653
13.27.3 Member Function Documentation . . . 653
13.27.3.1 is_current() . . . 653
13.27.3.2 origin() [1/2] . . . 653
13.27.3.3 origin() [2/2] . . . 653
13.27.3.4 printable_rect() . . . 654
13.27.3.5 set_current() . . . 654
13.27.3.6 translate() . . . 654
13.27.3.7 untranslate() . . . 654
13.28 Fl_Counter Class Reference . . . 655
13.28.1 Detailed Description . . . 663
13.28.2 Constructor & Destructor Documentation . . . 663
13.28.2.1 Fl_Counter() . . . 663
13.28.3 Member Function Documentation . . . 664
13.28.3.1 arrow_widths() . . . 664
13.28.3.2 draw() . . . 664
13.28.3.3 handle() . . . 664
13.28.3.4 lstep() . . . 665
13.28.3.5 step() [1/2] . . . 665
13.28.3.6 step() [2/2] . . . 665
13.29 Fl_Device_Plugin Class Reference . . . 665
13.29.1 Detailed Description . . . 666
13.29.2 Member Function Documentation . . . 666
13.29.2.1 rectangle_capture() . . . 666
13.30 Fl_Dial Class Reference . . . 667
13.30.1 Detailed Description . . . 675
13.30.2 Constructor & Destructor Documentation . . . 675
13.30.2.1 Fl_Dial() . . . 675
13.30.3 Member Function Documentation . . . 675
13.30.3.1 angle1() . . . 675
13.30.3.2 draw() [1/2] . . . 676
13.30.3.3 draw() [2/2] . . . 676
13.30.3.4 handle() [1/2] . . . 676
13.30.3.5 handle() [2/2] . . . 676
13.31 Fl_Display_Device Class Reference . . . 676
13.31.1 Detailed Description . . . 677
13.32 Fl_Double_Window Class Reference . . . 677

13.32.1 Detailed Description . . . 690
13.32.2 Constructor & Destructor Documentation . . . 690
13.32.2.1 ~Fl_Double_Window() . . . 690
13.32.3 Member Function Documentation . . . 690
13.32.3.1 as_double_window() . . . 690
13.32.3.2 flush() . . . 690
13.32.3.3 hide() . . . 690
13.32.3.4 resize() . . . 691
13.32.3.5 show() . . . 691
13.33 Fl_End Class Reference . . . 691
13.33.1 Detailed Description . . . 692
13.34 Fl_EPS_File_Surface Class Reference . . . 692
13.34.1 Detailed Description . . . 694
13.34.2 Constructor & Destructor Documentation . . . 694
13.34.2.1 Fl_EPS_File_Surface() . . . 694
13.34.2.2 ~Fl_EPS_File_Surface() . . . 694
13.34.3 Member Function Documentation . . . 694
13.34.3.1 close() . . . 694
13.34.3.2 origin() [1/2] . . . 695
13.34.3.3 origin() [2/2] . . . 695
13.34.3.4 printable_rect() . . . 695
13.34.3.5 translate() . . . 695
13.34.3.6 untranslate() . . . 695
13.35 Fl_File_Browser Class Reference . . . 696
13.35.1 Detailed Description . . . 709
13.35.2 Constructor & Destructor Documentation . . . 710
13.35.2.1 Fl_File_Browser() . . . 710
13.35.3 Member Function Documentation . . . 710
13.35.3.1 errmsg() [1/2] . . . 710
13.35.3.2 errmsg() [2/2] . . . 710
13.35.3.3 filetype() [1/2] . . . 710
13.35.3.4 filetype() [2/2] . . . 710
13.35.3.5 filter() [1/2] . . . 710
13.35.3.6 filter() [2/2] . . . 710
13.35.3.7 iconsize() [1/2] . . . 710
13.35.3.8 iconsize() [2/2] . . . 711
13.35.3.9 load() . . . 711
13.36 Fl_File_Chooser Class Reference . . . 711
13.36.1 Detailed Description . . . 714
13.36.2 Member Enumeration Documentation . . . 716
13.36.2.1 Type . . . 716
13.36.3 Constructor & Destructor Documentation . . . 716

13.36.3.1 Fl_File_Chooser() . . . 716
13.36.4 Member Function Documentation . . . 716
13.36.4.1 add_extra() . . . 716
13.36.4.2 filter() . . . 717
13.36.4.3 iconsize() [1/2] . . . 717
13.36.4.4 iconsize() [2/2] . . . 717
13.36.4.5 preview() . . . 717
13.36.4.6 shown() . . . 717
13.36.4.7 value() [1/2] . . . 718
13.36.4.8 value() [2/2] . . . 718
13.36.5 Member Data Documentation . . . 718
13.36.5.1 showHiddenButton . . . 718
13.37 Fl_File_Icon Class Reference . . . 718
13.37.1 Detailed Description . . . 720
13.37.2 Constructor & Destructor Documentation . . . 720
13.37.2.1 Fl_File_Icon() . . . 720
13.37.3 Member Function Documentation . . . 720
13.37.3.1 add() . . . 720
13.37.3.2 add_color() . . . 720
13.37.3.3 add_vertex() [1/2] . . . 720
13.37.3.4 add_vertex() [2/2] . . . 721
13.37.3.5 draw() . . . 721
13.37.3.6 find() . . . 721
13.37.3.7 label() . . . 721
13.37.3.8 labeltype() . . . 722
13.37.3.9 load() . . . 722
13.37.3.10 load_fti() . . . 722
13.37.3.11 load_image() . . . 722
13.37.3.12 load_system_icons() . . . 723
13.37.3.13 next() . . . 723
13.37.3.14 type() . . . 723
13.38 Fl_File_Input Class Reference . . . 723
13.38.1 Detailed Description . . . 733
13.38.2 Constructor & Destructor Documentation . . . 734
13.38.2.1 Fl_File_Input() . . . 734
13.38.3 Member Function Documentation . . . 734
13.38.3.1 down_box() . . . 734
13.38.3.2 draw() . . . 734
13.38.3.3 handle() . . . 734
13.38.3.4 value() [1/2] . . . 734
13.38.3.5 value() [2/2] . . . 735
13.39 Fl_Fill_Dial Class Reference . . . 735

13.39.1 Detailed Description . . . 743
13.40 Fl_Fill_Slider Class Reference . . . 743
13.40.1 Detailed Description . . . 752
13.41 Fl_Flex Class Reference . . . 752
13.41.1 Detailed Description . . . 762
13.41.2 Member Enumeration Documentation . . . 763
13.41.2.1 anonymous enum . . . 763
13.41.3 Constructor & Destructor Documentation . . . 763
13.41.3.1 Fl_Flex() [1/4] . . . 763
13.41.3.2 Fl_Flex() [2/4] . . . 763
13.41.3.3 Fl_Flex() [3/4] . . . 764
13.41.3.4 Fl_Flex() [4/4] . . . 764
13.41.4 Member Function Documentation . . . 765
13.41.4.1 alloc_size() . . . 765
13.41.4.2 draw() . . . 765
13.41.4.3 end() . . . 765
13.41.4.4 fixed() [1/3] . . . 765
13.41.4.5 fixed() [2/3] . . . 766
13.41.4.6 fixed() [3/3] . . . 766
13.41.4.7 gap() [1/2] . . . 766
13.41.4.8 gap() [2/2] . . . 767
13.41.4.9 horizontal() . . . 767
13.41.4.10 layout() . . . 767
13.41.4.11 margin() [1/4] . . . 767
13.41.4.12 margin() [2/4] . . . 768
13.41.4.13 margin() [3/4] . . . 768
13.41.4.14 margin() [4/4] . . . 768
13.41.4.15 need_layout() [1/2] . . . 769
13.41.4.16 need_layout() [2/2] . . . 769
13.41.4.17 on_remove() . . . 769
13.41.4.18 resize() . . . 769
13.41.4.19 spacing() [1/2] . . . 770
13.41.4.20 spacing() [2/2] . . . 770
13.42 Fl_Float_Input Class Reference . . . 770
13.42.1 Detailed Description . . . 780
13.42.2 Constructor & Destructor Documentation . . . 780
13.42.2.1 Fl_Float_Input() . . . 780
13.43 Fl_FormsBitmap Class Reference . . . 780
13.43.1 Detailed Description . . . 787
13.43.2 Member Function Documentation . . . 787
13.43.2.1 draw() . . . 787
13.43.2.2 set() . . . 787

13.44 Fl_FormsPixmap Class Reference . . . 788
13.44.1 Detailed Description . . . 794
13.44.2 Constructor & Destructor Documentation . . . 794
13.44.2.1 Fl_FormsPixmap() . . . 794
13.44.3 Member Function Documentation . . . 795
13.44.3.1 draw() . . . 795
13.44.3.2 Pixmap() . . . 795
13.44.3.3 set() . . . 795
13.45 Fl_FormsText Class Reference . . . 795
13.45.1 Member Function Documentation . . . 802
13.45.1.1 draw() . . . 802
13.46 Fl_Free Class Reference . . . 802
13.46.1 Detailed Description . . . 809
13.46.2 Constructor & Destructor Documentation . . . 809
13.46.2.1 Fl_Free() . . . 809
13.46.3 Member Function Documentation . . . 810
13.46.3.1 draw() . . . 810
13.46.3.2 handle() . . . 810
13.47 Fl_GIF_Image Class Reference . . . 811
13.47.1 Detailed Description . . . 814
13.47.2 Constructor & Destructor Documentation . . . 814
13.47.2.1 Fl_GIF_Image() [1/3] . . . 814
13.47.2.2 Fl_GIF_Image() [2/3] . . . 814
13.47.2.3 Fl_GIF_Image() [3/3] . . . 815
13.47.3 Member Data Documentation . . . 815
13.47.3.1 animate . . . 815
13.48 Fl_Gl_Choice Class Reference . . . 816
13.49 Fl_Gl_Window Class Reference . . . 816
13.49.1 Detailed Description . . . 830
13.49.2 Constructor & Destructor Documentation . . . 830
13.49.2.1 Fl_Gl_Window() [1/2] . . . 830
13.49.2.2 Fl_Gl_Window() [2/2] . . . 830
13.49.3 Member Function Documentation . . . 830
13.49.3.1 as_gl_window() [1/2] . . . 830
13.49.3.2 as_gl_window() [2/2] . . . 830
13.49.3.3 can_do() . . . 831
13.49.3.4 can_do_overlay() . . . 831
13.49.3.5 context() [1/2] . . . 831
13.49.3.6 context() [2/2] . . . 831
13.49.3.7 context_valid() . . . 831
13.49.3.8 draw() . . . 832
13.49.3.9 draw_begin() . . . 832

13.49.3.10 draw_end() . . . 832
13.49.3.11 flush() . . . 833
13.49.3.12 handle() . . . 833
13.49.3.13 hide() . . . 833
13.49.3.14 make_current() . . . 833
13.49.3.15 make_overlay_current() . . . 833
13.49.3.16 mode() [1/3] . . . 833
13.49.3.17 mode() [2/3] . . . 833
13.49.3.18 mode() [3/3] . . . 834
13.49.3.19 ortho() . . . 834
13.49.3.20 pixel_h() . . . 835
13.49.3.21 pixel_w() . . . 835
13.49.3.22 pixels_per_unit() . . . 835
13.49.3.23 redraw_overlay() . . . 835
13.49.3.24 resize() . . . 835
13.49.3.25 show() . . . 836
13.49.3.26 swap_buffers() . . . 836
13.49.3.27 swap_interval() [1/2] . . . 836
13.49.3.28 swap_interval() [2/2] . . . 837
13.49.3.29 valid() . . . 837
13.50 Fl_Glut_Bitmap_Font Struct Reference . . . 837
13.50.1 Detailed Description . . . 837
13.51 Fl_Glut_StrokeChar Struct Reference . . . 838
13.52 Fl_Glut_StrokeFont Struct Reference . . . 838
13.53 Fl_Glut_StrokeStrip Struct Reference . . . 838
13.54 Fl_Glut_StrokeVertex Struct Reference . . . 838
13.55 Fl_Glut_Window Class Reference . . . 838
13.55.1 Detailed Description . . . 852
13.55.2 Member Function Documentation . . . 853
13.55.2.1 draw() . . . 853
13.55.2.2 draw_overlay() . . . 853
13.55.2.3 handle() . . . 853
13.56 Fl_Grid Class Reference . . . 854
13.56.1 Detailed Description . . . 864
13.56.2 Constructor & Destructor Documentation . . . 865
13.56.2.1 Fl_Grid() . . . 865
13.56.3 Member Function Documentation . . . 865
13.56.3.1 cell() [1/2] . . . 865
13.56.3.2 cell() [2/2] . . . 866
13.56.3.3 clear_layout() . . . 866
13.56.3.4 col_gap() [1/2] . . . 866
13.56.3.5 col_gap() [2/2] . . . 867

13.56.3.6 col_weight() [1/2] . . . 867
13.56.3.7 col_weight() [2/2] . . . 867
13.56.3.8 col_width() [1/2] . . . 868
13.56.3.9 col_width() [2/2] . . . 868
13.56.3.10 debug() . . . 868
13.56.3.11 draw() . . . 869
13.56.3.12 draw_grid() . . . 869
13.56.3.13 gap() [1/2] . . . 869
13.56.3.14 gap() [2/2] . . . 869
13.56.3.15 layout() [1/2] . . . 870
13.56.3.16 layout() [2/2] . . . 870
13.56.3.17 margin() [1/2] . . . 870
13.56.3.18 margin() [2/2] . . . 871
13.56.3.19 need_layout() . . . 871
13.56.3.20 on_remove() . . . 872
13.56.3.21 resize() . . . 872
13.56.3.22 row_gap() [1/2] . . . 872
13.56.3.23 row_gap() [2/2] . . . 872
13.56.3.24 row_height() [1/2] . . . 872
13.56.3.25 row_height() [2/2] . . . 873
13.56.3.26 row_weight() [1/2] . . . 873
13.56.3.27 row_weight() [2/2] . . . 873
13.56.3.28 show_grid() [1/2] . . . 873
13.56.3.29 show_grid() [2/2] . . . 874
13.56.3.30 widget() [1/2] . . . 874
13.56.3.31 widget() [2/2] . . . 875
13.57 Fl_Group Class Reference . . . 875
13.57.1 Detailed Description . . . 884
13.57.2 Constructor & Destructor Documentation . . . 884
13.57.2.1 Fl_Group() . . . 884
13.57.2.2 ~Fl_Group() . . . 885
13.57.3 Member Function Documentation . . . 885
13.57.3.1 array() . . . 885
13.57.3.2 as_group() [1/2] . . . 885
13.57.3.3 as_group() [2/2] . . . 885
13.57.3.4 begin() . . . 885
13.57.3.5 bounds() . . . 886
13.57.3.6 child() . . . 886
13.57.3.7 clear() . . . 886
13.57.3.8 clip_children() [1/2] . . . 887
13.57.3.9 clip_children() [2/2] . . . 887
13.57.3.10 current() [1/2] . . . 887

13.57.3.11 current() [2/2] . . . 887
13.57.3.12 delete_child() . . . 887
13.57.3.13 draw() . . . 888
13.57.3.14 draw_child() . . . 888
13.57.3.15 draw_children() . . . 888
13.57.3.16 end() . . . 888
13.57.3.17 find() . . . 889
13.57.3.18 handle() . . . 889
13.57.3.19 init_sizes() . . . 889
13.57.3.20 insert() [1/2] . . . 890
13.57.3.21 insert() [2/2] . . . 890
13.57.3.22 on_insert() . . . 890
13.57.3.23 on_move() . . . 890
13.57.3.24 on_remove() . . . 891
13.57.3.25 remove() [1/3] . . . 891
13.57.3.26 remove() [2/3] . . . 891
13.57.3.27 remove() [3/3] . . . 891
13.57.3.28 resizable() [1/3] . . . 891
13.57.3.29 resizable() [2/3] . . . 892
13.57.3.30 resizable() [3/3] . . . 892
13.57.3.31 resize() . . . 893
13.57.3.32 sizes() . . . 893
13.57.3.33 update_child() . . . 894
13.58 Fl_Help_Dialog Class Reference . . . 894
13.58.1 Detailed Description . . . 895
13.58.2 Member Function Documentation . . . 896
13.58.2.1 load() . . . 896
13.58.2.2 show() . . . 896
13.58.2.3 textsize() . . . 896
13.58.2.4 value() [1/2] . . . 896
13.58.2.5 value() [2/2] . . . 896
13.59 Fl_Help_View Class Reference . . . 896
13.59.1 Detailed Description . . . 906
13.59.2 Constructor & Destructor Documentation . . . 908
13.59.2.1 Fl_Help_View() . . . 908
13.59.2.2 ~Fl_Help_View() . . . 908
13.59.3 Member Function Documentation . . . 908
13.59.3.1 copy() . . . 908
13.59.3.2 directory() . . . 908
13.59.3.3 draw() . . . 909
13.59.3.4 filename() . . . 909
13.59.3.5 find() . . . 909

13.59.3.6 handle() . . . 909
13.59.3.7 leftline() . . . 910
13.59.3.8 link() . . . 910
13.59.3.9 load() . . . 910
13.59.3.10 resize() . . . 911
13.59.3.11 scrollbar_size() [1/2] . . . 911
13.59.3.12 scrollbar_size() [2/2] . . . 911
13.59.3.13 text_selected() . . . 912
13.59.3.14 title() . . . 912
13.59.3.15 topline() [1/2] . . . 912
13.59.3.16 topline() [2/2] . . . 912
13.59.3.17 value() . . . 912
13.60 Fl_Hold_Browser Class Reference . . . 913
13.60.1 Detailed Description . . . 926
13.60.2 Constructor & Destructor Documentation . . . 926
13.60.2.1 Fl_Hold_Browser() . . . 926
13.61 Fl_Hor_Fill_Slider Class Reference . . . 927
13.62 Fl_Hor_Nice_Slider Class Reference . . . 936
13.62.1 Detailed Description . . . 945
13.63 Fl_Hor_Slider Class Reference . . . 945
13.63.1 Detailed Description . . . 954
13.64 Fl_Hor_Value_Slider Class Reference . . . 954
13.65 Fl_ICO_Image Class Reference . . . 963
13.65.1 Detailed Description . . . 967
13.65.2 Constructor & Destructor Documentation . . . 967
13.65.2.1 Fl_ICO_Image() . . . 967
13.66 Fl_Image Class Reference . . . 967
13.66.1 Detailed Description . . . 969
13.66.2 Constructor & Destructor Documentation . . . 970
13.66.2.1 Fl_Image() . . . 970
13.66.3 Member Function Documentation . . . 970
13.66.3.1 as_shared_image() . . . 970
13.66.3.2 color_average() . . . 970
13.66.3.3 copy() [1/2] . . . 970
13.66.3.4 copy() [2/2] . . . 971
13.66.3.5 count() . . . 971
13.66.3.6 d() . . . 972
13.66.3.7 data() [1/2] . . . 972
13.66.3.8 data() [2/2] . . . 972
13.66.3.9 desaturate() . . . 972
13.66.3.10 draw() [1/2] . . . 972
13.66.3.11 draw() [2/2] . . . 973

13.66.3.12 draw_empty() . . . 973
13.66.3.13 draw_scaled() . . . 973
13.66.3.14 fail() . . . 974
13.66.3.15 h() [1/2] . . . 974
13.66.3.16 h() [2/2] . . . 974
13.66.3.17 inactive() . . . 974
13.66.3.18 label() [1/2] . . . 974
13.66.3.19 label() [2/2] . . . 975
13.66.3.20 ld() [1/2] . . . 975
13.66.3.21 ld() [2/2] . . . 975
13.66.3.22 release() . . . 975
13.66.3.23 RGB_scaling() . . . 975
13.66.3.24 scale() . . . 976
13.66.3.25 scaling_algorithm() . . . 976
13.66.3.26 uncache() . . . 976
13.66.3.27 w() [1/2] . . . 977
13.66.3.28 w() [2/2] . . . 977
13.67 Fl_Image_Reader Class Reference . . . 977
13.68 Fl_Image_Surface Class Reference . . . 977
13.68.1 Detailed Description . . . 979
13.68.2 Constructor & Destructor Documentation . . . 980
13.68.2.1 Fl_Image_Surface() . . . 980
13.68.3 Member Function Documentation . . . 980
13.68.3.1 highres_image() . . . 980
13.68.3.2 image() . . . 980
13.68.3.3 is_current() . . . 980
13.68.3.4 mask() . . . 981
13.68.3.5 offscreen() . . . 981
13.68.3.6 origin() [1/2] . . . 982
13.68.3.7 origin() [2/2] . . . 982
13.68.3.8 printable_rect() . . . 982
13.68.3.9 rescale() . . . 982
13.68.3.10 set_current() . . . 982
13.68.3.11 translate() . . . 983
13.68.3.12 untranslate() . . . 983
13.69 Fl_Input Class Reference . . . 983
13.69.1 Detailed Description . . . 993
13.69.2 Constructor & Destructor Documentation . . . 994
13.69.2.1 Fl_Input() . . . 994
13.69.3 Member Function Documentation . . . 995
13.69.3.1 draw() . . . 995
13.69.3.2 handle() . . . 995

13.69.3.3 handle_key() . . . 995
13.69.3.4 handle_rmb() . . . 996
13.70 Fl_Input_ Class Reference . . . 996
13.70.1 Detailed Description . . . 1005
13.70.2 Constructor & Destructor Documentation . . . 1005
13.70.2.1 Fl_Input_() . . . 1005
13.70.2.2 ~Fl_Input_() . . . 1006
13.70.3 Member Function Documentation . . . 1006
13.70.3.1 append() . . . 1006
13.70.3.2 apply_undo() . . . 1006
13.70.3.3 can_redo() . . . 1007
13.70.3.4 can_undo() . . . 1007
13.70.3.5 copy() . . . 1007
13.70.3.6 copy_cuts() . . . 1007
13.70.3.7 cursor_color() [1/2] . . . 1007
13.70.3.8 cursor_color() [2/2] . . . 1008
13.70.3.9 cut() [1/3] . . . 1008
13.70.3.10 cut() [2/3] . . . 1008
13.70.3.11 cut() [3/3] . . . 1008
13.70.3.12 drawtext() [1/2] . . . 1009
13.70.3.13 drawtext() [2/2] . . . 1009
13.70.3.14 dvalue() . . . 1009
13.70.3.15 handle_mouse() . . . 1009
13.70.3.16 handletext() . . . 1010
13.70.3.17 index() . . . 1010
13.70.3.18 input_type() [1/2] . . . 1010
13.70.3.19 input_type() [2/2] . . . 1010
13.70.3.20 insert() . . . 1011
13.70.3.21 insert_position() [1/3] . . . 1011
13.70.3.22 insert_position() [2/3] . . . 1011
13.70.3.23 insert_position() [3/3] . . . 1011
13.70.3.24 ivalue() . . . 1012
13.70.3.25 line_end() . . . 1012
13.70.3.26 line_start() . . . 1012
13.70.3.27 mark() [1/2] . . . 1013
13.70.3.28 mark() [2/2] . . . 1013
13.70.3.29 maximum_size() [1/2] . . . 1013
13.70.3.30 maximum_size() [2/2] . . . 1013
13.70.3.31 placeholder() [1/2] . . . 1013
13.70.3.32 placeholder() [2/2] . . . 1014
13.70.3.33 position() [1/3] . . . 1014
13.70.3.34 position() [2/3] . . . 1014

13.70.3.35 position() [3/3] . . . 1014
13.70.3.36 readonly() [1/2] . . . 1014
13.70.3.37 readonly() [2/2] . . . 1014
13.70.3.38 redo() . . . 1015
13.70.3.39 replace() . . . 1015
13.70.3.40 resize() . . . 1015
13.70.3.41 shortcut() [1/2] . . . 1016
13.70.3.42 shortcut() [2/2] . . . 1016
13.70.3.43 size() [1/2] . . . 1016
13.70.3.44 size() [2/2] . . . 1016
13.70.3.45 static_value() [1/2] . . . 1017
13.70.3.46 static_value() [2/2] . . . 1017
13.70.3.47 tab_nav() [1/2] . . . 1017
13.70.3.48 tab_nav() [2/2] . . . 1018
13.70.3.49 textcolor() [1/2] . . . 1018
13.70.3.50 textcolor() [2/2] . . . 1018
13.70.3.51 textfont() [1/2] . . . 1018
13.70.3.52 textfont() [2/2] . . . 1019
13.70.3.53 textsize() [1/2] . . . 1019
13.70.3.54 textsize() [2/2] . . . 1019
13.70.3.55 undo() . . . 1019
13.70.3.56 up_down_position() . . . 1019
13.70.3.57 value() [1/5] . . . 1020
13.70.3.58 value() [2/5] . . . 1020
13.70.3.59 value() [3/5] . . . 1020
13.70.3.60 value() [4/5] . . . 1021
13.70.3.61 value() [5/5] . . . 1021
13.70.3.62 word_end() . . . 1021
13.70.3.63 word_start() . . . 1022
13.70.3.64 wrap() [1/2] . . . 1022
13.70.3.65 wrap() [2/2] . . . 1022
13.71 Fl_Input_Choice Class Reference . . . 1022
13.71.1 Detailed Description . . . 1032
13.71.2 Constructor & Destructor Documentation . . . 1033
13.71.2.1 Fl_Input_Choice() . . . 1033
13.71.3 Member Function Documentation . . . 1033
13.71.3.1 add() . . . 1033
13.71.3.2 draw() . . . 1034
13.71.3.3 inp_x() . . . 1034
13.71.3.4 input() . . . 1034
13.71.3.5 menu_x() . . . 1034
13.71.3.6 menubutton() . . . 1034

13.71.3.7 resize() . . . 1034
13.71.3.8 update_menubutton() . . . 1035
13.71.3.9 value() [1/2] . . . 1035
13.71.3.10 value() [2/2] . . . 1035
13.72 Fl_Int_Input Class Reference . . . 1035
13.72.1 Detailed Description . . . 1045
13.72.2 Constructor & Destructor Documentation . . . 1045
13.72.2.1 Fl_Int_Input() . . . 1045
13.73 Fl_JPEG_Image Class Reference . . . 1046
13.73.1 Detailed Description . . . 1049
13.73.2 Constructor & Destructor Documentation . . . 1049
13.73.2.1 Fl_JPEG_Image() [1/2] . . . 1049
13.73.2.2 Fl_JPEG_Image() [2/2] . . . 1049
13.74 Fl_Label Struct Reference . . . 1050
13.74.1 Detailed Description . . . 1050
13.74.2 Member Function Documentation . . . 1051
13.74.2.1 draw() . . . 1051
13.74.2.2 measure() . . . 1051
13.74.3 Member Data Documentation . . . 1051
13.74.3.1 type . . . 1051
13.75 Fl_Light_Button Class Reference . . . 1051
13.75.1 Detailed Description . . . 1059
13.75.2 Constructor & Destructor Documentation . . . 1059
13.75.2.1 Fl_Light_Button() . . . 1059
13.75.3 Member Function Documentation . . . 1059
13.75.3.1 draw() . . . 1059
13.75.3.2 handle() . . . 1059
13.76 Fl_Line_Dial Class Reference . . . 1060
13.77 Fl_Mac_App_Menu Class Reference . . . 1068
13.77.1 Member Function Documentation . . . 1069
13.77.1.1 custom_application_menu_items() . . . 1069
13.77.2 Member Data Documentation . . . 1069
13.77.2.1 print . . . 1069
13.78 Fl_Menu_ Class Reference . . . 1069
13.78.1 Detailed Description . . . 1078
13.78.2 Constructor & Destructor Documentation . . . 1079
13.78.2.1 Fl_Menu_() . . . 1079
13.78.3 Member Function Documentation . . . 1079
13.78.3.1 add() [1/2] . . . 1079
13.78.3.2 add() [2/2] . . . 1079
13.78.3.3 clear() . . . 1082
13.78.3.4 clear_submenu() . . . 1082

13.78.3.5 copy() . . . 1082
13.78.3.6 down_box() . . . 1082
13.78.3.7 find_index() [1/3] . . . 1082
13.78.3.8 find_index() [2/3] . . . 1083
13.78.3.9 find_index() [3/3] . . . 1083
13.78.3.10 find_item() [1/2] . . . 1084
13.78.3.11 find_item() [2/2] . . . 1084
13.78.3.12 find_item_with_argument() . . . 1085
13.78.3.13 find_item_with_user_data() . . . 1085
13.78.3.14 global() . . . 1085
13.78.3.15 insert() . . . 1085
13.78.3.16 item_pathname() . . . 1086
13.78.3.17 menu() [1/2] . . . 1087
13.78.3.18 menu() [2/2] . . . 1087
13.78.3.19 menu_box() [1/2] . . . 1087
13.78.3.20 menu_box() [2/2] . . . 1087
13.78.3.21 menu_end() . . . 1088
13.78.3.22 mode() [1/2] . . . 1088
13.78.3.23 mode() [2/2] . . . 1088
13.78.3.24 mvalue() . . . 1088
13.78.3.25 picked() . . . 1088
13.78.3.26 prev_mvalue() . . . 1089
13.78.3.27 remove() . . . 1089
13.78.3.28 replace() . . . 1089
13.78.3.29 size() . . . 1089
13.78.3.30 test_shortcut() . . . 1089
13.78.3.31 text() [1/2] . . . 1090
13.78.3.32 text() [2/2] . . . 1090
13.78.3.33 textcolor() . . . 1090
13.78.3.34 textfont() [1/2] . . . 1090
13.78.3.35 textfont() [2/2] . . . 1090
13.78.3.36 textsize() [1/2] . . . 1090
13.78.3.37 textsize() [2/2] . . . 1090
13.78.3.38 value() [1/3] . . . 1091
13.78.3.39 value() [2/3] . . . 1091
13.78.3.40 value() [3/3] . . . 1092
13.79 Fl_Menu_Bar Class Reference . . . 1092
13.79.1 Detailed Description . . . 1102
13.79.2 Constructor & Destructor Documentation . . . 1102
13.79.2.1 Fl_Menu_Bar() . . . 1102
13.79.3 Member Function Documentation . . . 1102
13.79.3.1 draw() . . . 1102

13.79.3.2 handle() . . . 1103
13.79.3.3 play_menu() . . . 1103
13.79.3.4 update() . . . 1103
13.80 Fl_Menu_Button Class Reference . . . 1104
13.80.1 Detailed Description . . . 1113
13.80.2 Member Enumeration Documentation . . . 1113
13.80.2.1 popup_buttons . . . 1113
13.80.3 Constructor & Destructor Documentation . . . 1114
13.80.3.1 Fl_Menu_Button() . . . 1114
13.80.4 Member Function Documentation . . . 1114
13.80.4.1 draw() . . . 1114
13.80.4.2 handle() . . . 1114
13.80.4.3 popup() . . . 1115
13.81 Fl_Menu_Item Struct Reference . . . 1115
13.81.1 Detailed Description . . . 1118
13.81.2 Member Function Documentation . . . 1120
13.81.2.1 add() . . . 1120
13.81.2.2 argument() [1/2] . . . 1120
13.81.2.3 argument() [2/2] . . . 1120
13.81.2.4 callback() [1/5] . . . 1120
13.81.2.5 callback() [2/5] . . . 1121
13.81.2.6 callback() [3/5] . . . 1121
13.81.2.7 callback() [4/5] . . . 1121
13.81.2.8 callback() [5/5] . . . 1121
13.81.2.9 checkbox() . . . 1121
13.81.2.10 deactivate() . . . 1121
13.81.2.11 do_callback() [1/3] . . . 1122
13.81.2.12 do_callback() [2/3] . . . 1122
13.81.2.13 do_callback() [3/3] . . . 1122
13.81.2.14 draw() . . . 1122
13.81.2.15 find_shortcut() . . . 1122
13.81.2.16 headline() . . . 1123
13.81.2.17 image() [1/2] . . . 1123
13.81.2.18 image() [2/2] . . . 1123
13.81.2.19 image_label() . . . 1123
13.81.2.20 insert() . . . 1123
13.81.2.21 label() [1/3] . . . 1124
13.81.2.22 label() [2/3] . . . 1124
13.81.2.23 label() [3/3] . . . 1125
13.81.2.24 labelcolor() [1/2] . . . 1125
13.81.2.25 labelcolor() [2/2] . . . 1125
13.81.2.26 labelfont() [1/2] . . . 1125

13.81.2.27 labelfont() [2/2] . . . 1125
13.81.2.28 labeltype() [1/2] . . . 1126
13.81.2.29 labeltype() [2/2] . . . 1126
13.81.2.30 measure() . . . 1126
13.81.2.31 multi_label() . . . 1126
13.81.2.32 next() [1/2] . . . 1126
13.81.2.33 next() [2/2] . . . 1127
13.81.2.34 popup() . . . 1127
13.81.2.35 pulldown() . . . 1127
13.81.2.36 radio() . . . 1128
13.81.2.37 selectable() . . . 1128
13.81.2.38 set() . . . 1128
13.81.2.39 setonly() . . . 1128
13.81.2.40 shortcut() . . . 1128
13.81.2.41 size() . . . 1129
13.81.2.42 submenu() . . . 1129
13.81.2.43 test_shortcut() . . . 1129
13.81.2.44 value() . . . 1129
13.82 Fl_Menu_Window Class Reference . . . 1129
13.82.1 Detailed Description . . . 1142
13.83 Fl_Multi_Browser Class Reference . . . 1142
13.83.1 Detailed Description . . . 1156
13.83.2 Constructor & Destructor Documentation . . . 1156
13.83.2.1 Fl_Multi_Browser() . . . 1156
13.84 Fl_Multi_Label Struct Reference . . . 1156
13.84.1 Detailed Description . . . 1157
13.84.2 Member Function Documentation . . . 1158
13.84.2.1 label() [1/2] . . . 1158
13.84.2.2 label() [2/2] . . . 1158
13.84.3 Member Data Documentation . . . 1158
13.84.3.1 labela . . . 1158
13.84.3.2 labelb . . . 1158
13.84.3.3 typea . . . 1159
13.84.3.4 typeb . . . 1159
13.85 Fl_Multiline_Input Class Reference . . . 1159
13.85.1 Detailed Description . . . 1169
13.85.2 Constructor & Destructor Documentation . . . 1169
13.85.2.1 Fl_Multiline_Input() . . . 1169
13.86 Fl_Multiline_Output Class Reference . . . 1169
13.86.1 Detailed Description . . . 1179
13.86.2 Constructor & Destructor Documentation . . . 1179
13.86.2.1 Fl_Multiline_Output() . . . 1179

13.87 Fl_Native_File_Chooser Class Reference . . . 1179
13.87.1 Detailed Description . . . 1181
13.87.2 Member Enumeration Documentation . . . 1182
13.87.2.1 Option . . . 1182
13.87.2.2 Type . . . 1182
13.87.3 Constructor & Destructor Documentation . . . 1183
13.87.3.1 Fl_Native_File_Chooser() . . . 1183
13.87.3.2 ~Fl_Native_File_Chooser() . . . 1183
13.87.4 Member Function Documentation . . . 1183
13.87.4.1 count() . . . 1183
13.87.4.2 directory() . . . 1183
13.87.4.3 errmsg() . . . 1183
13.87.4.4 filename() [1/2] . . . 1183
13.87.4.5 filename() [2/2] . . . 1183
13.87.4.6 filter() [1/2] . . . 1184
13.87.4.7 filter() [2/2] . . . 1184
13.87.4.8 filter_value() [1/2] . . . 1184
13.87.4.9 filter_value() [2/2] . . . 1184
13.87.4.10 options() . . . 1185
13.87.4.11 preset_file() . . . 1185
13.87.4.12 show() . . . 1185
13.87.4.13 title() [1/2] . . . 1185
13.87.4.14 title() [2/2] . . . 1185
13.88 Fl_Nice_Slider Class Reference . . . 1186
13.89 Fl_Output Class Reference . . . 1195
13.89.1 Detailed Description . . . 1204
13.89.2 Constructor & Destructor Documentation . . . 1205
13.89.2.1 Fl_Output() . . . 1205
13.90 Fl_Overlay_Window Class Reference . . . 1205
13.90.1 Detailed Description . . . 1218
13.90.2 Constructor & Destructor Documentation . . . 1218
13.90.2.1 Fl_Overlay_Window() . . . 1218
13.90.3 Member Function Documentation . . . 1219
13.90.3.1 as_overlay_window() . . . 1219
13.90.3.2 draw_overlay() . . . 1219
13.90.3.3 flush() . . . 1219
13.90.3.4 hide() . . . 1219
13.90.3.5 redraw_overlay() . . . 1219
13.90.3.6 resize() . . . 1219
13.90.3.7 show() . . . 1220
13.91 Fl_Pack Class Reference . . . 1220
13.91.1 Detailed Description . . . 1229

13.91.2 Constructor & Destructor Documentation . . . 1229
13.91.2.1 Fl_Pack() . . . 1229
13.91.3 Member Function Documentation . . . 1229
13.91.3.1 clear() . . . 1229
13.91.3.2 draw() . . . 1230
13.91.3.3 draw_filler_() . . . 1230
13.91.3.4 horizontal() . . . 1230
13.91.3.5 resize() . . . 1230
13.92 Fl_Paged_Device Class Reference . . . 1231
13.92.1 Detailed Description . . . 1233
13.92.2 Member Enumeration Documentation . . . 1233
13.92.2.1 Page_Format . . . 1233
13.92.2.2 Page_Layout . . . 1234
13.92.3 Member Function Documentation . . . 1234
13.92.3.1 begin_job() . . . 1234
13.92.3.2 begin_page() . . . 1235
13.92.3.3 end_job() . . . 1235
13.92.3.4 end_page() . . . 1235
13.92.3.5 margins() . . . 1236
13.92.3.6 rotate() . . . 1236
13.92.3.7 scale() . . . 1236
13.92.3.8 start_job() . . . 1236
13.92.3.9 start_page() . . . 1237
13.93 Fl_PDF_File_Surface Class Reference . . . 1237
13.93.1 Detailed Description . . . 1240
13.93.2 Member Function Documentation . . . 1240
13.93.2.1 begin_document() . . . 1240
13.93.2.2 begin_job() [1/2] . . . 1240
13.93.2.3 begin_job() [2/2] . . . 1240
13.93.2.4 begin_page() . . . 1241
13.93.2.5 end_job() . . . 1241
13.93.2.6 end_page() . . . 1241
13.93.2.7 is_current() . . . 1241
13.93.2.8 margins() . . . 1242
13.93.2.9 origin() [1/2] . . . 1242
13.93.2.10 origin() [2/2] . . . 1242
13.93.2.11 printable_rect() . . . 1242
13.93.2.12 rotate() . . . 1243
13.93.2.13 scale() . . . 1243
13.93.2.14 set_current() . . . 1243
13.93.2.15 translate() . . . 1244
13.93.2.16 untranslate() . . . 1244

13.94 Fl_Pixmap Class Reference . . . 1244
13.94.1 Detailed Description . . . 1246
13.94.2 Constructor & Destructor Documentation . . . 1247
13.94.2.1 Fl_Pixmap() [1/4] . . . 1247
13.94.2.2 Fl_Pixmap() [2/4] . . . 1247
13.94.2.3 Fl_Pixmap() [3/4] . . . 1247
13.94.2.4 Fl_Pixmap() [4/4] . . . 1247
13.94.3 Member Function Documentation . . . 1247
13.94.3.1 color_average() . . . 1247
13.94.3.2 copy() . . . 1247
13.94.3.3 desaturate() . . . 1248
13.94.3.4 draw() . . . 1248
13.94.3.5 label() [1/2] . . . 1249
13.94.3.6 label() [2/2] . . . 1249
13.94.3.7 uncache() . . . 1249
13.95 Fl_Plugin Class Reference . . . 1249
13.95.1 Detailed Description . . . 1249
13.95.2 Constructor & Destructor Documentation . . . 1250
13.95.2.1 Fl_Plugin() . . . 1250
13.96 Fl_Plugin_Manager Class Reference . . . 1250
13.96.1 Detailed Description . . . 1254
13.96.2 Constructor & Destructor Documentation . . . 1254
13.96.2.1 ~Fl_Plugin_Manager() . . . 1254
13.96.3 Member Function Documentation . . . 1254
13.96.3.1 addPlugin() . . . 1254
13.96.3.2 klass_list() . . . 1254
13.96.3.3 load() . . . 1254
13.96.3.4 loadAll() . . . 1255
13.96.3.5 removePlugin() . . . 1255
13.97 Fl_PNG_Image Class Reference . . . 1255
13.97.1 Detailed Description . . . 1258
13.97.2 Constructor & Destructor Documentation . . . 1258
13.97.2.1 Fl_PNG_Image() [1/2] . . . 1258
13.97.2.2 Fl_PNG_Image() [2/2] . . . 1258
13.98 Fl_PNM_Image Class Reference . . . 1259
13.98.1 Detailed Description . . . 1262
13.98.2 Constructor & Destructor Documentation . . . 1262
13.98.2.1 Fl_PNM_Image() . . . 1262
13.99 Fl_Positioner Class Reference . . . 1262
13.99.1 Detailed Description . . . 1269
13.99.2 Constructor & Destructor Documentation . . . 1270
13.99.2.1 Fl_Positioner() . . . 1270

13.99.3 Member Function Documentation . . . 1270
13.99.3.1 draw() . . . 1270
13.99.3.2 handle() . . . 1270
13.100 Fl_PostScript_File_Device Class Reference . . . 1271
13.100.1 Detailed Description . . . 1274
13.100.2 Member Function Documentation . . . 1275
13.100.2.1 begin_job() [1/3] . . . 1275
13.100.2.2 begin_job() [2/3] . . . 1275
13.100.2.3 begin_job() [3/3] . . . 1275
13.100.2.4 begin_page() . . . 1276
13.100.2.5 end_current() . . . 1276
13.100.2.6 end_job() . . . 1276
13.100.2.7 end_page() . . . 1276
13.100.2.8 margins() . . . 1277
13.100.2.9 origin() [1/2] . . . 1277
13.100.2.10 origin() [2/2] . . . 1277
13.100.2.11 printable_rect() . . . 1277
13.100.2.12 rotate() . . . 1278
13.100.2.13 scale() . . . 1278
13.100.2.14 set_current() . . . 1278
13.100.2.15 start_job() [1/2] . . . 1279
13.100.2.16 start_job() [2/2] . . . 1279
13.100.2.17 translate() . . . 1279
13.100.2.18 untranslate() . . . 1279
13.101 Fl_Preferences Class Reference . . . 1279
13.101.1 Detailed Description . . . 1283
13.101.2 Member Typedef Documentation . . . 1284
13.101.2.1 ID . . . 1284
13.101.3 Member Enumeration Documentation . . . 1284
13.101.3.1 Root . . . 1284
13.101.4 Constructor & Destructor Documentation . . . 1284
13.101.4.1 Fl_Preferences() [1/8] . . . 1284
13.101.4.2 Fl_Preferences() [2/8] . . . 1285
13.101.4.3 Fl_Preferences() [3/8] . . . 1286
13.101.4.4 Fl_Preferences() [4/8] . . . 1286
13.101.4.5 Fl_Preferences() [5/8] . . . 1287
13.101.4.6 Fl_Preferences() [6/8] . . . 1287
13.101.4.7 Fl_Preferences() [7/8] . . . 1287
13.101.4.8 ~Fl_Preferences() . . . 1288
13.101.4.9 Fl_Preferences() [8/8] . . . 1288
13.101.5 Member Function Documentation . . . 1288
13.101.5.1 delete_entry() . . . 1288

13.101.5.2 delete_group() . . . 1288
13.101.5.3 dirty() . . . 1289
13.101.5.4 entries() . . . 1289
13.101.5.5 entry() . . . 1289
13.101.5.6 entry_exists() . . . 1289
13.101.5.7 file_access() [1/2] . . . 1289
13.101.5.8 file_access() [2/2] . . . 1290
13.101.5.9 filename() [1/2] . . . 1290
13.101.5.10 filename() [2/2] . . . 1290
13.101.5.11 flush() . . . 1291
13.101.5.12 get() [1/9] . . . 1291
13.101.5.13 get() [2/9] . . . 1291
13.101.5.14 get() [3/9] . . . 1292
13.101.5.15 get() [4/9] . . . 1292
13.101.5.16 get() [5/9] . . . 1293
13.101.5.17 get() [6/9] . . . 1293
13.101.5.18 get() [7/9] . . . 1293
13.101.5.19 get() [8/9] . . . 1294
13.101.5.20 get() [9/9] . . . 1294
13.101.5.21 get_userdata_path() . . . 1295
13.101.5.22 group() . . . 1295
13.101.5.23 group_exists() . . . 1296
13.101.5.24 groups() . . . 1296
13.101.5.25 new_UUID() . . . 1296
13.101.5.26 set() [1/8] . . . 1296
13.101.5.27 set() [2/8] . . . 1297
13.101.5.28 set() [3/8] . . . 1297
13.101.5.29 set() [4/8] . . . 1297
13.101.5.30 set() [5/8] . . . 1298
13.101.5.31 set() [6/8] . . . 1298
13.101.5.32 set() [7/8] . . . 1298
13.101.5.33 set() [8/8] . . . 1299
13.101.5.34 size() . . . 1299
13.101.6 Member Data Documentation . . . 1299
13.101.6.1 CORE_READ_OK . . . 1299
13.101.6.2 CORE_WRITE_OK . . . 1299
13.101.6.3 NONE . . . 1299
13.102 Fl_Printer Class Reference . . . 1300
13.102.1 Detailed Description . . . 1303
13.102.2 Member Function Documentation . . . 1304
13.102.2.1 begin_job() . . . 1304
13.102.2.2 begin_page() . . . 1304

13.102.2.3 end_job() . . . 1305
13.102.2.4 end_page() . . . 1305
13.102.2.5 is_current() . . . 1305
13.102.2.6 margins() . . . 1305
13.102.2.7 origin() [1/2] . . . 1306
13.102.2.8 origin() [2/2] . . . 1306
13.102.2.9 printable_rect() . . . 1306
13.102.2.10 rotate() . . . 1306
13.102.2.11 scale() . . . 1307
13.102.2.12 set_current() . . . 1307
13.102.2.13 translate() . . . 1307
13.102.2.14 untranslate() . . . 1307
13.103 Fl_Progress Class Reference . . . 1308
13.103.1 Detailed Description . . . 1314
13.103.2 Constructor & Destructor Documentation . . . 1315
13.103.2.1 Fl_Progress() . . . 1315
13.103.3 Member Function Documentation . . . 1315
13.103.3.1 draw() . . . 1315
13.103.3.2 maximum() [1/2] . . . 1315
13.103.3.3 maximum() [2/2] . . . 1315
13.103.3.4 minimum() [1/2] . . . 1315
13.103.3.5 minimum() [2/2] . . . 1315
13.103.3.6 value() [1/2] . . . 1315
13.103.3.7 value() [2/2] . . . 1316
13.104 Fl_Radio_Button Class Reference . . . 1316
13.104.1 Constructor & Destructor Documentation . . . 1323
13.104.1.1 Fl_Radio_Button() . . . 1323
13.105 Fl_Radio_Light_Button Class Reference . . . 1324
13.106 Fl_Radio_Round_Button Class Reference . . . 1331
13.106.1 Constructor & Destructor Documentation . . . 1339
13.106.1.1 Fl_Radio_Round_Button() . . . 1339
13.107 Fl_Rect Class Reference . . . 1339
13.107.1 Detailed Description . . . 1340
13.107.2 Constructor & Destructor Documentation . . . 1341
13.107.2.1 Fl_Rect() . . . 1341
13.107.3 Member Function Documentation . . . 1341
13.107.3.1 b() . . . 1341
13.107.3.2 inset() [1/3] . . . 1341
13.107.3.3 inset() [2/3] . . . 1341
13.107.3.4 inset() [3/3] . . . 1341
13.107.3.5 r() . . . 1342
13.108 Fl_Repeat_Button Class Reference . . . 1342

13.108.1 Detailed Description . . . 1349
13.108.2 Constructor & Destructor Documentation . . . 1349
13.108.2.1 Fl_Repeat_Button() . . . 1349
13.108.3 Member Function Documentation . . . 1350
13.108.3.1 handle() . . . 1350
13.109 Fl_Return_Button Class Reference . . . 1350
13.109.1 Detailed Description . . . 1358
13.109.2 Constructor & Destructor Documentation . . . 1358
13.109.2.1 Fl_Return_Button() . . . 1358
13.109.3 Member Function Documentation . . . 1358
13.109.3.1 draw() . . . 1358
13.109.3.2 handle() . . . 1358
13.110 Fl_RGB_Image Class Reference . . . 1359
13.110.1 Detailed Description . . . 1362
13.110.2 Constructor & Destructor Documentation . . . 1362
13.110.2.1 Fl_RGB_Image() [1/3] . . . 1362
13.110.2.2 Fl_RGB_Image() [2/3] . . . 1363
13.110.2.3 Fl_RGB_Image() [3/3] . . . 1363
13.110.3 Member Function Documentation . . . 1364
13.110.3.1 as_svg_image() [1/2] . . . 1364
13.110.3.2 as_svg_image() [2/2] . . . 1364
13.110.3.3 color_average() . . . 1364
13.110.3.4 copy() . . . 1364
13.110.3.5 desaturate() . . . 1365
13.110.3.6 draw() . . . 1365
13.110.3.7 label() [1/2] . . . 1365
13.110.3.8 label() [2/2] . . . 1365
13.110.3.9 max_size() [1/2] . . . 1366
13.110.3.10 max_size() [2/2] . . . 1366
13.110.3.11 normalize() . . . 1366
13.110.3.12 uncache() . . . 1366
13.110.4 Member Data Documentation . . . 1366
13.110.4.1 array . . . 1366
13.111 Fl_Roller Class Reference . . . 1366
13.111.1 Detailed Description . . . 1374
13.111.2 Constructor & Destructor Documentation . . . 1375
13.111.2.1 Fl_Roller() . . . 1375
13.111.3 Member Function Documentation . . . 1375
13.111.3.1 draw() . . . 1375
13.111.3.2 handle() . . . 1375
13.112 Fl_Round_Button Class Reference . . . 1376
13.112.1 Detailed Description . . . 1383

13.112.2 Constructor & Destructor Documentation . . . 1384
13.112.2.1 Fl_Round_Button() . . . 1384
13.113 Fl_Round_Clock Class Reference . . . 1384
13.113.1 Detailed Description . . . 1391
13.113.2 Constructor & Destructor Documentation . . . 1392
13.113.2.1 Fl_Round_Clock() . . . 1392
13.114 Fl_Scheme Class Reference . . . 1392
13.114.1 Detailed Description . . . 1392
13.114.2 Member Function Documentation . . . 1393
13.114.2.1 add_scheme_name() . . . 1393
13.114.2.2 names() . . . 1393
13.114.2.3 num_schemes() . . . 1394
13.114.2.4 plastic_color_average() . . . 1394
13.115 Fl_Scheme_Choice Class Reference . . . 1395
13.115.1 Constructor & Destructor Documentation . . . 1404
13.115.1.1 Fl_Scheme_Choice() . . . 1404
13.115.2 Member Function Documentation . . . 1404
13.115.2.1 handle() . . . 1404
13.115.2.2 init_value() . . . 1405
13.115.2.3 scheme_cb_() . . . 1405
13.116 Fl_Scroll Class Reference . . . 1405
13.116.1 Detailed Description . . . 1414
13.116.2 Constructor & Destructor Documentation . . . 1415
13.116.2.1 Fl_Scroll() . . . 1415
13.116.2.2 ~Fl_Scroll() . . . 1415
13.116.3 Member Function Documentation . . . 1416
13.116.3.1 bbox() . . . 1416
13.116.3.2 delete_child() . . . 1416
13.116.3.3 draw() . . . 1416
13.116.3.4 fix_scrollbar_order() . . . 1417
13.116.3.5 handle() . . . 1417
13.116.3.6 on_insert() . . . 1417
13.116.3.7 on_move() . . . 1418
13.116.3.8 recalc_scrollbars() . . . 1418
13.116.3.9 resize() . . . 1419
13.116.3.10 scroll_to() . . . 1419
13.116.3.11 scrollbar_size() [1/2] . . . 1419
13.116.3.12 scrollbar_size() [2/2] . . . 1420
13.116.3.13 xposition() . . . 1420
13.116.3.14 yposition() . . . 1420
13.117 Fl_Scrollbar Class Reference . . . 1420
13.117.1 Detailed Description . . . 1430

13.117.2 Constructor & Destructor Documentation . . . 1430
13.117.2.1 Fl_Scrollbar() . . . 1430
13.117.3 Member Function Documentation . . . 1430
13.117.3.1 draw() . . . 1430
13.117.3.2 handle() . . . 1430
13.117.3.3 linesize() . . . 1431
13.117.3.4 value() [1/3] . . . 1431
13.117.3.5 value() [2/3] . . . 1431
13.117.3.6 value() [3/3] . . . 1432
13.118 Fl_Secret_Input Class Reference . . . 1432
13.118.1 Detailed Description . . . 1442
13.118.2 Constructor & Destructor Documentation . . . 1442
13.118.2.1 Fl_Secret_Input() . . . 1442
13.118.3 Member Function Documentation . . . 1442
13.118.3.1 handle() . . . 1442
13.119 Fl_Select_Browser Class Reference . . . 1443
13.119.1 Detailed Description . . . 1456
13.119.2 Constructor & Destructor Documentation . . . 1456
13.119.2.1 Fl_Select_Browser() . . . 1456
13.120 Fl_Shared_Image Class Reference . . . 1457
13.120.1 Detailed Description . . . 1460
13.120.2 Constructor & Destructor Documentation . . . 1460
13.120.2.1 Fl_Shared_Image() [1/2] . . . 1460
13.120.2.2 Fl_Shared_Image() [2/2] . . . 1461
13.120.2.3 ~Fl_Shared_Image() . . . 1461
13.120.3 Member Function Documentation . . . 1461
13.120.3.1 add() . . . 1461
13.120.3.2 add_handler() . . . 1461
13.120.3.3 as_shared_image() . . . 1461
13.120.3.4 color_average() . . . 1462
13.120.3.5 compare() . . . 1462
13.120.3.6 copy() [1/2] . . . 1463
13.120.3.7 copy() [2/2] . . . 1463
13.120.3.8 copy_() . . . 1463
13.120.3.9 desaturate() . . . 1463
13.120.3.10 draw() . . . 1464
13.120.3.11 find() . . . 1464
13.120.3.12 get() [1/2] . . . 1464
13.120.3.13 get() [2/2] . . . 1465
13.120.3.14 image() . . . 1465
13.120.3.15 images() . . . 1466
13.120.3.16 num_images() . . . 1466

13.120.3.17 original() . . . 1466
13.120.3.18 refcount() . . . 1467
13.120.3.19 release() . . . 1467
13.120.3.20 uncache() . . . 1467
13.120.3.21 update() . . . 1467
13.121 Fl_Shortcut_Button Class Reference . . . 1467
13.121.1 Detailed Description . . . 1475
13.121.2 Constructor & Destructor Documentation . . . 1475
13.121.2.1 Fl_Shortcut_Button() . . . 1475
13.121.3 Member Function Documentation . . . 1475
13.121.3.1 draw() . . . 1475
13.121.3.2 handle() . . . 1475
13.121.3.3 value() [1/2] . . . 1475
13.121.3.4 value() [2/2] . . . 1476
13.122 Fl_Simple_Counter Class Reference . . . 1476
13.122.1 Detailed Description . . . 1484
13.123 Fl_Single_Window Class Reference . . . 1485
13.123.1 Detailed Description . . . 1497
13.123.2 Member Function Documentation . . . 1497
13.123.2.1 flush() . . . 1497
13.123.2.2 show() . . . 1497
13.124 Fl_Slider Class Reference . . . 1498
13.124.1 Detailed Description . . . 1506
13.124.2 Member Enumeration Documentation . . . 1507
13.124.2.1 Scale_Type . . . 1507
13.124.2.2 Tick_Position . . . 1507
13.124.3 Constructor & Destructor Documentation . . . 1507
13.124.3.1 Fl_Slider() . . . 1507
13.124.4 Member Function Documentation . . . 1508
13.124.4.1 bounds() . . . 1508
13.124.4.2 draw() . . . 1508
13.124.4.3 draw_ticks() . . . 1508
13.124.4.4 handle() . . . 1508
13.124.4.5 increment_lin_log() . . . 1509
13.124.4.6 num_ticks() . . . 1509
13.124.4.7 position_to_value() . . . 1509
13.124.4.8 scale() [1/2] . . . 1509
13.124.4.9 scale() [2/2] . . . 1509
13.124.4.10 scrollvalue() . . . 1510
13.124.4.11 slider_size() . . . 1510
13.124.4.12 ticks() [1/2] . . . 1510
13.124.4.13 ticks() [2/2] . . . 1510

13.124.4.14 value_to_position() . . . 1511
13.125 Fl_Spinner Class Reference . . . 1511
13.125.1 Detailed Description . . . 1521
13.125.2 Constructor & Destructor Documentation . . . 1521
13.125.2.1 Fl_Spinner() . . . 1521
13.125.3 Member Function Documentation . . . 1521
13.125.3.1 draw() . . . 1521
13.125.3.2 handle() . . . 1521
13.125.3.3 resize() . . . 1522
13.125.3.4 step() [1/2] . . . 1522
13.125.3.5 step() [2/2] . . . 1522
13.125.3.6 type() [1/2] . . . 1522
13.125.3.7 type() [2/2] . . . 1523
13.125.3.8 value() . . . 1523
13.125.3.9 wrap() [1/2] . . . 1523
13.125.3.10 wrap() [2/2] . . . 1523
13.126 Fl_Spinner::Fl_Spinner_Input Class Reference . . . 1524
13.126.1 Member Function Documentation . . . 1533
13.126.1.1 handle() . . . 1533
13.127 Fl_Surface_Device Class Reference . . . 1534
13.127.1 Detailed Description . . . 1534
13.127.2 Member Function Documentation . . . 1535
13.127.2.1 end_current() . . . 1535
13.127.2.2 is_current() . . . 1535
13.127.2.3 pop_current() . . . 1535
13.127.2.4 push_current() . . . 1535
13.127.2.5 set_current() . . . 1536
13.127.2.6 surface() . . . 1536
13.128 Fl_SVG_File_Surface Class Reference . . . 1536
13.128.1 Detailed Description . . . 1538
13.128.2 Constructor & Destructor Documentation . . . 1538
13.128.2.1 Fl_SVG_File_Surface() . . . 1538
13.128.2.2 ~Fl_SVG_File_Surface() . . . 1539
13.128.3 Member Function Documentation . . . 1539
13.128.3.1 close() . . . 1539
13.128.3.2 origin() [1/2] . . . 1539
13.128.3.3 origin() [2/2] . . . 1539
13.128.3.4 printable_rect() . . . 1539
13.128.3.5 translate() . . . 1540
13.128.3.6 untranslate() . . . 1540
13.129 Fl_SVG_Image Class Reference . . . 1540
13.129.1 Detailed Description . . . 1543

13.129.2 Constructor & Destructor Documentation . . . 1544
13.129.2.1 Fl_SVG_Image() [1/3] . . . 1544
13.129.2.2 Fl_SVG_Image() [2/3] . . . 1545
13.129.2.3 Fl_SVG_Image() [3/3] . . . 1545
13.129.3 Member Function Documentation . . . 1545
13.129.3.1 as_svg_image() [1/2] . . . 1545
13.129.3.2 as_svg_image() [2/2] . . . 1546
13.129.3.3 color_average() . . . 1546
13.129.3.4 copy() . . . 1546
13.129.3.5 desaturate() . . . 1546
13.129.3.6 draw() . . . 1547
13.129.3.7 normalize() . . . 1547
13.129.3.8 resize() . . . 1547
13.129.3.9 scale() . . . 1547
13.129.4 Member Data Documentation . . . 1548
13.129.4.1 proportional . . . 1548
13.130 Fl_Sys_Menu_Bar Class Reference . . . 1548
13.130.1 Detailed Description . . . 1558
13.130.2 Member Enumeration Documentation . . . 1559
13.130.2.1 window_menu_style_enum . . . 1559
13.130.3 Constructor & Destructor Documentation . . . 1560
13.130.3.1 Fl_Sys_Menu_Bar() . . . 1560
13.130.4 Member Function Documentation . . . 1560
13.130.4.1 about() . . . 1560
13.130.4.2 add() [1/3] . . . 1560
13.130.4.3 add() [2/3] . . . 1560
13.130.4.4 add() [3/3] . . . 1561
13.130.4.5 clear() . . . 1561
13.130.4.6 clear_submenu() . . . 1561
13.130.4.7 create_window_menu() . . . 1561
13.130.4.8 draw() . . . 1562
13.130.4.9 insert() [1/2] . . . 1562
13.130.4.10 insert() [2/2] . . . 1562
13.130.4.11 is_global() . . . 1562
13.130.4.12 menu() . . . 1563
13.130.4.13 mode() . . . 1563
13.130.4.14 play_menu() . . . 1563
13.130.4.15 remove() . . . 1563
13.130.4.16 replace() . . . 1563
13.130.4.17 update() . . . 1564
13.130.4.18 window_menu_style() . . . 1564
13.131 Fl_Table Class Reference . . . 1564

13.131.1 Detailed Description . . . 1577
13.131.2 Member Enumeration Documentation . . . 1579
13.131.2.1 TableContext . . . 1579
13.131.3 Constructor & Destructor Documentation . . . 1579
13.131.3.1 Fl_Table() . . . 1579
13.131.3.2 ~Fl_Table() . . . 1580
13.131.4 Member Function Documentation . . . 1580
13.131.4.1 array() . . . 1580
13.131.4.2 callback() . . . 1580
13.131.4.3 callback_col() . . . 1581
13.131.4.4 callback_context() . . . 1581
13.131.4.5 callback_row() . . . 1581
13.131.4.6 child() . . . 1581
13.131.4.7 children() . . . 1581
13.131.4.8 clear() . . . 1581
13.131.4.9 col_header() . . . 1581
13.131.4.10 col_resize() . . . 1582
13.131.4.11 col_resize_min() . . . 1582
13.131.4.12 col_width() . . . 1582
13.131.4.13 col_width_all() . . . 1582
13.131.4.14 cursor2rowcol() . . . 1582
13.131.4.15 damage_zone() . . . 1582
13.131.4.16 do_callback() . . . 1582
13.131.4.17 draw() . . . 1583
13.131.4.18 draw_cell() . . . 1583
13.131.4.19 find_cell() . . . 1584
13.131.4.20 get_selection() . . . 1584
13.131.4.21 handle() . . . 1585
13.131.4.22 init_sizes() . . . 1585
13.131.4.23 insert() . . . 1585
13.131.4.24 is_interactive_resize() . . . 1585
13.131.4.25 is_selected() . . . 1585
13.131.4.26 move_cursor() . . . 1585
13.131.4.27 recalc_dimensions() . . . 1586
13.131.4.28 redraw_range() . . . 1586
13.131.4.29 resize() . . . 1586
13.131.4.30 row_col_clamp() . . . 1586
13.131.4.31 row_header() . . . 1586
13.131.4.32 row_height() . . . 1587
13.131.4.33 row_height_all() . . . 1587
13.131.4.34 row_resize() . . . 1587
13.131.4.35 row_resize_min() . . . 1587

13.131.4.36 rows() . . . 1587
13.131.4.37 scrollbar_size() [1/2] . . . 1587
13.131.4.38 scrollbar_size() [2/2] . . . 1587
13.131.4.39 set_selection() . . . 1588
13.131.4.40 tab_cell_nav() [1/2] . . . 1588
13.131.4.41 tab_cell_nav() [2/2] . . . 1588
13.131.4.42 table_box() . . . 1589
13.131.4.43 table_resized() . . . 1589
13.131.4.44 table_scrolled() . . . 1589
13.131.4.45 top_row() [1/2] . . . 1589
13.131.4.46 top_row() [2/2] . . . 1589
13.131.4.47 visible_cells() . . . 1589
13.131.4.48 when() . . . 1590
13.132 Fl_Table_Row Class Reference . . . 1590
13.132.1 Detailed Description . . . 1604
13.132.2 Constructor & Destructor Documentation . . . 1604
13.132.2.1 Fl_Table_Row() . . . 1604
13.132.2.2 ~Fl_Table_Row() . . . 1604
13.132.3 Member Function Documentation . . . 1604
13.132.3.1 clear() . . . 1604
13.132.3.2 handle() . . . 1604
13.132.3.3 row_selected() . . . 1605
13.132.3.4 rows() . . . 1605
13.132.3.5 select_all_rows() . . . 1605
13.132.3.6 select_row() . . . 1605
13.132.3.7 type() . . . 1606
13.133 Fl_Tabs Class Reference . . . 1606
13.133.1 Detailed Description . . . 1616
13.133.2 Member Enumeration Documentation . . . 1619
13.133.2.1 anonymous enum . . . 1619
13.133.3 Constructor & Destructor Documentation . . . 1619
13.133.3.1 Fl_Tabs() . . . 1619
13.133.4 Member Function Documentation . . . 1620
13.133.4.1 clear_tab_positions() . . . 1620
13.133.4.2 client_area() . . . 1620
13.133.4.3 draw() . . . 1621
13.133.4.4 draw_tab() . . . 1621
13.133.4.5 handle() . . . 1621
13.133.4.6 handle_overflow() . . . 1621
13.133.4.7 handle_overflow_menu() . . . 1622
13.133.4.8 hit_close() . . . 1622
13.133.4.9 hit_overflow_menu() . . . 1622

13.133.4.10 hit_tabs_area() . . . 1624
13.133.4.11 maybe_do_callback() . . . 1624
13.133.4.12 on_insert() . . . 1624
13.133.4.13 on_move() . . . 1624
13.133.4.14 on_remove() . . . 1625
13.133.4.15 push() [1/2] . . . 1625
13.133.4.16 push() [2/2] . . . 1625
13.133.4.17 redraw_tabs() . . . 1625
13.133.4.18 resize() . . . 1625
13.133.4.19 show() . . . 1625
13.133.4.20 tab_align() [1/2] . . . 1625
13.133.4.21 tab_align() [2/2] . . . 1626
13.133.4.22 tab_height() . . . 1626
13.133.4.23 tab_positions() . . . 1626
13.133.4.24 take_focus() . . . 1627
13.133.4.25 value() [1/3] . . . 1627
13.133.4.26 value() [2/3] . . . 1627
13.133.4.27 value() [3/3] . . . 1628
13.133.4.28 which() . . . 1628
13.133.5 Member Data Documentation . . . 1628
13.133.5.1 overflow_type . . . 1628
13.133.5.2 tab_count . . . 1628
13.133.5.3 tab_flags . . . 1628
13.133.5.4 tab_pos . . . 1629
13.133.5.5 tab_width . . . 1629
13.134 Fl_Terminal Class Reference . . . 1629
13.134.1 Detailed Description . . . 1644
13.134.2 The Fl_Terminal Widget . . . 1645
13.134.2.1 Examples . . . 1646
13.134.2.2 Writing To Terminal From Applications . . . 1646
13.134.2.3 Text Attributes . . . 1647
13.134.2.4 Text and Background Colors . . . 1647
13.134.2.5 Features . . . 1648
13.134.2.6 Margins . . . 1649
13.134.2.7 Caveats . . . 1650
13.134.3 Member Enumeration Documentation . . . 1650
13.134.3.1 Attrib . . . 1650
13.134.3.2 CharFlags . . . 1650
13.134.3.3 OutFlags . . . 1651
13.134.3.4 RedrawStyle . . . 1651
13.134.3.5 ScrollbarStyle . . . 1651
13.134.4 Constructor & Destructor Documentation . . . 1651

13.134.4.1 Fl_Terminal() [1/2] . . . 1651
13.134.4.2 Fl_Terminal() [2/2] . . . 1652
13.134.4.3 ~Fl_Terminal() . . . 1652
13.134.5 Member Function Documentation . . . 1652
13.134.5.1 ansi() [1/2] . . . 1652
13.134.5.2 ansi() [2/2] . . . 1652
13.134.5.3 append() . . . 1653
13.134.5.4 append_ascii() . . . 1653
13.134.5.5 append_utf8() . . . 1654
13.134.5.6 box() . . . 1654
13.134.5.7 clear() [1/2] . . . 1654
13.134.5.8 clear() [2/2] . . . 1654
13.134.5.9 clear_screen() . . . 1654
13.134.5.10 clear_screen_home() . . . 1655
13.134.5.11 color() . . . 1655
13.134.5.12 cursor_col() . . . 1655
13.134.5.13 cursor_cr() . . . 1655
13.134.5.14 cursor_down() . . . 1655
13.134.5.15 cursor_right() . . . 1655
13.134.5.16 cursor_row() . . . 1656
13.134.5.17 cursor_up() . . . 1656
13.134.5.18 delete_rows() . . . 1656
13.134.5.19 display_columns() [1/2] . . . 1656
13.134.5.20 display_columns() [2/2] . . . 1656
13.134.5.21 display_rows() [1/2] . . . 1656
13.134.5.22 display_rows() [2/2] . . . 1656
13.134.5.23 draw() . . . 1657
13.134.5.24 draw_buff() . . . 1657
13.134.5.25 draw_row() . . . 1657
13.134.5.26 draw_row_bg() . . . 1657
13.134.5.27 error_char() [1/2] . . . 1657
13.134.5.28 error_char() [2/2] . . . 1658
13.134.5.29 get_selection() . . . 1658
13.134.5.30 h_to_row() . . . 1659
13.134.5.31 handle() . . . 1659
13.134.5.32 handle_unknown_char() [1/2] . . . 1659
13.134.5.33 handle_unknown_char() [2/2] . . . 1659
13.134.5.34 history_lines() . . . 1659
13.134.5.35 history_use() . . . 1659
13.134.5.36 hscrollbar_style() [1/2] . . . 1660
13.134.5.37 hscrollbar_style() [2/2] . . . 1660
13.134.5.38 insert_char() . . . 1660

13.134.5.39 insert_rows() . . . 1660
13.134.5.40 is_inside_selection() . . . 1660
13.134.5.41 output_translate() . . . 1661
13.134.5.42 plot_char() [1/2] . . . 1661
13.134.5.43 plot_char() [2/2] . . . 1661
13.134.5.44 print_char() [1/2] . . . 1662
13.134.5.45 print_char() [2/2] . . . 1662
13.134.5.46 printf() . . . 1662
13.134.5.47 redraw_rate() . . . 1663
13.134.5.48 redraw_style() [1/2] . . . 1663
13.134.5.49 redraw_style() [2/2] . . . 1663
13.134.5.50 reset_terminal() . . . 1663
13.134.5.51 resize() . . . 1664
13.134.5.52 scroll() . . . 1664
13.134.5.53 scrollbar_actual_size() . . . 1664
13.134.5.54 scrollbar_size() [1/2] . . . 1664
13.134.5.55 scrollbar_size() [2/2] . . . 1664
13.134.5.56 selection_extend() . . . 1665
13.134.5.57 selection_text() . . . 1665
13.134.5.58 selection_text_len() . . . 1665
13.134.5.59 show_unknown() [1/2] . . . 1665
13.134.5.60 show_unknown() [2/2] . . . 1665
13.134.5.61 text() . . . 1665
13.134.5.62 textattrib() [1/2] . . . 1666
13.134.5.63 textattrib() [2/2] . . . 1666
13.134.5.64 textbgcolor() . . . 1666
13.134.5.65 textbgcolor_default() [1/2] . . . 1667
13.134.5.66 textbgcolor_default() [2/2] . . . 1667
13.134.5.67 textbgcolor_xterm() . . . 1667
13.134.5.68 textcolor() . . . 1668
13.134.5.69 textfgcolor() . . . 1668
13.134.5.70 textfgcolor_default() [1/2] . . . 1668
13.134.5.71 textfgcolor_default() [2/2] . . . 1668
13.134.5.72 textfgcolor_xterm() . . . 1669
13.134.5.73 textfont() . . . 1669
13.134.5.74 textsize() . . . 1669
13.134.5.75 u8c_disp_row() . . . 1669
13.134.5.76 u8c_hist_row() . . . 1670
13.134.5.77 u8c_hist_use_row() . . . 1670
13.134.5.78 u8c_ring_row() . . . 1670
13.134.5.79 utf8_char_at_disp() . . . 1671
13.134.5.80 utf8_char_at_glob() . . . 1671

13.134.5.81 vprintf() . . . 1671
13.134.5.82 w_to_col() . . . 1672
13.134.5.83 walk_selection() . . . 1672
13.134.6 Member Data Documentation . . . 1672
13.134.6.1 hscrollbar . . . 1672
13.134.6.2 scrollbar . . . 1673
13.135 Fl_Text_Buffer Class Reference . . . 1673
13.135.1 Detailed Description . . . 1678
13.135.2 Constructor & Destructor Documentation . . . 1678
13.135.2.1 Fl_Text_Buffer() . . . 1678
13.135.3 Member Function Documentation . . . 1678
13.135.3.1 add_modify_callback() . . . 1678
13.135.3.2 address() [1/2] . . . 1679
13.135.3.3 address() [2/2] . . . 1679
13.135.3.4 append() . . . 1679
13.135.3.5 appendfile() . . . 1679
13.135.3.6 byte_at() . . . 1679
13.135.3.7 can_redo() . . . 1680
13.135.3.8 can_undo() . . . 1680
13.135.3.9 canUndo() . . . 1680
13.135.3.10 char_at() . . . 1680
13.135.3.11 copy() . . . 1681
13.135.3.12 count_displayed_characters() . . . 1681
13.135.3.13 count_lines() . . . 1681
13.135.3.14 estimate_lines() . . . 1681
13.135.3.15 findchar_backward() . . . 1681
13.135.3.16 findchar_forward() . . . 1682
13.135.3.17 highlight_text() . . . 1682
13.135.3.18 insert() . . . 1682
13.135.3.19 insert_() . . . 1683
13.135.3.20 insertfile() . . . 1683
13.135.3.21 is_word_separator() . . . 1683
13.135.3.22 length() . . . 1683
13.135.3.23 line_end() . . . 1683
13.135.3.24 line_start() . . . 1684
13.135.3.25 line_text() . . . 1684
13.135.3.26 loadfile() . . . 1684
13.135.3.27 next_char() . . . 1684
13.135.3.28 outputfile() . . . 1685
13.135.3.29 prev_char() . . . 1685
13.135.3.30 printf() . . . 1685
13.135.3.31 remove() . . . 1686

13.135.3.32 remove_() . . . 1686
13.135.3.33 replace() . . . 1686
13.135.3.34 rewind_lines() . . . 1686
13.135.3.35 savefile() . . . 1686
13.135.3.36 search_backward() . . . 1687
13.135.3.37 search_forward() . . . 1687
13.135.3.38 secondary_selection_text() . . . 1687
13.135.3.39 selection_text() . . . 1688
13.135.3.40 skip_displayed_characters() . . . 1688
13.135.3.41 tab_distance() . . . 1688
13.135.3.42 text() [1/2] . . . 1688
13.135.3.43 text() [2/2] . . . 1688
13.135.3.44 text_range() . . . 1688
13.135.3.45 text_str() . . . 1689
13.135.3.46 undo() . . . 1689
13.135.3.47 vprintf() . . . 1689
13.135.3.48 word_end() . . . 1689
13.135.3.49 word_start() . . . 1690
13.135.4 Member Data Documentation . . . 1690
13.135.4.1 file_encoding_warning_message . . . 1690
13.135.4.2 mTabDist . . . 1690
13.135.4.3 transcoding_warning_action . . . 1690
13.136 Fl_Text_Display Class Reference . . . 1690
13.136.1 Detailed Description . . . 1706
13.136.2 Member Enumeration Documentation . . . 1707
13.136.2.1 anonymous enum . . . 1707
13.136.2.2 anonymous enum . . . 1708
13.136.2.3 anonymous enum . . . 1708
13.136.3 Constructor & Destructor Documentation . . . 1708
13.136.3.1 Fl_Text_Display() . . . 1708
13.136.3.2 ~Fl_Text_Display() . . . 1709
13.136.4 Member Function Documentation . . . 1709
13.136.4.1 absolute_top_line_number() . . . 1709
13.136.4.2 buffer() [1/3] . . . 1709
13.136.4.3 buffer() [2/3] . . . 1709
13.136.4.4 buffer() [3/3] . . . 1709
13.136.4.5 buffer_modified_cb() . . . 1710
13.136.4.6 buffer_predelete_cb() . . . 1710
13.136.4.7 calc_last_char() . . . 1710
13.136.4.8 calc_line_starts() . . . 1711
13.136.4.9 clear_rect() . . . 1711
13.136.4.10 col_to_x() . . . 1711

13.136.4.11 count_lines() . . . 1711
13.136.4.12 cursor_color() [1/2] . . . 1712
13.136.4.13 cursor_color() [2/2] . . . 1712
13.136.4.14 cursor_style() . . . 1712
13.136.4.15 display_insert() . . . 1713
13.136.4.16 display_needs_recalc() . . . 1713
13.136.4.17 draw() . . . 1713
13.136.4.18 draw_cursor() . . . 1713
13.136.4.19 draw_line_numbers() . . . 1713
13.136.4.20 draw_range() . . . 1713
13.136.4.21 draw_string() . . . 1714
13.136.4.22 draw_text() . . . 1714
13.136.4.23 draw_vline() . . . 1714
13.136.4.24 empty_vlines() . . . 1715
13.136.4.25 extend_range_for_styles() . . . 1715
13.136.4.26 find_line_end() . . . 1715
13.136.4.27 find_wrap_range() . . . 1716
13.136.4.28 find_x() . . . 1716
13.136.4.29 get_absolute_top_line_number() . . . 1716
13.136.4.30 grammar_underline_color() [1/2] . . . 1717
13.136.4.31 grammar_underline_color() [2/2] . . . 1717
13.136.4.32 handle() . . . 1717
13.136.4.33 handle_rmb() . . . 1717
13.136.4.34 handle_vline() . . . 1717
13.136.4.35 highlight_data() . . . 1718
13.136.4.36 in_selection() . . . 1719
13.136.4.37 insert() . . . 1719
13.136.4.38 insert_position() [1/2] . . . 1719
13.136.4.39 insert_position() [2/2] . . . 1720
13.136.4.40 line_end() . . . 1720
13.136.4.41 line_start() . . . 1720
13.136.4.42 linenumber_align() . . . 1720
13.136.4.43 linenumber_bgcolor() . . . 1721
13.136.4.44 linenumber_fgcolor() . . . 1721
13.136.4.45 linenumber_font() . . . 1721
13.136.4.46 linenumber_format() . . . 1721
13.136.4.47 linenumber_size() . . . 1721
13.136.4.48 linenumber_width() . . . 1722
13.136.4.49 longest_vline() . . . 1722
13.136.4.50 maintain_absolute_top_line_number() . . . 1722
13.136.4.51 maintaining_absolute_top_line_number() . . . 1722
13.136.4.52 measure_deleted_lines() . . . 1722

13.136.4.53 measure_proportional_character() . . . 1723
13.136.4.54 measure_vline() . . . 1723
13.136.4.55 move_down() . . . 1723
13.136.4.56 move_left() . . . 1723
13.136.4.57 move_right() . . . 1724
13.136.4.58 move_up() . . . 1724
13.136.4.59 offset_line_starts() . . . 1724
13.136.4.60 overstrike() . . . 1724
13.136.4.61 position_style() . . . 1724
13.136.4.62 position_to_line() . . . 1725
13.136.4.63 position_to_linecol() . . . 1725
13.136.4.64 position_to_xy() . . . 1726
13.136.4.65 recalc_display() . . . 1726
13.136.4.66 redisplay_range() . . . 1726
13.136.4.67 reset_absolute_top_line_number() . . . 1726
13.136.4.68 resize() . . . 1727
13.136.4.69 rewind_lines() . . . 1727
13.136.4.70 scroll() . . . 1727
13.136.4.71 scroll_() . . . 1727
13.136.4.72 scroll_timer_cb() . . . 1729
13.136.4.73 scrollbar_align() [1/2] . . . 1729
13.136.4.74 scrollbar_align() [2/2] . . . 1729
13.136.4.75 scrollbar_size() [1/2] . . . 1729
13.136.4.76 scrollbar_size() [2/2] . . . 1729
13.136.4.77 scrollbar_width() [1/2] . . . 1730
13.136.4.78 scrollbar_width() [2/2] . . . 1730
13.136.4.79 secondary_selection_color() [1/2] . . . 1730
13.136.4.80 secondary_selection_color() [2/2] . . . 1730
13.136.4.81 shortcut() [1/2] . . . 1731
13.136.4.82 shortcut() [2/2] . . . 1731
13.136.4.83 show_cursor() . . . 1731
13.136.4.84 show_insert_position() . . . 1731
13.136.4.85 skip_lines() . . . 1731
13.136.4.86 spelling_underline_color() [1/2] . . . 1732
13.136.4.87 spelling_underline_color() [2/2] . . . 1732
13.136.4.88 string_width() . . . 1732
13.136.4.89 style_buffer() . . . 1732
13.136.4.90 textcolor() [1/2] . . . 1733
13.136.4.91 textcolor() [2/2] . . . 1733
13.136.4.92 textfont() [1/2] . . . 1733
13.136.4.93 textfont() [2/2] . . . 1733
13.136.4.94 textsize() [1/2] . . . 1733

13.136.4.95 textsize() [2/2] . . . 1733
13.136.4.96 update_h_scrollbar() . . . 1734
13.136.4.97 update_line_starts() . . . 1734
13.136.4.98 update_v_scrollbar() . . . 1734
13.136.4.99 vline_length() . . . 1734
13.136.4.100 word_end() . . . 1734
13.136.4.101 word_start() . . . 1735
13.136.4.102 wrap_mode() . . . 1735
13.136.4.103 wrap_uses_character() . . . 1735
13.136.4.104 wrapped_column() . . . 1736
13.136.4.105 wrapped_line_counter() . . . 1736
13.136.4.106 wrapped_row() . . . 1737
13.136.4.107 x_to_col() . . . 1737
13.136.4.108 xy_to_position() . . . 1738
13.136.4.109 xy_to_rowcol() . . . 1738
13.137 Fl_Text_Editor Class Reference . . . 1738
13.137.1 Detailed Description . . . 1756
13.137.2 Member Function Documentation . . . 1756
13.137.2.1 add_key_binding() . . . 1756
13.137.2.2 handle() . . . 1756
13.137.2.3 insert_mode() [1/2] . . . 1756
13.137.2.4 insert_mode() [2/2] . . . 1756
13.137.2.5 kf_backspace() . . . 1756
13.137.2.6 kf_c_s_move() . . . 1756
13.137.2.7 kf_copy() . . . 1757
13.137.2.8 kf_ctrl_move() . . . 1757
13.137.2.9 kf_cut() . . . 1757
13.137.2.10 kf_default() . . . 1757
13.137.2.11 kf_delete() . . . 1757
13.137.2.12 kf_down() . . . 1757
13.137.2.13 kf_end() . . . 1757
13.137.2.14 kf_enter() . . . 1758
13.137.2.15 kf_home() . . . 1758
13.137.2.16 kf_ignore() . . . 1758
13.137.2.17 kf_insert() . . . 1758
13.137.2.18 kf_left() . . . 1758
13.137.2.19 kf_m_s_move() . . . 1758
13.137.2.20 kf_meta_move() . . . 1758
13.137.2.21 kf_move() . . . 1759
13.137.2.22 kf_page_down() . . . 1759
13.137.2.23 kf_page_up() . . . 1759
13.137.2.24 kf_paste() . . . 1759

13.137.2.25 kf_redo() . . . 1759
13.137.2.26 kf_right() . . . 1759
13.137.2.27 kf_select_all() . . . 1760
13.137.2.28 kf_shift_move() . . . 1760
13.137.2.29 kf_undo() . . . 1760
13.137.2.30 kf_up() . . . 1760
13.137.2.31 remove_key_binding() . . . 1760
13.137.2.32 tab_nav() [1/2] . . . 1760
13.137.2.33 tab_nav() [2/2] . . . 1761
13.137.3 Member Data Documentation . . . 1761
13.137.3.1 global_key_bindings . . . 1761
13.138 Fl_Text_Selection Class Reference . . . 1761
13.138.1 Detailed Description . . . 1762
13.138.2 Member Function Documentation . . . 1763
13.138.2.1 end() . . . 1763
13.138.2.2 includes() . . . 1763
13.138.2.3 length() . . . 1763
13.138.2.4 position() . . . 1763
13.138.2.5 selected() [1/3] . . . 1763
13.138.2.6 selected() [2/3] . . . 1763
13.138.2.7 selected() [3/3] . . . 1764
13.138.2.8 set() . . . 1764
13.138.2.9 start() . . . 1764
13.138.2.10 update() . . . 1765
13.139 Fl_Tile Class Reference . . . 1765
13.139.1 Detailed Description . . . 1774
13.139.2 Constructor & Destructor Documentation . . . 1776
13.139.2.1 Fl_Tile() . . . 1776
13.139.3 Member Function Documentation . . . 1776
13.139.3.1 cursor() . . . 1776
13.139.3.2 drag_intersection() . . . 1777
13.139.3.3 handle() . . . 1777
13.139.3.4 init_size_range() . . . 1777
13.139.3.5 move_intersection() . . . 1778
13.139.3.6 on_insert() . . . 1778
13.139.3.7 on_move() . . . 1778
13.139.3.8 on_remove() . . . 1778
13.139.3.9 position() . . . 1778
13.139.3.10 request_grow_b() . . . 1779
13.139.3.11 request_grow_l() . . . 1779
13.139.3.12 request_grow_r() . . . 1779
13.139.3.13 request_grow_t() . . . 1779

13.139.3.14 request_shrink_b() . . . 1780
13.139.3.15 request_shrink_l() . . . 1780
13.139.3.16 request_shrink_r() . . . 1780
13.139.3.17 request_shrink_t() . . . 1781
13.139.3.18 resize() . . . 1781
13.139.3.19 set_cursor() . . . 1781
13.139.3.20 size_range() [1/2] . . . 1782
13.139.3.21 size_range() [2/2] . . . 1782
13.140 Fl_Tiled_Image Class Reference . . . 1782
13.140.1 Detailed Description . . . 1785
13.140.2 Constructor & Destructor Documentation . . . 1785
13.140.2.1 Fl_Tiled_Image() . . . 1785
13.140.3 Member Function Documentation . . . 1785
13.140.3.1 color_average() . . . 1785
13.140.3.2 copy() . . . 1786
13.140.3.3 desaturate() . . . 1786
13.140.3.4 draw() . . . 1786
13.141 Fl_Timeout Class Reference . . . 1787
13.141.1 Detailed Description . . . 1788
13.141.2 Member Function Documentation . . . 1789
13.141.2.1 add_timeout() . . . 1789
13.141.2.2 current() . . . 1790
13.141.2.3 elapse_timeouts() . . . 1790
13.141.2.4 get() . . . 1790
13.141.2.5 has_timeout() . . . 1791
13.141.2.6 insert() . . . 1792
13.141.2.7 make_current() . . . 1792
13.141.2.8 release() . . . 1792
13.141.2.9 remove_next_timeout() . . . 1792
13.141.2.10 remove_timeout() . . . 1793
13.141.2.11 repeat_timeout() . . . 1793
13.141.2.12 time_to_wait() . . . 1793
13.141.3 Member Data Documentation . . . 1794
13.141.3.1 current_timeout . . . 1794
13.141.3.2 first_timeout . . . 1794
13.141.3.3 free_timeout . . . 1794
13.142 Fl_Timer Class Reference . . . 1794
13.142.1 Detailed Description . . . 1801
13.142.2 Constructor & Destructor Documentation . . . 1801
13.142.2.1 Fl_Timer() . . . 1801
13.142.3 Member Function Documentation . . . 1802
13.142.3.1 direction() [1/2] . . . 1802

13.142.3.2 direction() [2/2] . . . 1802
13.142.3.3 draw() . . . 1802
13.142.3.4 handle() . . . 1802
13.142.3.5 suspended() . . . 1803
13.143 Fl_Toggle_Button Class Reference . . . 1803
13.143.1 Detailed Description . . . 1810
13.143.2 Constructor & Destructor Documentation . . . 1810
13.143.2.1 Fl_Toggle_Button() . . . 1810
13.144 Fl_Tooltip Class Reference . . . 1811
13.144.1 Detailed Description . . . 1812
13.144.2 Member Function Documentation . . . 1813
13.144.2.1 color() [1/2] . . . 1813
13.144.2.2 color() [2/2] . . . 1813
13.144.2.3 current() . . . 1813
13.144.2.4 delay() [1/2] . . . 1813
13.144.2.5 delay() [2/2] . . . 1813
13.144.2.6 disable() . . . 1813
13.144.2.7 enable() . . . 1813
13.144.2.8 enabled() . . . 1813
13.144.2.9 enter_area() . . . 1814
13.144.2.10 font() [1/2] . . . 1814
13.144.2.11 font() [2/2] . . . 1814
13.144.2.12 hidedelay() [1/2] . . . 1814
13.144.2.13 hidedelay() [2/2] . . . 1814
13.144.2.14 hoverdelay() [1/2] . . . 1814
13.144.2.15 hoverdelay() [2/2] . . . 1814
13.144.2.16 margin_height() [1/2] . . . 1815
13.144.2.17 margin_height() [2/2] . . . 1815
13.144.2.18 margin_width() [1/2] . . . 1815
13.144.2.19 margin_width() [2/2] . . . 1815
13.144.2.20 override_text() . . . 1815
13.144.2.21 size() [1/2] . . . 1815
13.144.2.22 size() [2/2] . . . 1816
13.144.2.23 textcolor() [1/2] . . . 1816
13.144.2.24 textcolor() [2/2] . . . 1816
13.144.2.25 wrap_width() [1/2] . . . 1816
13.144.2.26 wrap_width() [2/2] . . . 1816
13.145 Fl_Tree Class Reference . . . 1816
13.145.1 Detailed Description . . . 1830
13.145.2 Member Function Documentation . . . 1835
13.145.2.1 add() [1/2] . . . 1835
13.145.2.2 add() [2/2] . . . 1836

13.145.2.3 auto_resize_children() [1/2] . . . 1836
13.145.2.4 auto_resize_children() [2/2] . . . 1836
13.145.2.5 calc_dimensions() . . . 1837
13.145.2.6 calc_tree() . . . 1837
13.145.2.7 callback_item() [1/2] . . . 1837
13.145.2.8 callback_item() [2/2] . . . 1837
13.145.2.9 callback_reason() [1/2] . . . 1838
13.145.2.10 callback_reason() [2/2] . . . 1838
13.145.2.11 clear() . . . 1838
13.145.2.12 clear_children() . . . 1838
13.145.2.13 close() [1/2] . . . 1838
13.145.2.14 close() [2/2] . . . 1839
13.145.2.15 closeicon() [1/2] . . . 1839
13.145.2.16 closeicon() [2/2] . . . 1840
13.145.2.17 connectorstyle() . . . 1840
13.145.2.18 deselect() [1/2] . . . 1840
13.145.2.19 deselect() [2/2] . . . 1840
13.145.2.20 deselect_all() . . . 1841
13.145.2.21 display() . . . 1841
13.145.2.22 displayed() . . . 1841
13.145.2.23 draw() . . . 1842
13.145.2.24 extend_selection() . . . 1842
13.145.2.25 extend_selection_dir() . . . 1842
13.145.2.26 find_clicked() . . . 1843
13.145.2.27 find_item() . . . 1843
13.145.2.28 first() . . . 1844
13.145.2.29 first_selected_item() . . . 1844
13.145.2.30 first_visible() . . . 1845
13.145.2.31 first_visible_item() . . . 1845
13.145.2.32 get_selected_items() . . . 1845
13.145.2.33 handle() . . . 1845
13.145.2.34 hposition() [1/2] . . . 1846
13.145.2.35 hposition() [2/2] . . . 1846
13.145.2.36 insert() . . . 1846
13.145.2.37 insert_above() . . . 1847
13.145.2.38 is_close() [1/2] . . . 1847
13.145.2.39 is_close() [2/2] . . . 1848
13.145.2.40 is_hscroll_visible() . . . 1848
13.145.2.41 is_open() [1/2] . . . 1848
13.145.2.42 is_open() [2/2] . . . 1849
13.145.2.43 is_scrollbar() . . . 1849
13.145.2.44 is_selected() [1/2] . . . 1849

13.145.2.45 is_selected() [2/2] . . . 1850
13.145.2.46 is_vscroll_visible() . . . 1850
13.145.2.47 item_clicked() [1/2] . . . 1850
13.145.2.48 item_clicked() [2/2] . . . 1850
13.145.2.49 item_draw_mode() [1/3] . . . 1850
13.145.2.50 item_draw_mode() [2/3] . . . 1851
13.145.2.51 item_draw_mode() [3/3] . . . 1851
13.145.2.52 item_labelbgcolor() [1/2] . . . 1851
13.145.2.53 item_labelbgcolor() [2/2] . . . 1851
13.145.2.54 item_labelfgcolor() . . . 1851
13.145.2.55 item_labelfont() . . . 1851
13.145.2.56 item_labelsize() . . . 1851
13.145.2.57 item_pathname() . . . 1852
13.145.2.58 item_reselect_mode() [1/2] . . . 1852
13.145.2.59 item_reselect_mode() [2/2] . . . 1852
13.145.2.60 last() . . . 1852
13.145.2.61 last_selected_item() . . . 1853
13.145.2.62 last_visible() . . . 1853
13.145.2.63 last_visible_item() . . . 1853
13.145.2.64 load() . . . 1853
13.145.2.65 next() . . . 1854
13.145.2.66 next_item() . . . 1854
13.145.2.67 next_selected_item() . . . 1855
13.145.2.68 next_visible_item() . . . 1856
13.145.2.69 open() [1/2] . . . 1856
13.145.2.70 open() [2/2] . . . 1857
13.145.2.71 open_toggle() . . . 1857
13.145.2.72 openicon() [1/2] . . . 1858
13.145.2.73 openicon() [2/2] . . . 1858
13.145.2.74 prev() . . . 1858
13.145.2.75 recalc_tree() . . . 1858
13.145.2.76 remove() . . . 1858
13.145.2.77 resize() . . . 1859
13.145.2.78 root() . . . 1859
13.145.2.79 root_label() . . . 1859
13.145.2.80 scrollbar_size() [1/2] . . . 1859
13.145.2.81 scrollbar_size() [2/2] . . . 1860
13.145.2.82 select() [1/2] . . . 1860
13.145.2.83 select() [2/2] . . . 1860
13.145.2.84 select_all() . . . 1861
13.145.2.85 select_only() . . . 1861
13.145.2.86 select_toggle() . . . 1862

13.145.2.87 selectbox() [1/2] . . . 1862
13.145.2.88 selectbox() [2/2] . . . 1862
13.145.2.89 selectmode() [1/2] . . . 1862
13.145.2.90 selectmode() [2/2] . . . 1863
13.145.2.91 set_item_focus() . . . 1863
13.145.2.92 show_item() [1/2] . . . 1863
13.145.2.93 show_item() [2/2] . . . 1863
13.145.2.94 show_item_bottom() . . . 1863
13.145.2.95 show_item_middle() . . . 1864
13.145.2.96 show_item_top() . . . 1864
13.145.2.97 show_self() . . . 1864
13.145.2.98 showcollapse() [1/2] . . . 1864
13.145.2.99 showcollapse() [2/2] . . . 1864
13.145.2.100 showroot() . . . 1865
13.145.2.101 sortorder() . . . 1865
13.145.2.102 usericon() [1/2] . . . 1865
13.145.2.103 usericon() [2/2] . . . 1865
13.145.2.104 vposition() [1/2] . . . 1865
13.145.2.105 vposition() [2/2] . . . 1865
13.146 Fl_Tree_Item Class Reference . . . 1866
13.146.1 Detailed Description . . . 1871
13.146.2 Constructor & Destructor Documentation . . . 1871
13.146.2.1 Fl_Tree_Item() [1/2] . . . 1871
13.146.2.2 Fl_Tree_Item() [2/2] . . . 1871
13.146.3 Member Function Documentation . . . 1872
13.146.3.1 activate() . . . 1872
13.146.3.2 add() [1/4] . . . 1872
13.146.3.3 add() [2/4] . . . 1872
13.146.3.4 add() [3/4] . . . 1872
13.146.3.5 add() [4/4] . . . 1873
13.146.3.6 calc_item_height() . . . 1873
13.146.3.7 child() . . . 1873
13.146.3.8 deactivate() . . . 1873
13.146.3.9 deparent() . . . 1873
13.146.3.10 depth() . . . 1874
13.146.3.11 deselect_all() . . . 1874
13.146.3.12 draw() . . . 1874
13.146.3.13 draw_horizontal_connector() . . . 1874
13.146.3.14 draw_item_content() . . . 1875
13.146.3.15 draw_vertical_connector() . . . 1876
13.146.3.16 drawbgcolor() . . . 1876
13.146.3.17 drawfgcolor() . . . 1876

13.146.3.18 find_child() [1/2] . . . 1876
13.146.3.19 find_child() [2/2] . . . 1877
13.146.3.20 find_child_item() [1/2] . . . 1877
13.146.3.21 find_child_item() [2/2] . . . 1877
13.146.3.22 find_clicked() . . . 1877
13.146.3.23 find_item() . . . 1878
13.146.3.24 hide_widgets() . . . 1878
13.146.3.25 insert() . . . 1878
13.146.3.26 insert_above() . . . 1878
13.146.3.27 is_visible_r() . . . 1878
13.146.3.28 label() . . . 1879
13.146.3.29 label_h() . . . 1879
13.146.3.30 label_w() . . . 1879
13.146.3.31 label_x() . . . 1879
13.146.3.32 label_y() . . . 1879
13.146.3.33 labelbgcolor() [1/2] . . . 1879
13.146.3.34 labelbgcolor() [2/2] . . . 1879
13.146.3.35 move() [1/2] . . . 1880
13.146.3.36 move() [2/2] . . . 1880
13.146.3.37 move_above() . . . 1880
13.146.3.38 move_below() . . . 1881
13.146.3.39 move_into() . . . 1881
13.146.3.40 next() . . . 1881
13.146.3.41 next_displayed() . . . 1881
13.146.3.42 next_sibling() . . . 1881
13.146.3.43 next_visible() . . . 1882
13.146.3.44 parent() . . . 1882
13.146.3.45 prefs() . . . 1882
13.146.3.46 prev() . . . 1882
13.146.3.47 prev_displayed() . . . 1882
13.146.3.48 prev_sibling() . . . 1882
13.146.3.49 prev_visible() . . . 1883
13.146.3.50 recalc_tree() . . . 1883
13.146.3.51 remove_child() [1/2] . . . 1883
13.146.3.52 remove_child() [2/2] . . . 1883
13.146.3.53 reparent() . . . 1883
13.146.3.54 replace() . . . 1884
13.146.3.55 replace_child() . . . 1884
13.146.3.56 select() . . . 1885
13.146.3.57 select_all() . . . 1885
13.146.3.58 show_self() . . . 1885
13.146.3.59 show_widgets() . . . 1885

13.146.3.60 swap_children() [1/2] . . . 1885
13.146.3.61 swap_children() [2/2] . . . 1885
13.146.3.62 tree() [1/2] . . . 1886
13.146.3.63 tree() [2/2] . . . 1886
13.146.3.64 update_prev_next() . . . 1886
13.146.3.65 userdeicon() [1/2] . . . 1886
13.146.3.66 userdeicon() [2/2] . . . 1886
13.146.3.67 usericon() . . . 1887
13.146.3.68 visible_r() . . . 1887
13.147 Fl_Tree_Item_Array Class Reference . . . 1887
13.147.1 Detailed Description . . . 1888
13.147.2 Constructor & Destructor Documentation . . . 1888
13.147.2.1 Fl_Tree_Item_Array() . . . 1888
13.147.3 Member Function Documentation . . . 1888
13.147.3.1 add() . . . 1888
13.147.3.2 clear() . . . 1888
13.147.3.3 deparent() . . . 1889
13.147.3.4 insert() . . . 1889
13.147.3.5 manage_item_destroy() . . . 1889
13.147.3.6 move() . . . 1889
13.147.3.7 remove() [1/2] . . . 1889
13.147.3.8 remove() [2/2] . . . 1889
13.147.3.9 reparent() . . . 1890
13.147.3.10 replace() . . . 1890
13.148 Fl_Tree_Prefs Class Reference . . . 1890
13.148.1 Detailed Description . . . 1893
13.148.2 Member Function Documentation . . . 1893
13.148.2.1 closedeicon() . . . 1893
13.148.2.2 closeicon() . . . 1893
13.148.2.3 item_draw_mode() . . . 1893
13.148.2.4 item_labelbgcolor() [1/2] . . . 1893
13.148.2.5 item_labelbgcolor() [2/2] . . . 1894
13.148.2.6 marginbottom() . . . 1894
13.148.2.7 opendeicon() . . . 1894
13.148.2.8 openicon() [1/2] . . . 1894
13.148.2.9 openicon() [2/2] . . . 1894
13.148.2.10 selectmode() . . . 1894
13.148.2.11 showcollapse() . . . 1894
13.148.2.12 showroot() . . . 1895
13.148.2.13 sortorder() . . . 1895
13.148.2.14 userdeicon() . . . 1895
13.149 Fl_Valuator Class Reference . . . 1895

13.149.1 Detailed Description . . . 1903
13.149.2 Constructor & Destructor Documentation . . . 1903
13.149.2.1 Fl_Valuator() . . . 1903
13.149.3 Member Function Documentation . . . 1904
13.149.3.1 format() . . . 1904
13.149.3.2 format_str() . . . 1904
13.149.3.3 increment() . . . 1904
13.149.3.4 maximum() [1/2] . . . 1904
13.149.3.5 maximum() [2/2] . . . 1904
13.149.3.6 minimum() [1/2] . . . 1905
13.149.3.7 minimum() [2/2] . . . 1905
13.149.3.8 precision() . . . 1905
13.149.3.9 range() . . . 1905
13.149.3.10 round() . . . 1905
13.149.3.11 step() . . . 1905
13.149.3.12 value() [1/2] . . . 1905
13.149.3.13 value() [2/2] . . . 1906
13.149.3.14 value_damage() . . . 1906
13.150 Fl_Value_Input Class Reference . . . 1906
13.150.1 Detailed Description . . . 1914
13.150.2 Constructor & Destructor Documentation . . . 1915
13.150.2.1 Fl_Value_Input() . . . 1915
13.150.3 Member Function Documentation . . . 1915
13.150.3.1 cursor_color() [1/2] . . . 1915
13.150.3.2 cursor_color() [2/2] . . . 1915
13.150.3.3 draw() . . . 1915
13.150.3.4 handle() . . . 1915
13.150.3.5 resize() . . . 1916
13.150.3.6 shortcut() [1/2] . . . 1916
13.150.3.7 shortcut() [2/2] . . . 1916
13.150.3.8 soft() . . . 1917
13.150.3.9 textcolor() . . . 1917
13.150.3.10 textfont() [1/2] . . . 1917
13.150.3.11 textfont() [2/2] . . . 1917
13.150.3.12 textsize() [1/2] . . . 1917
13.150.3.13 textsize() [2/2] . . . 1917
13.151 Fl_Value_Output Class Reference . . . 1917
13.151.1 Detailed Description . . . 1926
13.151.2 Constructor & Destructor Documentation . . . 1926
13.151.2.1 Fl_Value_Output() . . . 1926
13.151.3 Member Function Documentation . . . 1926
13.151.3.1 draw() . . . 1926

13.151.3.2 handle() . . . 1926
13.151.3.3 soft() [1/2] . . . 1927
13.151.3.4 soft() [2/2] . . . 1927
13.151.3.5 textcolor() [1/2] . . . 1927
13.151.3.6 textcolor() [2/2] . . . 1927
13.151.3.7 textfont() [1/2] . . . 1927
13.151.3.8 textfont() [2/2] . . . 1927
13.151.3.9 textsize() . . . 1928
13.152 Fl_Value_Slider Class Reference . . . 1928
13.152.1 Detailed Description . . . 1937
13.152.2 Constructor & Destructor Documentation . . . 1937
13.152.2.1 Fl_Value_Slider() . . . 1937
13.152.3 Member Function Documentation . . . 1937
13.152.3.1 draw() . . . 1937
13.152.3.2 handle() . . . 1938
13.152.3.3 value_height() [1/2] . . . 1938
13.152.3.4 value_height() [2/2] . . . 1938
13.152.3.5 value_width() [1/2] . . . 1940
13.152.3.6 value_width() [2/2] . . . 1940
13.153 Fl_Widget Class Reference . . . 1940
13.153.1 Detailed Description . . . 1947
13.153.2 Member Enumeration Documentation . . . 1948
13.153.2.1 anonymous enum . . . 1948
13.153.3 Constructor & Destructor Documentation . . . 1949
13.153.3.1 Fl_Widget() . . . 1949
13.153.3.2 ~Fl_Widget() . . . 1950
13.153.4 Member Function Documentation . . . 1950
13.153.4.1 activate() . . . 1950
13.153.4.2 active() . . . 1950
13.153.4.3 active_r() . . . 1950
13.153.4.4 align() [1/2] . . . 1951
13.153.4.5 align() [2/2] . . . 1951
13.153.4.6 argument() [1/2] . . . 1951
13.153.4.7 argument() [2/2] . . . 1951
13.153.4.8 as_gl_window() . . . 1952
13.153.4.9 as_group() . . . 1952
13.153.4.10 as_window() . . . 1952
13.153.4.11 bind_deimage() [1/2] . . . 1953
13.153.4.12 bind_deimage() [2/2] . . . 1953
13.153.4.13 bind_image() [1/2] . . . 1953
13.153.4.14 bind_image() [2/2] . . . 1954
13.153.4.15 box() [1/2] . . . 1954

13.153.4.16 box() [2/2] . . . 1954
13.153.4.17 callback() [1/6] . . . 1954
13.153.4.18 callback() [2/6] . . . 1955
13.153.4.19 callback() [3/6] . . . 1955
13.153.4.20 callback() [4/6] . . . 1955
13.153.4.21 callback() [5/6] . . . 1955
13.153.4.22 callback() [6/6] . . . 1955
13.153.4.23 changed() . . . 1956
13.153.4.24 clear_active() . . . 1956
13.153.4.25 clear_changed() . . . 1956
13.153.4.26 clear_damage() . . . 1956
13.153.4.27 clear_output() . . . 1957
13.153.4.28 clear_visible() . . . 1957
13.153.4.29 clear_visible_focus() . . . 1957
13.153.4.30 color() [1/3] . . . 1957
13.153.4.31 color() [2/3] . . . 1957
13.153.4.32 color() [3/3] . . . 1958
13.153.4.33 contains() . . . 1958
13.153.4.34 copy_label() . . . 1958
13.153.4.35 copy_tooltip() . . . 1959
13.153.4.36 damage() [1/3] . . . 1959
13.153.4.37 damage() [2/3] . . . 1959
13.153.4.38 damage() [3/3] . . . 1959
13.153.4.39 deactivate() . . . 1960
13.153.4.40 default_callback() . . . 1960
13.153.4.41 deimage() [1/4] . . . 1960
13.153.4.42 deimage() [2/4] . . . 1960
13.153.4.43 deimage() [3/4] . . . 1961
13.153.4.44 deimage() [4/4] . . . 1961
13.153.4.45 deimage_bound() . . . 1961
13.153.4.46 do_callback() [1/3] . . . 1961
13.153.4.47 do_callback() [2/3] . . . 1962
13.153.4.48 do_callback() [3/3] . . . 1962
13.153.4.49 draw() . . . 1963
13.153.4.50 draw_focus() [1/3] . . . 1963
13.153.4.51 draw_focus() [2/3] . . . 1963
13.153.4.52 draw_focus() [3/3] . . . 1963
13.153.4.53 draw_label() [1/3] . . . 1964
13.153.4.54 draw_label() [2/3] . . . 1964
13.153.4.55 draw_label() [3/3] . . . 1964
13.153.4.56 h() [1/2] . . . 1965
13.153.4.57 h() [2/2] . . . 1965

13.153.4.58 handle() . . . 1965
13.153.4.59 hide() . . . 1965
13.153.4.60 horizontal_label_margin() [1/2] . . . 1966
13.153.4.61 horizontal_label_margin() [2/2] . . . 1966
13.153.4.62 image() [1/4] . . . 1966
13.153.4.63 image() [2/4] . . . 1966
13.153.4.64 image() [3/4] . . . 1966
13.153.4.65 image() [4/4] . . . 1967
13.153.4.66 image_bound() . . . 1967
13.153.4.67 inside() . . . 1967
13.153.4.68 is_label_copied() . . . 1967
13.153.4.69 label() [1/3] . . . 1968
13.153.4.70 label() [2/3] . . . 1968
13.153.4.71 label() [3/3] . . . 1968
13.153.4.72 label_image_spacing() [1/2] . . . 1968
13.153.4.73 label_image_spacing() [2/2] . . . 1969
13.153.4.74 label_shortcut() . . . 1969
13.153.4.75 labelcolor() [1/2] . . . 1969
13.153.4.76 labelcolor() [2/2] . . . 1969
13.153.4.77 labelfont() [1/2] . . . 1969
13.153.4.78 labelfont() [2/2] . . . 1970
13.153.4.79 labelsize() [1/2] . . . 1970
13.153.4.80 labelsize() [2/2] . . . 1970
13.153.4.81 labeltype() [1/2] . . . 1970
13.153.4.82 labeltype() [2/2] . . . 1971
13.153.4.83 measure_label() . . . 1971
13.153.4.84 needs_keyboard() [1/2] . . . 1971
13.153.4.85 needs_keyboard() [2/2] . . . 1971
13.153.4.86 output() . . . 1972
13.153.4.87 parent() [1/2] . . . 1972
13.153.4.88 parent() [2/2] . . . 1972
13.153.4.89 position() . . . 1972
13.153.4.90 redraw() . . . 1973
13.153.4.91 redraw_label() . . . 1973
13.153.4.92 resize() . . . 1973
13.153.4.93 selection_color() [1/2] . . . 1973
13.153.4.94 selection_color() [2/2] . . . 1974
13.153.4.95 send() . . . 1974
13.153.4.96 set_active() . . . 1974
13.153.4.97 set_changed() . . . 1974
13.153.4.98 set_output() . . . 1974
13.153.4.99 set_visible() . . . 1975

13.153.4.100 set_visible_focus() . . . 1975
13.153.4.101 shortcut_label() [1/2] . . . 1975
13.153.4.102 shortcut_label() [2/2] . . . 1975
13.153.4.103 show() . . . 1975
13.153.4.104 size() . . . 1976
13.153.4.105 take_focus() . . . 1976
13.153.4.106 takesevents() . . . 1976
13.153.4.107 test_shortcut() [1/2] . . . 1976
13.153.4.108 test_shortcut() [2/2] . . . 1976
13.153.4.109 tooltip() [1/2] . . . 1977
13.153.4.110 tooltip() [2/2] . . . 1977
13.153.4.111 top_window() . . . 1978
13.153.4.112 top_window_offset() . . . 1978
13.153.4.113 type() [1/2] . . . 1978
13.153.4.114 type() [2/2] . . . 1978
13.153.4.115 user_data() . . . 1978
13.153.4.116 vertical_label_margin() [1/2] . . . 1979
13.153.4.117 vertical_label_margin() [2/2] . . . 1979
13.153.4.118 visible() . . . 1979
13.153.4.119 visible_focus() [1/2] . . . 1979
13.153.4.120 visible_focus() [2/2] . . . 1979
13.153.4.121 visible_r() . . . 1980
13.153.4.122 w() [1/2] . . . 1980
13.153.4.123 w() [2/2] . . . 1980
13.153.4.124 when() [1/2] . . . 1980
13.153.4.125 when() [2/2] . . . 1980
13.153.4.126 window() . . . 1981
13.153.4.127 x() [1/2] . . . 1981
13.153.4.128 x() [2/2] . . . 1982
13.153.4.129 y() [1/2] . . . 1982
13.153.4.130 y() [2/2] . . . 1982
13.154 Fl_Widget_Surface Class Reference . . . 1982
13.154.1 Detailed Description . . . 1984
13.154.2 Constructor & Destructor Documentation . . . 1984
13.154.2.1 Fl_Widget_Surface() . . . 1984
13.154.3 Member Function Documentation . . . 1984
13.154.3.1 draw() . . . 1984
13.154.3.2 draw_decorated_window() . . . 1984
13.154.3.3 origin() [1/2] . . . 1984
13.154.3.4 origin() [2/2] . . . 1985
13.154.3.5 print_window_part() . . . 1985
13.154.3.6 printable_rect() . . . 1985

13.154.3.7 translate() . . . 1986
13.154.3.8 untranslate() . . . 1986
13.155 Fl_Widget_Tracker Class Reference . . . 1986
13.155.1 Detailed Description . . . 1987
13.155.2 Member Function Documentation . . . 1987
13.155.2.1 deleted() . . . 1987
13.155.2.2 exists() . . . 1987
13.155.2.3 widget() . . . 1987
13.156 Fl_Window Class Reference . . . 1988
13.156.1 Detailed Description . . . 2000
13.156.2 Constructor & Destructor Documentation . . . 2000
13.156.2.1 Fl_Window() [1/2] . . . 2000
13.156.2.2 Fl_Window() [2/2] . . . 2000
13.156.2.3 ~Fl_Window() . . . 2001
13.156.3 Member Function Documentation . . . 2001
13.156.3.1 allow_expand_outside_parent() . . . 2001
13.156.3.2 as_double_window() . . . 2001
13.156.3.3 as_overlay_window() . . . 2001
13.156.3.4 as_window() [1/2] . . . 2001
13.156.3.5 as_window() [2/2] . . . 2001
13.156.3.6 border() . . . 2002
13.156.3.7 clear_border() . . . 2002
13.156.3.8 clear_modal_states() . . . 2002
13.156.3.9 current() . . . 2002
13.156.3.10 cursor() [1/3] . . . 2003
13.156.3.11 cursor() [2/3] . . . 2003
13.156.3.12 cursor() [3/3] . . . 2003
13.156.3.13 decorated_h() . . . 2003
13.156.3.14 decorated_w() . . . 2004
13.156.3.15 default_cursor() [1/2] . . . 2004
13.156.3.16 default_cursor() [2/2] . . . 2004
13.156.3.17 default_icon() . . . 2004
13.156.3.18 default_icons() [1/2] . . . 2004
13.156.3.19 default_icons() [2/2] . . . 2005
13.156.3.20 default_size_range() . . . 2005
13.156.3.21 default_xclass() [1/2] . . . 2006
13.156.3.22 default_xclass() [2/2] . . . 2006
13.156.3.23 draw() . . . 2007
13.156.3.24 flush() . . . 2007
13.156.3.25 force_position() [1/2] . . . 2007
13.156.3.26 force_position() [2/2] . . . 2007
13.156.3.27 free_icons() . . . 2007

13.156.3.28 free_position() . . . 2008
13.156.3.29 fullscreen() . . . 2008
13.156.3.30 fullscreen_screens() . . . 2008
13.156.3.31 get_size_range() . . . 2008
13.156.3.32 handle() . . . 2009
13.156.3.33 hide() . . . 2009
13.156.3.34 hotspot() . . . 2010
13.156.3.35 icon() [1/3] . . . 2010
13.156.3.36 icon() [2/3] . . . 2010
13.156.3.37 icon() [3/3] . . . 2010
13.156.3.38 iconize() . . . 2011
13.156.3.39 icons() [1/2] . . . 2011
13.156.3.40 icons() [2/2] . . . 2011
13.156.3.41 is_a_rescale() . . . 2012
13.156.3.42 is_resizable() . . . 2012
13.156.3.43 make_current() . . . 2012
13.156.3.44 maximize() . . . 2012
13.156.3.45 modal() . . . 2013
13.156.3.46 os_id() . . . 2013
13.156.3.47 resize() . . . 2013
13.156.3.48 screen_num() [1/2] . . . 2013
13.156.3.49 screen_num() [2/2] . . . 2014
13.156.3.50 set_menu_window() . . . 2014
13.156.3.51 set_modal() . . . 2014
13.156.3.52 set_non_modal() . . . 2015
13.156.3.53 set_tooltip_window() . . . 2015
13.156.3.54 shape() [1/3] . . . 2015
13.156.3.55 shape() [2/3] . . . 2015
13.156.3.56 shape() [3/3] . . . 2015
13.156.3.57 show() [1/2] . . . 2016
13.156.3.58 show() [2/2] . . . 2016
13.156.3.59 show_next_window_iconic() [1/2] . . . 2018
13.156.3.60 show_next_window_iconic() [2/2] . . . 2018
13.156.3.61 shown() . . . 2018
13.156.3.62 size_range() . . . 2018
13.156.3.63 un_maximize() . . . 2019
13.156.3.64 wait_for_expose() . . . 2019
13.156.3.65 xclass() [1/2] . . . 2020
13.156.3.66 xclass() [2/2] . . . 2020
13.156.4 Member Data Documentation . . . 2021
13.156.4.1 current_ . . . 2021
13.157 Fl_Wizard Class Reference . . . 2021

13.157.1 Detailed Description . . . 2029
13.157.2 Constructor & Destructor Documentation . . . 2029
13.157.2.1 Fl_Wizard() . . . 2029
13.157.3 Member Function Documentation . . . 2030
13.157.3.1 draw() . . . 2030
13.157.3.2 next() . . . 2030
13.157.3.3 value() . . . 2030
13.158 Fl_XBM_Image Class Reference . . . 2030
13.158.1 Detailed Description . . . 2033
13.158.2 Constructor & Destructor Documentation . . . 2033
13.158.2.1 Fl_XBM_Image() . . . 2033
13.159 Fl_XPM_Image Class Reference . . . 2033
13.159.1 Detailed Description . . . 2036
13.159.2 Constructor & Destructor Documentation . . . 2036
13.159.2.1 Fl_XPM_Image() . . . 2036
13.160 Fl_GIF_Image::GIF_FRAME Struct Reference . . . 2036
13.161 Fl_ICO_Image::IconDirEntry Struct Reference . . . 2037
13.161.1 Detailed Description . . . 2037
13.162 Fl_Text_Editor::Key_Binding Struct Reference . . . 2037
13.162.1 Detailed Description . . . 2038
13.163 Fl_Terminal::Margin Class Reference . . . 2038
13.164 Fl_Preferences::Name Class Reference . . . 2038
13.164.1 Detailed Description . . . 2038
13.164.2 Constructor & Destructor Documentation . . . 2038
13.164.2.1 Name() [1/2] . . . 2038
13.164.2.2 Name() [2/2] . . . 2039
13.165 Fl_Preferences::Node Class Reference . . . 2039
13.166 Fl_Paged_Device::page_format Struct Reference . . . 2040
13.166.1 Detailed Description . . . 2040
13.167 Fl_Terminal::PartialUtf8Buf Class Reference . . . 2040
13.168 Fl_Terminal::RingBuffer Class Reference . . . 2040
13.169 Fl_Preferences::RootNode Class Reference . . . 2041
13.170 Fl_Scroll::Scrollbar_Data Struct Reference . . . 2041
13.170.1 Detailed Description . . . 2042
13.171 Fl_Scroll::ScrollInfo Struct Reference . . . 2042
13.171.1 Detailed Description . . . 2042
13.172 Fl_Terminal::Selection Class Reference . . . 2043
13.172.1 Member Function Documentation . . . 2043
13.172.1.1 get_selection() . . . 2043
13.173 Fl_Tile::Size_Range Struct Reference . . . 2043
13.174 Fl_Text_Display::Style_Table_Entry Struct Reference . . . 2044
13.174.1 Detailed Description . . . 2044

13.175 Fl::TimeoutData Struct Reference . . . 2044
13.176 Fl_Terminal::Utf8Char Class Reference . . . 2044

14 File Documentation **2047**
14.1 fl_config.h . . . 2047
14.2 events.H File Reference . . . 2048
14.2.1 Detailed Description . . . 2051
14.3 events.H . . . 2051
14.4 function_types.H File Reference . . . 2054
14.4.1 Detailed Description . . . 2055
14.5 function_types.H . . . 2055
14.6 options.H File Reference . . . 2056
14.6.1 Detailed Description . . . 2057
14.7 options.H . . . 2057
14.8 pen_events.H File Reference . . . 2058
14.8.1 Detailed Description . . . 2060
14.9 pen_events.H . . . 2060
14.10 Enumerations.H File Reference . . . 2062
14.10.1 Detailed Description . . . 2073
14.10.2 Macro Definition Documentation . . . 2073
14.10.2.1 FL_IMAGE_LABEL . . . 2073
14.10.2.2 FL_MULTI_LABEL . . . 2073
14.10.2.3 FL_SYMBOL_LABEL . . . 2074
14.10.3 Typedef Documentation . . . 2074
14.10.3.1 Fl_Contrast_Function . . . 2074
14.10.3.2 Fl_Fontsize . . . 2074
14.10.4 Enumeration Type Documentation . . . 2074
14.10.4.1 anonymous enum . . . 2074
14.10.4.2 Fl_Arrow_Type . . . 2074
14.10.4.3 Fl_Boxtype . . . 2075
14.10.4.4 Fl_Callback_Reason . . . 2076
14.10.4.5 Fl_Contrast_Mode . . . 2077
14.10.4.6 Fl_Cursor . . . 2077
14.10.4.7 Fl_Damage . . . 2078
14.10.4.8 Fl_Event . . . 2078
14.10.4.9 Fl_Labeltype . . . 2082
14.10.4.10 Fl_Orientation . . . 2082
14.10.4.11 Fl_When . . . 2083
14.10.5 Function Documentation . . . 2084
14.10.5.1 fl_box() . . . 2084
14.10.5.2 fl_color_cube() . . . 2084
14.10.5.3 fl_define_FL_EMBOSSED_LABEL() . . . 2084

14.10.5.4 fl_define_FL_ENGRAVED_LABEL() . . . 2084
14.10.5.5 fl_define_FL_ICON_LABEL() . . . 2084
14.10.5.6 fl_define_FL_IMAGE_LABEL() . . . 2084
14.10.5.7 fl_define_FL_MULTI_LABEL() . . . 2084
14.10.5.8 fl_define_FL_SHADOW_LABEL() . . . 2084
14.10.5.9 fl_down() . . . 2085
14.10.5.10 fl_frame() . . . 2085
14.10.5.11 fl_gray_ramp() . . . 2085
14.10.6 Variable Documentation . . . 2085
14.10.6.1 FL_ALIGN_LEFT . . . 2085
14.10.6.2 FL_ALIGN_TOP . . . 2085
14.10.6.3 FL_NORMAL_SIZE . . . 2085
14.11 Enumerations.H . . . 2085
14.12 filename.H File Reference . . . 2093
14.12.1 Detailed Description . . . 2095
14.13 filename.H . . . 2095
14.14 Fl.H File Reference . . . 2096
14.14.1 Detailed Description . . . 2103
14.15 Fl.H . . . 2104
14.16 Fl_Adjuster.H . . . 2108
14.17 Fl_Anim_GIF_Image.H . . . 2108
14.18 fl_ask.H File Reference . . . 2110
14.18.1 Detailed Description . . . 2111
14.18.2 Enumeration Type Documentation . . . 2112
14.18.2.1 Fl_Beep . . . 2112
14.18.3 Function Documentation . . . 2112
14.18.3.1 fl_message_position() . . . 2112
14.19 fl_ask.H . . . 2112
14.20 fl_attr.h File Reference . . . 2114
14.20.1 Detailed Description . . . 2114
14.20.2 Macro Definition Documentation . . . 2114
14.20.2.1 __fl_attr . . . 2114
14.20.2.2 FL_DEPRECATED . . . 2114
14.20.2.3 FL_OVERRIDE . . . 2115
14.21 fl_attr.h . . . 2115
14.22 Fl_Bitmap.H . . . 2117
14.23 Fl_BMP_Image.H . . . 2118
14.24 Fl_Box.H File Reference . . . 2118
14.24.1 Detailed Description . . . 2118
14.25 Fl_Box.H . . . 2118
14.26 Fl_Browser.H . . . 2119
14.27 Fl_Browser_.H . . . 2121

14.28 Fl_Button.H . . . 2122
14.29 Fl_Cairo.H File Reference . . . 2124
14.29.1 Detailed Description . . . 2124
14.30 Fl_Cairo.H . . . 2124
14.31 Fl_Cairo_Window.H File Reference . . . 2125
14.31.1 Detailed Description . . . 2125
14.32 Fl_Cairo_Window.H . . . 2125
14.33 fl_callback_macros.H File Reference . . . 2126
14.33.1 Detailed Description . . . 2126
14.33.2 Macro Definition Documentation . . . 2126
14.33.2.1 FL_FUNCTION_CALLBACK_3 . . . 2126
14.33.2.2 FL_INLINE_CALLBACK_2 . . . 2127
14.33.2.3 FL_METHOD_CALLBACK_1 . . . 2128
14.34 fl_callback_macros.H . . . 2129
14.35 fl_casts.H . . . 2134
14.36 Fl_Chart.H File Reference . . . 2135
14.36.1 Detailed Description . . . 2135
14.37 Fl_Chart.H . . . 2135
14.38 Fl_Check_Browser.H . . . 2137
14.39 Fl_Check_Button.H . . . 2138
14.40 Fl_Choice.H . . . 2138
14.41 Fl_Clock.H . . . 2139
14.42 Fl_Color_Chooser.H File Reference . . . 2140
14.42.1 Detailed Description . . . 2140
14.43 Fl_Color_Chooser.H . . . 2140
14.44 Fl_Copy_Surface.H . . . 2142
14.45 Fl_Counter.H . . . 2142
14.46 Fl_Device.H File Reference . . . 2143
14.46.1 Detailed Description . . . 2144
14.47 Fl_Device.H . . . 2144
14.48 Fl_Dial.H . . . 2145
14.49 Fl_Double_Window.H . . . 2145
14.50 fl_draw.H File Reference . . . 2146
14.50.1 Detailed Description . . . 2152
14.51 fl_draw.H . . . 2152
14.52 Fl_Export.H . . . 2157
14.53 Fl_File_Browser.H . . . 2158
14.54 Fl_File_Chooser.H . . . 2159
14.55 Fl_File_Icon.H . . . 2161
14.56 Fl_File_Input.H . . . 2162
14.57 Fl_Fill_Dial.H . . . 2163
14.58 Fl_Fill_Slider.H . . . 2164

14.59 Fl_Flex.H . . . 2164
14.60 Fl_Float_Input.H . . . 2166
14.61 Fl_FormsBitmap.H . . . 2166
14.62 Fl_FormsPixmap.H . . . 2167
14.63 Fl_Free.H . . . 2167
14.64 Fl_GIF_Image.H . . . 2168
14.65 Fl_Gl_Window.H . . . 2169
14.66 Fl_Graphics_Driver.H . . . 2170
14.67 Fl_Grid.H File Reference . . . 2173
14.67.1 Detailed Description . . . 2174
14.68 Fl_Grid.H . . . 2174
14.69 Fl_Group.H File Reference . . . 2177
14.69.1 Detailed Description . . . 2177
14.70 Fl_Group.H . . . 2178
14.71 Fl_Help_Dialog.H . . . 2179
14.72 Fl_Help_View.H . . . 2180
14.73 Fl_Hold_Browser.H . . . 2181
14.74 Fl_Hor_Fill_Slider.H . . . 2182
14.75 Fl_Hor_Nice_Slider.H . . . 2182
14.76 Fl_Hor_Slider.H . . . 2183
14.77 Fl_Hor_Value_Slider.H . . . 2183
14.78 Fl_ICO_Image.H . . . 2183
14.79 Fl_Image.H File Reference . . . 2184
14.79.1 Detailed Description . . . 2184
14.79.2 Enumeration Type Documentation . . . 2185
14.79.2.1 Fl_RGB_Scaling . . . 2185
14.80 Fl_Image.H . . . 2185
14.81 Fl_Image_Surface.H . . . 2187
14.82 Fl_Input.H . . . 2188
14.83 Fl_Input_.H . . . 2189
14.84 Fl_Input_Choice.H . . . 2192
14.85 Fl_Int_Input.H . . . 2194
14.86 Fl_JPEG_Image.H . . . 2194
14.87 Fl_Light_Button.H . . . 2195
14.88 Fl_Line_Dial.H . . . 2195
14.89 Fl_Menu.H . . . 2195
14.90 Fl_Menu_.H . . . 2196
14.91 Fl_Menu_Bar.H . . . 2197
14.92 Fl_Menu_Button.H . . . 2198
14.93 Fl_Menu_Item.H File Reference . . . 2198
14.93.1 Enumeration Type Documentation . . . 2199
14.93.1.1 anonymous enum . . . 2199

14.94 Fl_Menu_Item.H . . . 2199
14.95 Fl_Menu_Window.H . . . 2202
14.96 fl_message.H . . . 2202
14.97 Fl_Multi_Browser.H . . . 2202
14.98 Fl_Multi_Label.H . . . 2203
14.99 Fl_Multiline_Input.H . . . 2203
14.100 Fl_Multiline_Output.H . . . 2204
14.101 Fl_Native_File_Chooser.H File Reference . . . 2204
14.101.1 Detailed Description . . . 2204
14.102 Fl_Native_File_Chooser.H . . . 2204
14.103 Fl_Nice_Slider.H . . . 2207
14.104 Fl_Object.H . . . 2207
14.105 Fl_Output.H . . . 2207
14.106 Fl_Overlay_Window.H . . . 2208
14.107 Fl_Pack.H . . . 2208
14.108 Fl_Paged_Device.H File Reference . . . 2209
14.108.1 Detailed Description . . . 2209
14.109 Fl_Paged_Device.H . . . 2209
14.110 Fl_PDF_File_Surface.H . . . 2211
14.111 Fl_Pixmap.H . . . 2211
14.112 Fl_Plugin.H . . . 2212
14.113 Fl_PNG_Image.H . . . 2213
14.114 Fl_PNM_Image.H . . . 2214
14.115 Fl_Positioner.H . . . 2214
14.116 Fl_PostScript.H File Reference . . . 2215
14.116.1 Detailed Description . . . 2215
14.116.2 Typedef Documentation . . . 2215
14.116.2.1 Fl_PostScript_Close_Command . . . 2215
14.117 Fl_PostScript.H . . . 2216
14.118 Fl_Preferences.H . . . 2217
14.119 Fl_Printer.H File Reference . . . 2220
14.119.1 Detailed Description . . . 2220
14.120 Fl_Printer.H . . . 2220
14.121 Fl_Progress.H . . . 2221
14.122 Fl_Radio_Button.H . . . 2221
14.123 Fl_Radio_Light_Button.H . . . 2222
14.124 Fl_Radio_Round_Button.H . . . 2222
14.125 Fl_Rect.H . . . 2223
14.126 Fl_Repeat_Button.H . . . 2224
14.127 Fl_Return_Button.H . . . 2224
14.128 Fl_RGB_Image.H . . . 2225
14.129 Fl_Roller.H . . . 2225

14.130 Fl_Round_Button.H . . . 2226
14.131 Fl_Round_Clock.H . . . 2226
14.132 Fl_Scheme.H . . . 2226
14.133 Fl_Scheme_Choice.H . . . 2227
14.134 Fl_Scroll.H . . . 2228
14.135 Fl_Scrollbar.H . . . 2229
14.136 Fl_Secret_Input.H . . . 2230
14.137 Fl_Select_Browser.H . . . 2230
14.138 Fl_Shared_Image.H File Reference . . . 2230
14.138.1 Detailed Description . . . 2231
14.138.2 Typedef Documentation . . . 2231
14.138.2.1 Fl_Shared_Handler . . . 2231
14.138.3 Function Documentation . . . 2232
14.138.3.1 fl_register_images() . . . 2232
14.139 Fl_Shared_Image.H . . . 2232
14.140 Fl_Shortcut_Button.H . . . 2233
14.141 fl_show_colormap.H File Reference . . . 2234
14.141.1 Detailed Description . . . 2234
14.142 fl_show_colormap.H . . . 2234
14.143 fl_show_input.H . . . 2234
14.144 Fl_Simple_Counter.H . . . 2235
14.145 Fl_Single_Window.H . . . 2235
14.146 Fl_Slider.H . . . 2236
14.147 Fl_Spinner.H . . . 2237
14.148 fl_string_functions.h File Reference . . . 2238
14.148.1 Detailed Description . . . 2239
14.149 fl_string_functions.h . . . 2239
14.150 Fl_SVG_File_Surface.H . . . 2239
14.151 Fl_SVG_Image.H . . . 2240
14.152 Fl_Sys_Menu_Bar.H File Reference . . . 2241
14.152.1 Detailed Description . . . 2241
14.153 Fl_Sys_Menu_Bar.H . . . 2241
14.154 Fl_Table.H . . . 2242
14.155 Fl_Table_Row.H . . . 2248
14.156 Fl_Tabs.H . . . 2249
14.157 Fl_Terminal.H File Reference . . . 2250
14.157.1 Detailed Description . . . 2251
14.158 Fl_Terminal.H . . . 2251
14.159 Fl_Text_Buffer.H . . . 2260
14.160 Fl_Text_Display.H . . . 2264
14.161 Fl_Text_Editor.H . . . 2269
14.162 Fl_Tile.H . . . 2270

14.163 Fl_Tiled_Image.H . . . 2271
14.164 Fl_Timer.H . . . 2272
14.165 Fl_Toggle_Button.H . . . 2272
14.166 Fl_Toggle_Light_Button.H . . . 2273
14.167 Fl_Toggle_Round_Button.H . . . 2273
14.168 Fl_Tooltip.H . . . 2274
14.169 Fl_Tree.H File Reference . . . 2275
14.169.1 Detailed Description . . . 2275
14.169.2 Enumeration Type Documentation . . . 2275
14.169.2.1 Fl_Tree_Reason . . . 2275
14.170 Fl_Tree.H . . . 2275
14.171 Fl_Tree_Item.H File Reference . . . 2278
14.171.1 Detailed Description . . . 2279
14.172 Fl_Tree_Item.H . . . 2279
14.173 Fl_Tree_Item_Array.H File Reference . . . 2283
14.173.1 Detailed Description . . . 2283
14.174 Fl_Tree_Item_Array.H . . . 2283
14.175 Fl_Tree_Prefs.H File Reference . . . 2284
14.175.1 Detailed Description . . . 2284
14.175.2 Enumeration Type Documentation . . . 2284
14.175.2.1 Fl_Tree_Connector . . . 2284
14.175.2.2 Fl_Tree_Item_Draw_Mode . . . 2285
14.175.2.3 Fl_Tree_Item_Reselect_Mode . . . 2285
14.175.2.4 Fl_Tree_Select . . . 2285
14.175.2.5 Fl_Tree_Sort . . . 2285
14.176 Fl_Tree_Prefs.H . . . 2286
14.177 fl_types.h File Reference . . . 2289
14.177.1 Detailed Description . . . 2289
14.177.2 Typedef Documentation . . . 2289
14.177.2.1 Fl_Shortcut . . . 2289
14.178 fl_types.h . . . 2290
14.179 fl_utf8.h File Reference . . . 2290
14.179.1 Detailed Description . . . 2292
14.180 fl_utf8.h . . . 2293
14.181 Fl_Valuator.H . . . 2295
14.182 Fl_Value_Input.H . . . 2296
14.183 Fl_Value_Output.H . . . 2297
14.184 Fl_Value_Slider.H . . . 2297
14.185 Fl_Widget.H File Reference . . . 2298
14.185.1 Detailed Description . . . 2299
14.185.2 Macro Definition Documentation . . . 2299
14.185.2.1 FL_RESERVED_TYPE . . . 2299

14.186 Fl_Widget.H . . . 2299
14.187 Fl_Widget_Surface.H . . . 2304
14.188 Fl_Widget_Tracker.H File Reference . . . 2305
14.188.1 Detailed Description . . . 2305
14.189 Fl_Widget_Tracker.H . . . 2305
14.190 Fl_Window.H File Reference . . . 2306
14.190.1 Detailed Description . . . 2306
14.191 Fl_Window.H . . . 2307
14.192 Fl_Wizard.H . . . 2309
14.193 Fl_XBM_Image.H . . . 2310
14.194 Fl_XPM_Image.H . . . 2310
14.195 forms.H . . . 2311
14.196 gl.h File Reference . . . 2321
14.196.1 Detailed Description . . . 2322
14.196.2 Function Documentation . . . 2322
14.196.2.1 gl_color() . . . 2322
14.196.2.2 gl_draw() [1/7] . . . 2322
14.196.2.3 gl_draw() [2/7] . . . 2322
14.196.2.4 gl_draw() [3/7] . . . 2322
14.196.2.5 gl_draw() [4/7] . . . 2323
14.196.2.6 gl_draw() [5/7] . . . 2323
14.196.2.7 gl_draw() [6/7] . . . 2323
14.196.2.8 gl_draw() [7/7] . . . 2323
14.196.2.9 gl_font() . . . 2323
14.196.2.10 gl_rect() . . . 2324
14.196.2.11 gl_rectf() . . . 2324
14.196.2.12 gl_texture_pile_height() [1/2] . . . 2324
14.196.2.13 gl_texture_pile_height() [2/2] . . . 2324
14.197 gl.h . . . 2324
14.198 gl2opengl.h . . . 2325
14.199 gl_draw.H . . . 2326
14.200 glu.h . . . 2326
14.201 glut.H . . . 2327
14.202 mac.H File Reference . . . 2333
14.202.1 Detailed Description . . . 2333
14.203 mac.H . . . 2334
14.204 math.h . . . 2335
14.205 names.h File Reference . . . 2336
14.205.1 Detailed Description . . . 2337
14.206 names.h . . . 2337
14.207 platform.H . . . 2338
14.208 platform_types.h File Reference . . . 2339

14.208.1 Detailed Description . . . 2340
14.208.2 Typedef Documentation . . . 2340
14.208.2.1 fl_intptr_t . . . 2340
14.208.2.2 Fl_Offscreen . . . 2340
14.208.2.3 Fl_Region . . . 2340
14.208.2.4 Fl_Timestamp . . . 2341
14.208.2.5 fl_uintptr_t . . . 2341
14.208.2.6 GLContext . . . 2341
14.209 platform_types.h . . . 2341
14.210 rgb_colors.H . . . 2342
14.211 wayland.H File Reference . . . 2348
14.211.1 Detailed Description . . . 2349
14.211.2 Function Documentation . . . 2349
14.211.2.1 fl_wl_compositor() . . . 2349
14.211.2.2 fl_wl_gc() . . . 2349
14.212 wayland.H . . . 2349
14.213 win32.H File Reference . . . 2350
14.213.1 Detailed Description . . . 2350
14.214 win32.H . . . 2350
14.215 win32_target.h . . . 2351
14.216 x.H . . . 2352
14.217 x11.H File Reference . . . 2352
14.217.1 Detailed Description . . . 2352
14.217.2 Function Documentation . . . 2352
14.217.2.1 fl_x11_find() . . . 2352
14.217.2.2 fl_x11_gc() . . . 2353
14.217.2.3 fl_x11_use_display() . . . 2353
14.217.2.4 fl_x11_xid() . . . 2353
14.218 x11.H . . . 2353
14.219 cgdebug.h . . . 2354
14.220 fastarrow.h . . . 2357
14.221 Fl.cxx File Reference . . . 2357
14.221.1 Detailed Description . . . 2358
14.221.2 Function Documentation . . . 2358
14.221.2.1 fl_close_display() . . . 2358
14.221.2.2 fl_find() . . . 2358
14.221.2.3 fl_open_display() . . . 2358
14.221.3 Variable Documentation . . . 2359
14.221.3.1 fl_disable_wayland . . . 2359
14.222 fl_arc.cxx File Reference . . . 2359
14.222.1 Detailed Description . . . 2359
14.223 fl_ask.cxx File Reference . . . 2359

14.223.1 Detailed Description . . . 2361
14.224 fl_boxtype.cxx File Reference . . . 2361
14.224.1 Detailed Description . . . 2363
14.224.2 Function Documentation . . . 2363
14.224.2.1 fl_rectbound() . . . 2363
14.225 fl_cmap.h . . . 2364
14.226 fl_color.cxx File Reference . . . 2367
14.226.1 Detailed Description . . . 2367
14.226.2 Variable Documentation . . . 2367
14.226.2.1 fl_cmap . . . 2367
14.227 Fl_compose.cxx File Reference . . . 2367
14.227.1 Detailed Description . . . 2367
14.228 fl_contrast.cxx File Reference . . . 2368
14.228.1 Detailed Description . . . 2368
14.229 fl_curve.cxx File Reference . . . 2368
14.229.1 Detailed Description . . . 2368
14.230 Fl_Double_Window.cxx File Reference . . . 2369
14.230.1 Detailed Description . . . 2369
14.231 Fl_get_system_colors.cxx File Reference . . . 2369
14.231.1 Detailed Description . . . 2370
14.231.2 Function Documentation . . . 2370
14.231.2.1 fl_parse_color() . . . 2370
14.232 Fl_Gl_Choice.H . . . 2370
14.233 Fl_Gl_Window_Driver.H . . . 2371
14.234 Fl_Graphics_Driver.cxx File Reference . . . 2372
14.234.1 Detailed Description . . . 2373
14.235 Fl_Grid.cxx File Reference . . . 2373
14.235.1 Detailed Description . . . 2373
14.236 Fl_Image_Reader.h . . . 2373
14.237 Fl_Message.h . . . 2374
14.238 Fl_Native_File_Chooser_Kdialog.H . . . 2376
14.239 Fl_Native_File_Chooser_Zenity.H . . . 2377
14.240 fl_oxy.h . . . 2377
14.241 Fl_Paged_Device.cxx File Reference . . . 2378
14.241.1 Detailed Description . . . 2378
14.242 fl_plastic.cxx File Reference . . . 2378
14.242.1 Detailed Description . . . 2378
14.243 Fl_Private.H File Reference . . . 2378
14.243.1 Detailed Description . . . 2379
14.243.2 Function Documentation . . . 2379
14.243.2.1 cairo_make_current() . . . 2379
14.243.2.2 set_idle_() . . . 2379

14.243.3 Variable Documentation . . . 2380
14.243.3.1 idle_ . . . 2380
14.244 Fl_Private.H . . . 2380
14.245 fl_rect.cxx File Reference . . . 2381
14.245.1 Detailed Description . . . 2381
14.246 Fl_Scalable_Graphics_Driver.H . . . 2381
14.247 Fl_Scheme.cxx File Reference . . . 2383
14.247.1 Detailed Description . . . 2383
14.248 Fl_Screen_Driver.H . . . 2383
14.249 Fl_Sys_Menu_Bar_Driver.H . . . 2386
14.250 Fl_System_Driver.H . . . 2387
14.251 Fl_Timeout.cxx File Reference . . . 2389
14.252 Fl_Timeout.h File Reference . . . 2389
14.252.1 Detailed Description . . . 2390
14.253 Fl_Timeout.h . . . 2390
14.254 fl_vertex.cxx File Reference . . . 2391
14.254.1 Detailed Description . . . 2391
14.255 Fl_Window_Driver.H . . . 2392
14.256 fl_write_jpeg.cxx File Reference . . . 2394
14.256.1 Detailed Description . . . 2394
14.256.2 Function Documentation . . . 2394
14.256.2.1 fl_write_jpeg() [1/3] . . . 2394
14.256.2.2 fl_write_jpeg() [2/3] . . . 2395
14.256.2.3 fl_write_jpeg() [3/3] . . . 2395
14.257 fl_write_png.cxx File Reference . . . 2396
14.257.1 Detailed Description . . . 2396
14.257.2 Function Documentation . . . 2396
14.257.2.1 fl_write_png() [1/3] . . . 2396
14.257.2.2 fl_write_png() [2/3] . . . 2397
14.257.2.3 fl_write_png() [3/3] . . . 2397
14.258 Fl_XColor.H . . . 2398
14.259 flstring.h . . . 2398
14.260 freeglut_teapot_data.h . . . 2402
14.261 mediumarrow.h . . . 2403
14.262 numericsort.c File Reference . . . 2404
14.262.1 Function Documentation . . . 2404
14.262.1.1 fl_casenumericsort() . . . 2404
14.262.1.2 fl_numericsort() . . . 2404
14.263 print_button.h . . . 2405
14.264 print_panel.h . . . 2406
14.265 slowarrow.h . . . 2406
14.266 utf8_internal.h . . . 2406

14.267 vsnprintf.c File Reference 2407
14.267.1 Detailed Description 2407
14.267.2 Function Documentation 2407
14.267.2.1 fl_vsnprintf() 2407
14.268 Xutf8.h 2408

Index **2411**

Chapter 1

FLTK Programming Manual

FLTK 1.5.0 Programming Manual

By F. Costantini, M. Melcher, A. Schlosser, B. Spitzak and M. Sweet.

Copyright © 1998 - 2026 by Bill Spitzak and others.

This software and manual are provided under the terms of the GNU Library General Public License. Permission is granted to reproduce this manual or any portion for any purpose, provided this copyright and permission notice are preserved.

Preface Introduction to FLTK FLTK Basics Common Widgets and Attributes • Colors • Box Types • Labels and Label Types • Drawing Images Coordinates and Layout Widgets How Does Resizing Work? Designing a Simple Text Editor Drawing Things in FLTK • When Can You Draw Things in FLTK? • What Units Do FLTK Functions Use? • Drawing Functions • Drawing Images • Offscreen Drawing Handling Events • Fl::event_∗() methods • Event Propagation	Adding and Extending Widgets Using OpenGL FLTK Runtime Options Advanced FLTK Unicode and UTF-8 Support **Appendices:** • Constants and Enumerations • GLUT Compatibility – class Fl_Glut_Window • Forms Compatibility • Operating System Issues • Migrating Code from FLTK 1.3 to 1.4 • Migrating Code from FLTK 1.4 to 1.5 • Software License • Example Source Code • FAQ (Frequently Asked Questions)

1.1 Preface

This manual describes the Fast Light Tool Kit ("FLTK")

1.5.0 , a C++ Graphical User Interface ("GUI") toolkit for UNIX, Microsoft Windows, and Apple macOS.

Version 1.4.0 introduced support for a new windowing system under Linux/Unix: Wayland. FLTK applications under Linux/Unix run unchanged as Wayland or X11 clients depending on availability at run-time.

Each of the chapters in this manual is designed as a tutorial for using FLTK, while the appendices provide a convenient reference for all FLTK widgets, functions, and operating system interfaces.

This manual may be printed, modified, and/or used under the terms of the FLTK license provided in↩ : Software License.

1.1.1 Organization

This manual is organized into the following chapters and appendices:

- Introduction to FLTK
- FLTK Basics

- Common Widgets and Attributes
- Designing a Simple Text Editor
- Drawing Things in FLTK
- Handling Events
- Adding and Extending Widgets
- Using OpenGL
- FLTK Runtime Options
- Advanced FLTK
- Unicode and UTF-8 Support
- Constants and Enumerations
- GLUT Compatibility
- Forms Compatibility
- Operating System Issues
- Migrating Code from FLTK 1.3 to 1.4
- Developer Information
- Software License
- Example Source Code

1.1.2 Conventions

This manual was generated using Doxygen (see `https://www.doxygen.org/`) to process the source code itself, special comments in the code, and additional documentation files. In general, Doxygen recognizes and denotes the following entities as shown:

- classes, such as Fl_Widget,
- methods, such as Fl_Widget::callback(Fl_Callback* cb, void* p),
- functions, such as fl_draw(const char *str, int x, int y),
- internal links, such as Conventions,
- external links, such as `https://www.fltk.org/`.

Other code samples and commands are shown in `regular courier type`.

1.1.3 Abbreviations

The following abbreviations are used in this manual:

X11

The X Window System version 11.

Xlib

The X Window System interface library.

Windows, `WIN32`

The Microsoft Windows Application Programmer's Interface for Windows 2000, Windows XP, Windows Vista, Windows 7 and later Windows versions. FLTK uses the preprocessor definition `_WIN32` for the 32 bit and 64 bit Windows API.

macOS (aka Mac OS X), `APPLE`

The Apple desktop operating sytem macOS 10.0 and later. MacOS 8 and 9 support was dropped after FLTK 1.0.10. FLTK uses the preprocessor definition `__APPLE__` for macOS.

1.1.4 Copyrights and Trademarks

FLTK is Copyright © 1998 - 2026 by Bill Spitzak and others. Use and distribution of FLTK is governed by the GNU Library General Public License with 4 exceptions, located in Software License.

UNIX is a registered trademark of the X Open Group, Inc. Microsoft and Windows are registered trademarks of Microsoft Corporation. OpenGL is a registered trademark of Silicon Graphics, Inc. Apple, Macintosh, MacOS, macOS, and Mac OS X are registered trademarks of Apple Computer, Inc.

1.2 Introduction to FLTK

The Fast Light Tool Kit ("FLTK") is a cross-platform C++ GUI toolkit for UNIX®/Linux® (X11 and Wayland), Microsoft® Windows®, and Apple® macOS®.

FLTK provides modern GUI functionality without bloat and supports 3D graphics via OpenGL® and its built-in GLUT emulation. It was originally developed by Mr. Bill Spitzak and is currently maintained by a small group of developers across the world with a central repository on GitHub.

1.2.1 History of FLTK

It has always been Bill's belief that the GUI API of all modern systems is much too high level. Toolkits (even FLTK) are *not* what should be provided and documented as part of an operating system. The system only has to provide arbitrary shaped but featureless windows, a powerful set of graphics drawing calls, and a simple *unalterable* method of delivering events to the owners of the windows. NeXT (if you ignored NextStep) provided this, but they chose to hide it and tried to push their own baroque toolkit instead.

Many of the ideas in FLTK were developed on a NeXT (but *not* using NextStep) in 1987 in a C toolkit Bill called "views". Here he came up with passing events downward in the tree and having the handle routine return a value indicating whether it used the event, and the table-driven menus. In general he was trying to prove that complex UI ideas could be entirely implemented in a user space toolkit, with no knowledge or support by the system.

After going to film school for a few years, Bill worked at Sun Microsystems on the (doomed) NeWS project. Here he found an even better and cleaner windowing system, and he reimplemented "views" atop that. NeWS did have an unnecessarily complex method of delivering events which hurt it. But the designers did admit that perhaps the user could write just as good of a button as they could, and officially exposed the lower level interface.

With the death of NeWS Bill realized that he would have to live with X. The biggest problem with X is the "window manager", which means that the toolkit can no longer control the window borders or drag the window around.

At Digital Domain Bill discovered another toolkit, "Forms". Forms was similar to his work, but provided many more widgets, since it was used in many real applications, rather than as theoretical work. He decided to use Forms, except he integrated his table-driven menus into it. Several very large programs were created using this version of Forms.

The need to switch to OpenGL and GLX, portability, and a desire to use C++ subclassing required a rewrite of Forms. This produced the first version of FLTK. The conversion to C++ required so many changes it made it impossible to recompile any Forms objects. Since it was incompatible anyway, Bill decided to incorporate his older ideas as much as possible by simplifying the lower level interface and the event passing mechanism.

Bill received permission to release it for free on the Internet, with the GNU general public license. Response from Internet users indicated that the Linux market dwarfed the SGI and high-speed GL market, so he rewrote it to use X for all drawing, greatly speeding it up on these machines.

Digital Domain has since withdrawn support for FLTK. While Bill is no longer able to actively develop it, he still contributes to FLTK in his free time and is a part of the FLTK development team.

FLTK was later ported to Windows and macOS. FLTK 1.4 added a "driver based" system of virtual device drivers that enabled porting to Wayland as well. Drawing features include Windows GDI+, Cairo (Wayland and X11), and improved text layout with Pango.

There have been experiments using this driver system to build FLTK based on SDL2, Android, and other graphics systems based solely on simple pixel drawing, but this experimental code is not included in FLTK 1.4. There are thoughts to enable more platforms in later FLTK versions.

FLTK 1.5 removed autotools (configure + Makefile) support in favor of exclusive use of CMake to generate the build files. FLTK 1.5 allows to use a basic set of C++ features from the C++11 standard.

1.2.2 Features

FLTK was designed to be statically linked. This was done by splitting it into many small objects and designing it so that functions that are not used do not have pointers to them in the parts that are used, and thus do not get linked in. This allows you to make an easy-to-install program or to modify FLTK to the exact requirements of your application without worrying about bloat. FLTK works fine as a shared library, though, and is included with several Linux distributions.

Here are some of the core features unique to FLTK:

Note: sizes given below are mostly from 32-bit systems and FLTK 1.1 or earlier, this list needs updates for current FLTK 1.5 (64-bit).

- sizeof(Fl_Widget) == 64 to 92 (120 in FLTK 1.4 on 64-bit Linux).
- The "core" (the "hello" program compiled & linked with a static FLTK library using gcc on a 486 and then stripped) is 114K. (FLTK 1.4 on 64-bit Linux: 1.1 MB).
- The FLUID program (which includes every widget) is 538k. (FLTK 1.4 with more widgets on 64-bit Linux: 2.3 MB and 2.0 MB on 32-bit Windows).
- Written directly atop core libraries (Xlib, Wayland, Windows or Cocoa) for maximum speed and carefully optimized for code size and performance.
- Precise low-level compatibility between the X11/Wayland, Windows, and macOS versions - only about 10% of the code is different.
- Interactive user interface builder program FLUID. Its output is human-readable and editable C++ source code.
- Support for overlay hardware, with emulation if none is available. Note: hardware support has been removed in 1.4.0.
- Very small & fast portable 2-D drawing library to hide Xlib, Cairo, Windows, or macOS Quartz. Since FLTK 1.4 Wayland graphics use Cairo for drawing. This is optionally available for X11 drawing as well.
- OpenGL/Mesa drawing area widget.
- Support for OpenGL overlay hardware on both X11 and Windows, with emulation if none is available.
- Text widgets with cut & paste, undo, and support for Unicode text and international input methods.
- Compatibility header file for the GLUT library.
- Compatibility header file for the XForms library.

1.2.3 Licensing

FLTK comes with complete free source code. FLTK is available under the terms of the GNU Library General Public License with exceptions that allow for static linking. Contrary to popular belief, it can be used in commercial software - even Bill Gates could use it!

1.2.4 What Does "FLTK" Mean?

FLTK was originally designed to be compatible with the Forms Library written for SGI machines. In that library all the functions and structures started with "fl_". This naming was extended to all new methods and widgets in the C++ library, and this prefix was taken as the name of the library. It is almost impossible to search for "FL" on the Internet, due to the fact that it is also the abbreviation for Florida. After much debating and searching for a new name for the toolkit, which was already in use by several people, Bill came up with "FLTK", including a bogus excuse that it stands for "The Fast Light Toolkit".

1.2.5 FLUID

FLTK comes bundled with FLUID. FLUID, short for Fast Light User Interface Designer, is a graphical editor capable of generating C++ source code and header files ready for compilation. These files ultimately create the graphical user interface for an application.

The FLUID User Handbook is available at `https://www.fltk.org/documentation.php`. It can also be compiled from the FLTK source repository using the `fluid_docs` target in the CMake build environment.

1.2.6 Building and Installing FLTK with CMake

Starting with version 1.4, the **recommended** FLTK build system was CMake. Since FLTK 1.5.0 CMake is the **only** available build system for FLTK.

CMake is a "Build System Generator" that can generate build environments for usage with Ninja, Make, and many more, for instance several IDE's. See file README.CMake.txt of the FLTK source tree for more information.

By default CMake will look for a C++ compiler named `CC`, `c++`, `g++`, or `gcc`. To use another compiler you can set the `CXX` and `CC` environment variables for the C++ and C compiler, respectively. The code to use depends on your shell, for instance:

```
CXX=clang++; export CXX
setenv CXX "clang++"

CC=clang; export CC
setenv CC "clang"
```

Then execute CMake like

```
cd /path/to/fltk
cmake . -B build [ -G <GENERATOR> ] [ options ... ]
```

Note that you can also specify the compiler and build tools on the CMake command line or in the CMake GUI.

When CMake is done you can just run CMake again to build the FLTK library, fluid, fltk-options (setup tool), and all test programs, depending on the configuration.

```
cmake --build build
```

To install the library, become root and type "cmake --install build". This will copy the "fluid" executable to "bindir", the header files to "includedir", and the library files to "libdir".

1.2.7 Building FLTK Under Microsoft Windows

NOTE: This documentation section is currently under review. More up-to-date information for this release may be available in the files "README.Windows.txt" and "README.CMake.txt" and you should read these files to determine if there are changes that may be applicable to your build environment.

FLTK 1.5 is **officially supported** on Windows 10 (11) and later. We don't intend to remove the ability to build and run FLTK 1.5 programs on older Windows versions but the FLTK Team can't build and test on Windows systems that are no longer supported by Microsoft. Therefore, such older systems (including Windows 10, starting in Oct. 2025, when Microsoft does no longer support Windows 10) may work with FLTK or not.

Note

FLTK 1.4 is known to work on Windows (2000,) 2003, XP, and later. The main prerequisite is that the OS version supports UTF-8. FLTK 1.4 is also known to work on recent versions of Windows such as Windows 7, Windows 8/8.1, Windows 10, and Windows 11, and has been reported to work in both 32-bit and 64-bit Windows versions. If you need to use FLTK on older versions than Windows 11 (or Windows 10) you may use FLTK 1.4.x.

Note

Libraries built by any one of the following build environments can not be mixed with object files from any of the other environments because they use incompatible C++ conventions internally.

FLTK currently supports the following development environments on the Windows platform:

1.2.7.1 Free and Commercial Microsoft Visual Studio Versions

Visual Studio 2015 Community or later versions use workspace and project files generated by CMake. Older versions and the commercial versions can be used as well, if they can open the project files generated by CMake. However, FLTK 1.5 and later requires newer C++ features (C++11 or higher) that may not be available in older versions of Visual Studio.

FLTK support of Visual Studio is limited to CMake's support for these Visual Studio versions. Be sure to get your service packs!

Since FLTK 1.5 the project files MUST be generated with CMake. Please read "README.txt", "README.CMake.↩ txt", and "README.Windows.txt" for more information about this.

1.2.7.2 Using the Visual C++ DLL Library

The Visual Studio project files can be used to build a DLL version of the FLTK library if CMake option 'FLTK_↩ BUILD_SHARED_LIBS=ON' is set. Because of name mangling differences between PC compilers (even between different versions of Visual Studio) you can only use the DLL that is generated with the same compiler version that you built it with.

When compiling an application or DLL that uses the FLTK DLL with Visual Studio, you need to define the `FL_DLL` preprocessor symbol to get the correct linkage commands embedded within the FLTK header files.

New since FLTK 1.4.0:

If you build your application project with CMake and use the CMake target 'fltk::fltk-shared' to link your application, then 'FL_DLL' is defined automatically for you (by CMake Compile Definitions). If you use your own (hand-made) Visual Studio project you still need to define FL_DLL to compile all source files that use FLTK headers.

1.2.7.3 GNU toolsets (Cygwin or MinGW) hosted on Windows

If using Cygwin with the Cygwin shell, or MinGW with the Msys shell, these build environments behave very much like a Unix or macOS build and the notes above can be applied. Build files are generated by CMake for the platform you are using. See README.CMake.txt for more info.

In general for a build using these tools, e.g. for the Msys shell with MinGW, it should suffice to "cd" into the directory where you have extracted the FLTK tarball and type:

```
cmake . -B build
cmake --build build
```

This will build the FLTK libraries and they can then be utilised directly from the build location. NOTE: this may be simpler than "installing" them in many cases as different tool chains on Windows have different ideas about where the files should be "installed" to.

For example, if you "install" the libraries using Msys/MinGW with the following command

```
cmake --install build
```

then Msys will "install" the libraries to where it thinks the path "/usr/local/" leads to. If you only ever build code from within the Msys environment this works well, but the actual "Windows path" these files are located in will be something like "C:\msys\1.0\local\lib", depending on where your Msys installation is rooted, which may not be useful to other tools.

If you want to install your built FLTK libraries in a non-standard location you may do:

```
cmake . -B build -D CMAKE_INSTALL_PATH="C:/FLTK"
cmake --build build
cmake --install build
```

Where the value passed to "CMAKE_INSTALL_PATH" is the path at which you would like FLTK to be installed. Depending on the build environment using standard Windows path syntax may work, or you need to use MinGW (or Cygwin) syntax.

1.2.8 Internet Resources

FLTK is available on the internet in a bunch of locations:

FLTK Source Repository on GitHub

https://github.com/fltk/fltk

WWW (fltk.org)

https://www.fltk.org/
https://www.fltk.org/bugs.php [for reporting bugs]
https://www.fltk.org/software.php [download source code]
https://www.fltk.org/newsgroups.php [newsgroup/forums]

User Forums and NNTP Newsgroups

https://groups.google.com/forum/#!forum/fltkgeneral [Google Groups interface]
news://fltk.org:1024/ [NNTP interface]
https://www.fltk.org/newsgroups.php [web interface]

GitHub

https://github.com/fltk/fltk https://github.com/fltk/fltk/discussions [FLTK Discussions on GitHub] https://github.com/fltk/fltk/discussions/categories/q-a [Q&A: Questions and Answers] https://github.com/fltk/fltk/releases [Releases, i.e. Source Code and Documentation]

1.2.9 Reporting Bugs

To report a bug in FLTK, or for feature requests, please use https://www.fltk.org/bugs.php for information about where and how to post bugs, feature requests, or ask for help on using FLTK.

For general support and questions, please use the fltk.general newsgroup (see above, "NNTP Newsgroups") or the web interface to the newsgroups at https://www.fltk.org/newsgroups.php, or GitHub Discussions (see link above).

1.3 FLTK Basics

This chapter teaches you the basics of writing and compiling programs that use FLTK.

1.3.1 Writing Your First FLTK Program

Up to FLTK 1.3.x all FLTK programs were required to include the file <FL/Fl.H> as the first FLTK header file.

Since FLTK 1.4.0 this requirement was relaxed and <FL/Fl.H> needs only be included if the class Fl is used or if some other stuff like enumerations is used in the source code. Example code in this documentation may still include it "everywhere" even if it is no longer strictly required.

In addition the program must include a header file for each FLTK class it uses. Listing 1 shows a simple "Hello, World!" program that uses FLTK to display the window.

Listing 1 - "hello.cxx"

```
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Box.H>

int main(int argc, char **argv) {
  Fl_Window *window = new Fl_Window(340, 180);
  Fl_Box *box = new Fl_Box(20, 40, 300, 100, "Hello, World!");
  box->box(FL_UP_BOX);
  box->labelfont(FL_BOLD + FL_ITALIC);
  box->labelsize(36);
  box->labeltype(FL_SHADOW_LABEL);
  window->end();
  window->show(argc, argv);
  return Fl::run();
}
```

After including the required header files, the program then creates a window. All following widgets will automatically be children of this window.

```
Fl_Window *window = new Fl_Window(340, 180);
```

Then we create a box with the "Hello, World!" string in it. FLTK automatically adds the new box to `window`, the current grouping widget.

```
Fl_Box *box = new Fl_Box(20, 40, 300, 100, "Hello, World!");
```

Next, we set the type of box and the font, size, and style of the label:

```
box->box(FL_UP_BOX);
box->labelfont(FL_BOLD + FL_ITALIC);
box->labelsize(36);
box->labeltype(FL_SHADOW_LABEL);
```

We tell FLTK that we will not add any more widgets to `window`.

```
window->end();
```

Finally, we show the window and enter the FLTK event loop:

```
window->show(argc, argv);
return Fl::run();
```

The resulting program will display the "Hello, World!" window:

Figure 1.1 The Hello, World! Window

You can quit the program by closing the window or pressing the `ESC`ape key.

1.3.1.1 Creating the Widgets

The widgets are created using the C++ `new` operator. For most widgets the arguments to the constructor are:
`Fl_Widget(x, y, width, height, label)`

The `x` and `y` parameters determine where the widget or window is placed on the screen. In FLTK the top left corner of the window or screen is the origin (i.e. `x = 0, y = 0`).

The `width` and `height` parameters determine the size of the widget or window. The maximum widget size is typically governed by the underlying window system or hardware.

What Units Do FLTK Functions Use? describes the unit FLTK employs for `x, y, width`, and `height`, and more generally, for all graphical quantities.

`label` is a pointer to a character string to label the widget with or `NULL`. If not specified the label defaults to `NULL`. The label string must be in static storage such as a string constant because FLTK does not make a copy of it - it just uses the pointer.

1.3.1.2 Creating Widget Hierarchies

Widgets are commonly ordered into functional groups, which in turn may be grouped again, creating a hierarchy of widgets. FLTK makes it easy to fill groups by automatically adding all widgets that are created between a `my↩Group->begin()` and `myGroup->end()`. In this example, `myGroup` would be the *current* group.

Newly created groups and their derived widgets implicitly call `begin()` in the constructor, effectively adding all subsequently created widgets to itself until `end()` is called.

Calling end() on one group widget transfers the "current group" property to the **parent** of that widget. Calling end() on a top level window (which has no parent) sets the current group to `NULL`.

Setting the current group to `NULL` will stop automatic hierarchies. New widgets can now be added manually using `Fl_Group::add(...)` and `Fl_Group::insert(...)`.

1.3.1.3 Get/Set Methods

`box->box(FL_UP_BOX)` sets the type of box the Fl_Box draws, changing it from the default of `FL_NO_BOX`, which means that no box is drawn. In our "Hello, World!" example we use `FL_UP_BOX`, which means that a raised button border will be drawn around the widget. More details are available in the Box Types section.

You could examine the boxtype by doing `box->box()`. FLTK uses method name overloading to make short names for get/set methods. A "set" method is always of the form "void name(type)", and a "get" method is always of the form "type name() const".

1.3.1.4 Redrawing After Changing Attributes

Almost all of the get/set pairs are very fast, short inline functions and thus very efficient. However, *the "set" methods do not call `redraw()`* - you have to call it yourself. This greatly reduces code size and execution time. The only common exceptions are `value()` which calls `redraw()` and `label()` which calls `redraw_label()` if necessary.

1.3.1.5 Labels

All widgets support labels. In the case of window widgets, the label is used for the label in the title bar. Our example program calls the `labelfont()`, `labelsize()`, and `labeltype()` methods.

The `labelfont()` method sets the typeface and style that is used for the label, which for this example we are using `FL_BOLD` and `FL_ITALIC`.

The `labelsize()` method sets the height of the font in FLTK units.

The `labeltype()` method sets the type of label. FLTK supports normal, embossed, and shadowed labels internally, and more types can be added as desired.

A complete list of all label options can be found in the section on Labels and Label Types.

1.3.1.6 Showing the Window

The `show()` method shows the widget or window. For windows you can also provide the command-line arguments to allow users to customize the appearance, size, and position of your windows.

1.3.1.7 The Main Event Loop

All FLTK applications (and most GUI applications in general) are based on a simple event processing model. User actions such as mouse movement, button clicks, and keyboard activity generate events that are sent to an application. The application may then ignore the events or respond to the user, typically by redrawing a button in the "down" position, adding the text to an input field, and so forth.

FLTK also supports idle, timer, and file pseudo-events that cause a function to be called when they occur. Idle functions are called when no user input is present and no timers or files need to be handled - in short, when the application is not doing anything. Idle callbacks are often used to update a 3D display or do other background processing.

Timer functions are called after a specific amount of time has expired. They can be used to pop up a progress dialog after a certain amount of time or do other things that need to happen at more-or-less regular intervals. FLTK timers are not 100% accurate, so they should not be used to measure time intervals, for example.

File functions are called when data is ready to read or write, or when an error condition occurs on a file. They are most often used to monitor network connections (sockets) for data-driven displays.

FLTK applications must periodically check (Fl::check()) or wait (Fl::wait()) for events or use the Fl::run() method to enter a standard event processing loop. Calling Fl::run() is equivalent to the following code:

```
while (Fl::wait());
```

Fl::run() does not return until all of the windows under FLTK control are closed by the user or your program.

1.3.2 Naming Conventions

All public symbols in FLTK start with the characters 'F' and 'L':

- Functions are either `Fl::foo()` or `fl_foo()`.
- Class and type names are capitalized: `Fl_Foo`.
- Constants and Enumerations are uppercase: `FL_FOO`.
- All header files start with `<FL/...>`.

1.3.3 Header Files

The proper way to include FLTK header files is:

```
#include <FL/Fl_xyz.H>
```

Note

Case *is significant* on many operating systems, and the C standard uses the forward slash (/) to separate directories. *Do not use any of the following include lines:*

```
#include <FL\Fl_xyz.H>
#include <fl/fl_xyz.h>
#include <Fl/fl_xyz.h>
```

1.3.4 Compiling Programs that Use FLTK

Since FLTK 1.4 CMake is the recommended build system. The details below show the "old" methods and reference information in case you like to write your build configuration manually (e.g. Makefiles, Visual Studio, other IDE's ...).

CMake can simplify this task substantially. For now, refer to README.CMake.txt for further information.

Todo This section needs a major rework. Add a chapter "Building FLTK with CMake".

1.3.4.1 Compiling Programs with Standard Compilers

Under UNIX (and under Microsoft Windows when using the GNU development tools) you will probably need to tell the compiler where to find the header files. This is usually done using the `-I` option:

```
c++ -I/usr/local/include ...
```

Note

You need a C++ compiler to build FLTK. The commands given in this chapter are **examples** using `'c++'`. Please replace this command with the C++ compiler suitable for your system or use the `fltk-config` script as described below (this is recommended).

1.3.4.2 Compiling Programs with the 'fltk-config' Script

The `fltk-config` script included with FLTK can be used on systems with a Posix compliant shell, for instance Unix/Linux, macOS, Windows with MinGW, MSYS2, or Cygwin.

Note

> `fltk-config` is not designed to work on Windows with Visual Studio compilers. If it works, then only by accident and this is undefined behavior.

```
fltk-config --help
```

displays all available options.

`fltk-config` can be used to get the compiler and the options that are required by your compiler to build a program using the FLTK library:

```
fltk-config --cc
fltk-config --cxx
```

return the C and C++ compiler commands used to build FLTK.

```
c++ `fltk-config --cxxflags` ...
```

can be used to include the required compiler flags in the command line.

Similarly, when linking your application you will need to tell the compiler to use the FLTK library:

```
c++ ... -L/usr/local/lib -lfltk -lXext -lX11 ... -lm -ldl
```

Aside from the "fltk" library, there are also the following libraries

- "fltk_forms" for the XForms compatibility classes (deprecated)
- "fltk_gl" for the OpenGL and GLUT classes
- "fltk_images" for the image file classes, Fl_Help_Dialog widget, and system icon support.

The libraries are named `fltk.lib`, `fltk_forms.lib`, `fltk_gl.lib`, and `fltk_images.lib` under Windows.

Note

> The separate `fltk_cairo` library is no longer necessary since FLTK 1.4.0. It is no longer built since FLTK 1.5.0.

As before, the `fltk-config` script can be used to get the options that are required by your linker:

```
c++ ... `fltk-config --ldflags`
```

The forms, GL, and images libraries are included with the "--use-foo" options, as follows:

```
c++ ... `fltk-config --use-forms  --ldflags`
c++ ... `fltk-config --use-gl     --ldflags`
c++ ... `fltk-config --use-images --ldflags`
c++ ... `fltk-config --use-cairo  --ldflags`
c++ ... `fltk-config --use-forms --use-gl --use-images --ldflags`
```

The option `--use-cairo` may be used to build your program with Cairo libs if you use Cairo in your code. It does no longer include the `fltk_cairo` lib but all necessary Cairo compiler flags and Cairo libs, if and only if FLTK has been configured with the optional Cairo support by CMake.

Finally, you can use the `fltk-config` script to compile one or more source files as a FLTK program.

The following examples will create an executable named `filename` (or `filename.exe` under Windows) from a single source file:

```
fltk-config --compile filename.cxx
fltk-config --use-forms --compile filename.cpp
fltk-config --use-gl --compile filename.C
fltk-config --use-images --compile filename.cc
fltk-config --use-cairo --compile filename.cpp
fltk-config --use-forms --use-gl --use-images --compile filename.cpp
```

Note

> `'fltk-config --compile'` accepts only a limited set of file extensions for C++ source files: `'`.cpp', `'`.cxx', `'`.cc', and `'`.C' (capital 'C').

1.3.4.3 Compiling Multiple Source Files with 'fltk-config'

Before version 1.4.0 `fltk-config` accepted only a single source file and no additional compiler options or libraries. Since FLTK 1.4.0 it is possible to use additional compiler flags, more than one source file, and additional link libraries.

This is intended to be used for quick prototyping and not for production code development. It can be used to test compiler command options (like `-Wall` or `-Wextra`) or additional link libraries if these are required.

Building from more than one source file with flags and libraries can be achieved as follows:

```
fltk-config [USE-FLAGS] --compile MAIN [FLAGS] [SOURCES] [--link LFLAGS LIBS]
```

where

- arguments in `[...]` are optional
- `USE-FLAGS` are as described above, e.g. `--use-images`
- `MAIN` is the main C++ source file as documented above
- `FLAGS` are additional compiler flags
- `SOURCES` are additional source files or libraries
- `--link` is used to separate source files and flags from linker flags and libs
- `LFLAGS` are optional linker flags
- `LIBS` are additional libraries to link against

The final commandline is composed like this example:

```
$ fltk-config --compile main.cxx button.o -Wextra x1.a --link -L/usr/include/cairo/ -lcairo

g++ {fltk-flags} -o main -Wextra main.cxx button.o x1.a {fltk-libs} -L/usr/include/cairo/ -lcairo
```

where `{fltk-flags}` are the compiler flags generated by `fltk-config` as before and `{fltk-libs}` are the usual linker flags and libraries. All optional parameters are used as-is, i.e. there is no syntax checking or special parsing except: the order of flags and source files is preserved (from the commandline) but all flags (`-something`) are positioned before all sources, i.e. arguments w/o leading dash ('-'). All compiler flags and libraries generated from the library build follow all options and source files given on the commandline, and finally everything after `--link` is appended.

1.3.4.4 Compiling Programs with Makefiles

The previous sections described how to use `fltk-config` to build a program from the command line, and this is very convenient for small test programs. But `fltk-config` can also be used to set the compiler and linker options as variables within a `Makefile` that can be used to build larger programs.

```
CXX      = $(shell fltk-config --cxx)
DEBUG    = -g
CXXFLAGS = $(shell fltk-config --use-gl --use-images --cxxflags ) -I.
LDFLAGS  = $(shell fltk-config --use-gl --use-images --ldflags )
LDSTATIC = $(shell fltk-config --use-gl --use-images --ldstaticflags )
LINK     = $(CXX)

TARGET = cube
OBJS   = CubeMain.o CubeView.o CubeViewUI.o
SRCS   = CubeMain.cxx CubeView.cxx CubeViewUI.cxx

.SUFFIXES: .o .cxx

%.o: %.cxx
        $(CXX) $(CXXFLAGS) $(DEBUG) -c $<

all: $(TARGET)

$(TARGET): $(OBJS)
        $(LINK) -o $(TARGET) $(OBJS) $(LDSTATIC)

CubeMain.o:   CubeMain.cxx CubeViewUI.h
CubeView.o:   CubeView.cxx CubeView.h CubeViewUI.h
CubeViewUI.o: CubeViewUI.cxx CubeView.h

clean:
        rm -f *.o 2> /dev/null
        rm -f $(TARGET) 2> /dev/null
```

1.3.4.5 Compiling Programs with Microsoft Visual C++

In Visual C++ you will need to tell the compiler where to find the FLTK header files. This can be done by selecting "Settings" from the "Project" menu and then changing the "Preprocessor" settings under the "C/C++" tab.

You will also need to add the following libraries to the `Linker` settings:

- `fltk.lib` or `fltkd.lib`, the main FLTK library (postfix 'd' = Debug)
- all FLTK libraries your program requires (fltk_gl, fltk_images, ...)
- additional libraries like `libpng.lib`, `libjpeg.lib`, etc.
- the Windows Common Controls (`comctl32.lib`) and
- the GDIplus library if used to build FLTK (`gdiplus.lib`) and
- the Windows Socket (`ws2_32.lib`) libraries.

Note

There's a `Linker` setting "Additional Library Directories" or similar; the exact name depends on the Visual Studio version you're using. You can and **should** use this to simplify adding the libraries above. If you set this to the FLTK library path you can just use the library **names** and don't need to use the full paths to all libraries.

You must also define `_WIN32` if the compiler doesn't do this. Currently all known Windows compilers define _WIN32 - unless you use Cygwin (that's correct, you must not define _WIN32 if you use Cygwin).

More information can be found in `README.Windows.txt`.

You can build your Microsoft Windows applications as Console or Desktop applications. If you want to use the standard C `main()` function as the entry point, FLTK includes a `WinMain()` function that will call your `main()` function for you.

1.4 Common Widgets and Attributes

This chapter describes many of the widgets that are provided with FLTK and covers how to query and set the standard attributes.

1.4.1 Buttons

FLTK provides many types of buttons:

- Fl_Button - A standard push button.
- Fl_Check_Button - A button with a check box.
- Fl_Light_Button - A push button with a light.
- Fl_Repeat_Button - A push button that repeats when held.

- Fl_Return_Button - A push button that is activated by the `Enter` key.
- Fl_Round_Button - A button with a radio circle.

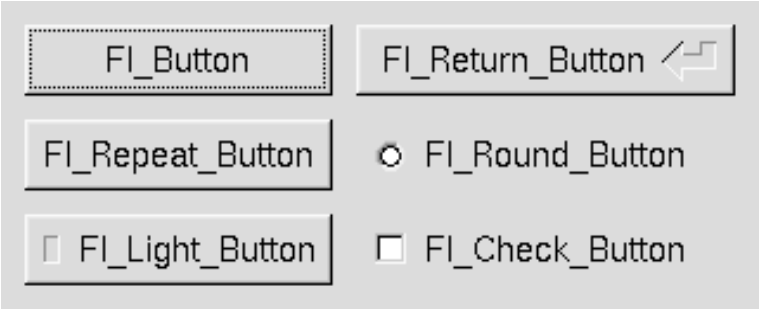


Figure 1.2 FLTK Button Widgets

All of these buttons just need the corresponding `<FL/Fl_xyz_Button.H>` header file. The constructor takes the bounding box of the button and optionally a label string:

```
Fl_Button *button = new Fl_Button(x, y, width, height, "label");
Fl_Light_Button *lbutton = new Fl_Light_Button(x, y, width, height);
Fl_Round_Button *rbutton = new Fl_Round_Button(x, y, width, height, "label");
```

Each button has an associated `type()` which allows it to behave as a push button, toggle button, or radio button:

```
button->type(FL_NORMAL_BUTTON);
lbutton->type(FL_TOGGLE_BUTTON);
rbutton->type(FL_RADIO_BUTTON);
```

For toggle and radio buttons, the `value()` method returns the current button state (0 = off, 1 = on). The `set()` and `clear()` methods can be used on toggle buttons to turn a toggle button on or off, respectively. Radio buttons can be turned on with the `setonly()` method; this will also turn off other radio buttons in the same group.

1.4.2 Text

FLTK provides several text widgets for displaying and receiving text:

- Fl_Input - A one-line text input field.
- Fl_Output - A one-line text output field.
- Fl_Multiline_Input - A multi-line text input field.
- Fl_Multiline_Output - A multi-line text output field.
- Fl_Text_Display - A multi-line text display widget.
- Fl_Text_Editor - A multi-line text editing widget.
- Fl_Help_View - A HTML text display widget.

The Fl_Output and Fl_Multiline_Output widgets allow the user to copy text from the output field but not change it.

The `value()` method is used to get or set the string that is displayed:

```
Fl_Input *input = new Fl_Input(x, y, width, height, "label");
input->value("Now is the time for all good men...");
```

The string is copied to the widget's own storage when you set the `value()` of the widget.

The Fl_Text_Display and Fl_Text_Editor widgets use an associated Fl_Text_Buffer class for the value, instead of a simple string.

1.4.3 Valuators

Unlike text widgets, valuators keep track of numbers instead of strings. FLTK provides the following valuators:

- Fl_Counter - A widget with arrow buttons that shows the current value.
- Fl_Dial - A round knob.
- Fl_Roller - An SGI-like dolly widget.
- Fl_Scrollbar - A standard scrollbar widget.
- Fl_Slider - A scrollbar with a knob.
- Fl_Value_Slider - A slider that shows the current value.

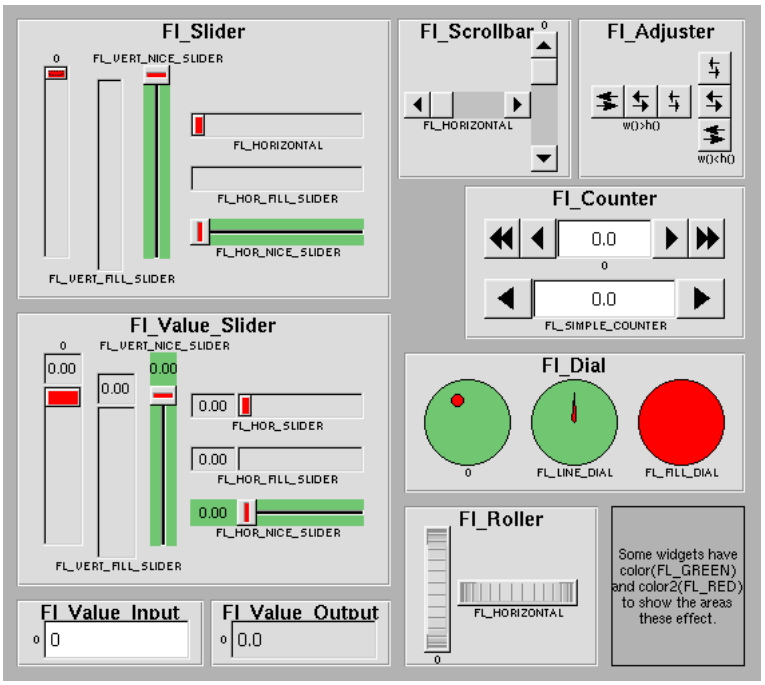


Figure 1.3 FLTK valuator widgets

The `value()` method gets and sets the current value of the widget. The `minimum()` and `maximum()` methods set the range of values that are reported by the widget.

1.4.4 Groups

The Fl_Group widget class is used as a general purpose "container" widget. Besides grouping radio buttons, the groups are used to encapsulate windows, tabs, and scrolled windows. The following group classes are available with FLTK:

- Fl_Double_Window - A double-buffered window on the screen.
- Fl_Flex - A flexible container of one row or column of widgets. Fl_Flex widgets may be nested, but see also Fl_Grid.
- Fl_Gl_Window - An OpenGL window on the screen.
- Fl_Grid - A flexible container of one or more rows and columns of widgets arranged in a "grid" with auto-layout features. Fl_Grid widgets can be nested with other Fl_Grid or many Fl_Group type widgets. Nesting with other self-resizing containers like Fl_Pack and Fl_Tile is not recommended but **may** work.
- Fl_Group - The base container class; can be used to group any widgets together.
- Fl_Pack - A collection of widgets that are packed into the group area.
- Fl_Scroll - A scrolled window area.
- Fl_Tabs - Displays child widgets as tabs.
- Fl_Tile - A tiled window area.
- Fl_Window - A window on the screen.
- Fl_Wizard - Displays one group of widgets at a time.

1.4.5 Setting the Size and Position of Widgets

The size and position of widgets is usually set when you create them. You can access them with the `x()`, `y()`, `w()`, and `h()` methods.

You can change the size and position by using the `position()`, `resize()`, and `size()` methods:

```
button->position(x, y);
group->resize(x, y, width, height);
window->size(width, height);
```

If you change a widget's size or position after it is displayed you will have to call `redraw()` on the widget's parent.

1.4.6 Colors

FLTK stores the colors of widgets as a 32-bit unsigned number that is either an index into a color palette of 256 colors (0 <= color <= 255) or a 24-bit RGB color (color > 255). The color palette is *not* the X or Windows colormap, but instead is an internal table with fixed contents.

See the Colors section of Drawing Things in FLTK for implementation details.

There are symbols for naming some of the more common colors:

- `FL_BLACK`
- `FL_RED`
- `FL_GREEN`
- `FL_YELLOW`
- `FL_BLUE`
- `FL_MAGENTA`
- `FL_CYAN`
- `FL_WHITE`

Other symbols are used as the default colors for all FLTK widgets.

- `FL_FOREGROUND_COLOR`
- `FL_BACKGROUND_COLOR`
- `FL_INACTIVE_COLOR`
- `FL_SELECTION_COLOR`

The full list of named color values can be found in FLTK Enumerations.

A color value can be created from its RGB components by using the `fl_rgb_color`() function, and decomposed again with `Fl::get_color()`:

```
Fl_Color c = fl_rgb_color(85, 170, 255);    // RGB to Fl_Color
Fl::get_color(c, r, g, b);                  // Fl_Color to RGB
```

The widget color is set using the `color()` method:

```
button->color(FL_RED);                      // set color using named value
```

Similarly, the label color is set using the `labelcolor()` method:

```
button->labelcolor(FL_WHITE);
```

The Fl_Color encoding maps to a 32-bit unsigned integer representing RGBI, so it is also possible to specify a color using a hex constant as a color map index:

```
button->color(0x000000ff);                  // colormap index #255 (FL_WHITE)
```

or specify a color using a hex constant for the RGB components:

```
button->color(0xff000000);                  // RGB: red
button->color(0x00ff0000);                  // RGB: green
button->color(0x0000ff00);                  // RGB: blue
button->color(0xffffff00);                  // RGB: white
```

Note

If TrueColor is not available, any RGB colors will be set to the nearest entry in the colormap.

1.4.7 Box Types

The type Fl_Boxtype stored and returned in Fl_Widget::box() is an enumeration defined in Enumerations.H.

These are the standard box types included with FLTK:

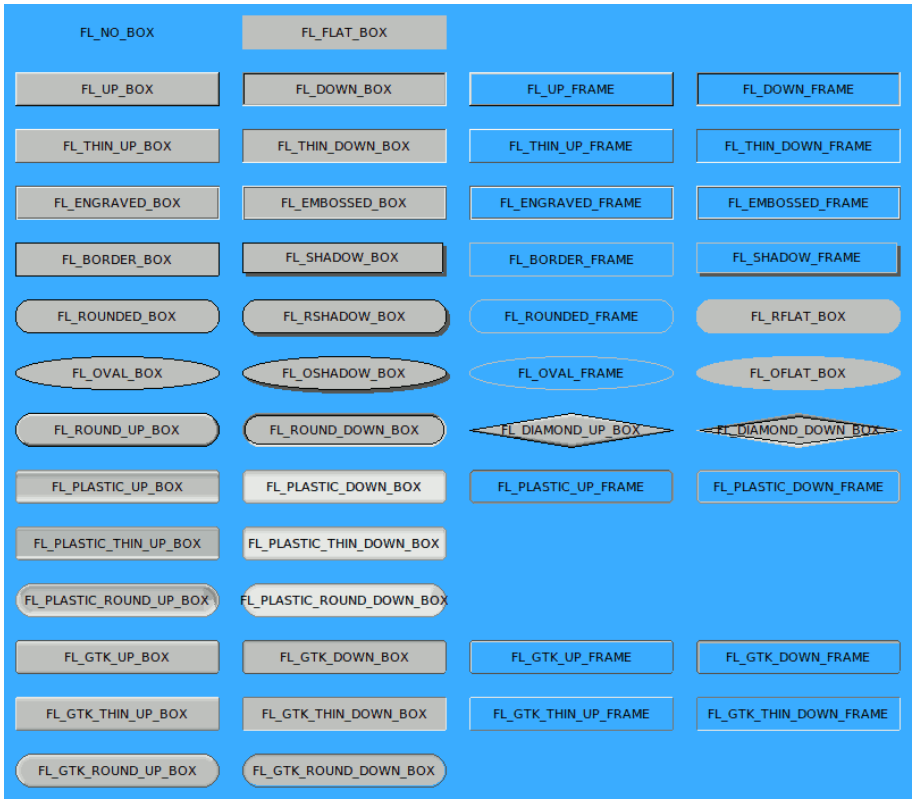


Figure 1.4 FLTK Standard Box Types

`FL_NO_BOX` means nothing is drawn at all, so whatever is already on the screen remains. The `FL_..._FRAME` types only draw their edges, leaving the interior unchanged. The blue color in the image above is the area that is not drawn by the frame types.

1.4.7.1 Making Your Own Boxtypes

You can define your own boxtypes by making a small function that draws the box and adding it to the table of boxtypes.

The Drawing Function

The drawing function is passed the bounding box and background color for the widget:

```
void xyz_draw(int x, int y, int w, int h, Fl_Color c) {
...
}
```

A simple drawing function might fill a rectangle with the given color and then draw a black outline:

```
void xyz_draw(int x, int y, int w, int h, Fl_Color c) {
  fl_color(c);
  fl_rectf(x, y, w, h);
  fl_color(FL_BLACK);
  fl_rect(x, y, w, h);
}
```

Fl_Boxtype fl_down(Fl_Boxtype b)

fl_down() returns the "pressed" or "down" version of a box. If no "down" version of a given box exists, the behavior of this function is undefined and some random box or frame is returned. See Drawing Functions for more details.

Fl_Boxtype fl_frame(Fl_Boxtype b)

fl_frame() returns the unfilled, frame-only version of a box. If no frame version of a given box exists, the behavior of this function is undefined and some random box or frame is returned. See Drawing Functions for more details.

Fl_Boxtype fl_box(Fl_Boxtype b)

fl_box() returns the filled version of a frame. If no filled version of a given frame exists, the behavior of this function is undefined and some random box or frame is returned. See Drawing Functions for more details.

Adding Your Box Type

The Fl::set_boxtype() method adds or replaces the specified box type:

```
#define XYZ_BOX FL_FREE_BOXTYPE

Fl::set_boxtype(XYZ_BOX, xyz_draw, 1, 1, 2, 2);
```

The last 4 arguments to Fl::set_boxtype() are the offsets for the `x`, `y`, `width`, and `height` values that should be subtracted when drawing the label inside the box.

A complete box design contains four box types in this order: a filled, neutral box (`UP_BOX`), a filled, depressed box (`DOWN_BOX`), and the same as outlines only (`UP_FRAME` and `DOWN_FRAME`). The function fl_down(Fl_Boxtype) expects the neutral design on a boxtype with a numerical value evenly dividable by two. fl_frame(Fl_Boxtype) expects the `UP_BOX` design at a value dividable by four.

1.4.8 Labels and Label Types

The `label()`, `align()`, `labelfont()`, `labelsize()`, `labeltype()`, `image()`, and `deimage()` methods control the labeling of widgets.

label()

The `label()` method sets the string that is displayed for the label. Symbols can be included with the label string by escaping them using the "@" symbol - "@@" displays a single at sign. These are the available symbols:

Figure 1.5 FLTK label symbols

The @ sign may also be followed by the following optional "formatting" characters, in this order:

- '#' forces square scaling, rather than distortion to the widget's shape.
- +[1-9] or -[1-9] tweaks the scaling a little bigger or smaller.
- '$' flips the symbol horizontally, '%' flips it vertically.
- [0-9] - rotates by a multiple of 45 degrees. '5' and '6' do no rotation while the others point in the direction of that key on a numeric keypad. '0', followed by four more digits rotates the symbol by that amount in degrees.

Thus, to show a very large arrow pointing downward you would use the label string "@+92->".

Symbols and text can be combined in a label, however the symbol must be at the beginning and/or at the end of the text. If the text spans multiple lines, the symbol or symbols will scale up to match the height of all the lines.

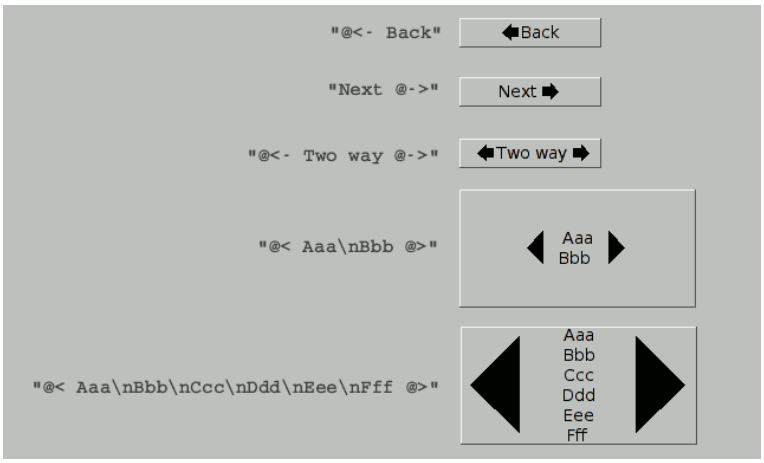


Figure 1.6 FLTK symbols and text

align()

The `align()` method positions the label. The following constants are defined and may be OR'd together as needed:

- `FL_ALIGN_CENTER` - center the label in the widget.
- `FL_ALIGN_TOP` - align the label at the top of the widget.
- `FL_ALIGN_BOTTOM` - align the label at the bottom of the widget.
- `FL_ALIGN_LEFT` - align the label to the left of the widget.
- `FL_ALIGN_RIGHT` - align the label to the right of the widget.
- `FL_ALIGN_LEFT_TOP` - The label appears to the left of the widget, aligned at the top. Outside labels only.
- `FL_ALIGN_RIGHT_TOP` - The label appears to the right of the widget, aligned at the top. Outside labels only.
- `FL_ALIGN_LEFT_BOTTOM` - The label appears to the left of the widget, aligned at the bottom. Outside labels only.
- `FL_ALIGN_RIGHT_BOTTOM` - The label appears to the right of the widget, aligned at the bottom. Outside labels only.
- `FL_ALIGN_INSIDE` - align the label inside the widget.
- `FL_ALIGN_CLIP` - clip the label to the widget's bounding box.
- `FL_ALIGN_WRAP` - wrap the label text as needed.
- `FL_ALIGN_TEXT_OVER_IMAGE` - show the label text over the image.
- `FL_ALIGN_IMAGE_OVER_TEXT` - show the label image over the text (default).
- `FL_ALIGN_IMAGE_NEXT_TO_TEXT` - The image will appear to the left of the text.
- `FL_ALIGN_TEXT_NEXT_TO_IMAGE` - The image will appear to the right of the text.
- `FL_ALIGN_IMAGE_BACKDROP` - The image will be used as a background for the widget.

labeltype()

The `labeltype()` method sets the type of the label. The following standard label types are included:

- `FL_NORMAL_LABEL` - draws the text.
- `FL_NO_LABEL` - does nothing.
- `FL_SHADOW_LABEL` - draws a drop shadow under the text.
- `FL_ENGRAVED_LABEL` - draws edges as though the text is engraved.
- `FL_EMBOSSED_LABEL` - draws edges as though the text is raised.
- `FL_ICON_LABEL` - draws the icon (Fl_Image) associated with the text.
- `FL_IMAGE_LABEL` - draws the image (Fl_Image) associated with the text.
- `FL_MULTI_LABEL` - draws multiple parts side by side, see Fl_Multi_Label.

Note

Some of these labeltypes are no longer necessary for normal widgets. Widgets allow for an image and a text side by side, depending on the widget's align() flag. FL_MULTI_LABEL was designed to be used with Fl_Menu_Item's to support icons or small images, typically left of the menu text.
As of this writing (FLTK 1.4.0) Fl_Menu_Items support only one label part (text **or** image), but using Fl_Multi_Label as the label can extend this to more than one part.

See also

class Fl_Multi_Label, Fl_Widget::align()

image() and deimage()

The `image()` and `deimage()` methods set an image that will be displayed with the widget. The `deimage()` method sets the image that is shown when the widget is inactive, while the `image()` method sets the image that is shown when the widget is active.

To make an image you use a subclass of Fl_Image.

Making Your Own Label Types

Label types are actually indexes into a table of functions that draw them. The primary purpose of this is to use this to draw the labels in ways inaccessible through the fl_font() mechanism (e.g. `FL_ENGRAVED_LABEL`) or with program-generated letters or symbology.

Label Type Functions

To setup your own label type you will need to write two functions: one to draw and one to measure the label. The draw function is called with a pointer to a Fl_Label structure containing the label information, the bounding box for the label, and the label alignment:

```
void xyz_draw(const Fl_Label *label, int x, int y, int w, int h, Fl_Align align) {
...
}
```

The label should be drawn *inside* this bounding box, even if `FL_ALIGN_INSIDE` is not enabled. The function is not called if the label value is `NULL`.

The measure function is called with a pointer to a Fl_Label structure and references to the width and height:

```
void xyz_measure(const Fl_Label *label, int &w, int &h) {
...
}
```

The function should measure the size of the label and set `w` and `h` to the size it will occupy.

Adding Your Label Type

The Fl::set_labeltype() method creates a label type using your draw and measure functions:

```
#define XYZ_LABEL FL_FREE_LABELTYPE

Fl::set_labeltype(XYZ_LABEL, xyz_draw, xyz_measure);
```

The label type number `n` can be any integer value starting at the constant `FL_FREE_LABELTYPE`. Once you have added the label type you can use the `labeltype()` method to select your label type.

The Fl::set_labeltype() method can also be used to overload an existing label type such as `FL_NORMAL_LABEL`.

Making your own symbols

It is also possible to define your own drawings and add them to the symbol list, so they can be rendered as part of any label.

To create a new symbol, you implement a drawing function `void drawit(Fl_Color c)` which typically uses the functions described in Drawing Complex Shapes to generate a vector shape inside a two-by-two units sized box around the origin. This function is then linked into the symbols table using fl_add_symbol():

```
int fl_add_symbol(const char *name, void (*drawit)(Fl_Color), int scalable)
```

`name` is the name of the symbol without the "@"; `scalable` must be set to 1 if the symbol is generated using scalable vector drawing functions.

```
int fl_draw_symbol(const char *name,int x,int y,int w,int h,Fl_Color col)
```

This function draws a named symbol fitting the given rectangle.

1.4.9 Callbacks

Callbacks are functions that are called when the value of a widget changes. A callback function is sent a Fl_Widget pointer of the widget that changed and a pointer to data that you provide:

```
void xyz_callback(Fl_Widget *w, void *data) {
...
}
```

The `callback()` method sets the callback function for a widget. You can optionally pass a pointer to some data needed for the callback:

```
int xyz_data;

button->callback(xyz_callback, &xyz_data);
```

It's also possible to use a non-capturing lambda function directly, for example:

```
button->callback( [](Fl_Widget*, void*) { puts("Hello"); } , nullptr );
```

Many programmers new to FLTK or C++ try to use a non-static class method instead of a static class method or function for their callback. Since callbacks are done outside a C++ class, the `this` pointer is not initialized for class methods.

To work around this problem, define a static method in your class that accepts a pointer to the class, and then have the static method call the class method(s) as needed. The data pointer you provide to the `callback()` method of the widget can be a pointer to the instance of your class.

```
class Foo {
  void my_callback(Fl_Widget *w);
  static void my_static_callback(Fl_Widget *w, void *f) { ((Foo *)f)->my_callback(w); }
  ...
}
...
w->callback(my_static_callback, (void *)this);
```

In an effort to make callbacks easier, more flexible, and type safe, FLTK provides three groups of macros that generate the code needed to call class methods directly with up to five custom parameters.

- `FL_FUNCTION_CALLBACK_#(WIDGET, FUNCTION, ...)` creates code for callbacks to functions and static class methods with up to five arguments. The `#` must be replaced by the number of callback arguments.
- `FL_METHOD_CALLBACK_#(WIDGET, CLASS, SELF, METH, ...)` creates code for callbacks to arbitrary public class methods
- `FL_INLINE_CALLBACK_#(WIDGET, ..., FUNCTION_BODY)` creates code for callback functions that are very close to (almost in the same line) the widget creation code, similar to lambda function in C++11. The last argument of this macro is the callback code.

The syntax is a bit unconventional, but the resulting code is flexible and needs no additional maintenance. It is also C++98 compatible. For example:

```
#include <FL/fl_callback_macros.H>
...
std::string *str = new std::string("FLTK");
Fl_Button *btn = new Fl_Button(10, 10, 100, 100);
FL_METHOD_CALLBACK_2(btn,  std::string, str, insert,  int, 2,  const char*, "...");
...
Fl_Button *inline_cb_btn_2 = new Fl_Button(390, 60, 180, 25, "2 args");
FL_INLINE_CALLBACK_2( inline_cb_btn_2,
                      const char *, text, "FLTK",   int, number, 2,
                      {
                        fl_message("We received the message %s with %d!", text, number);
                      }
                    );
```

See also

Fl_Widget::callback(Fl_Callback*, void*), FL_FUNCTION_CALLBACK_3, FL_METHOD_CALLBACK_1, FL_INLINE_CALLBACK_2

1.4.10 When and Reason

Normally callbacks are performed only when the value of the widget changes. You can change this using the Fl_Widget::when() method:

```
button->when(FL_WHEN_NEVER);
button->when(FL_WHEN_CHANGED);
button->when(FL_WHEN_RELEASE);
button->when(FL_WHEN_RELEASE_ALWAYS);
button->when(FL_WHEN_ENTER_KEY);
button->when(FL_WHEN_ENTER_KEY_ALWAYS);
button->when(FL_WHEN_CHANGED | FL_WHEN_NOT_CHANGED);
```

Within the callback, you can query why the callback was called using Fl::callback_reason(). For example, setting

```
myInput->when(FL_WHEN_RELEASE|FL_WHEN_CHANGED)
```

for a text input field may return FL_REASON_LOST_FOCUS or FL_REASON_CHANGED as a callback reason.

1.4.11 Shortcuts

Shortcuts are key sequences that activate widgets such as buttons or menu items. The `shortcut()` method sets the shortcut for a widget:

```
button->shortcut(FL_Enter);
button->shortcut(FL_SHIFT + 'b');
button->shortcut(FL_CTRL + 'b');
button->shortcut(FL_ALT + 'b');
button->shortcut(FL_CTRL + FL_ALT + 'b');
button->shortcut(0); // no shortcut
```

The shortcut value is the key event value - the ASCII value or one of the special keys described in Fl::event_key() Values combined with any modifiers like `Shift` , `Alt` , and `Control`.

1.5 Coordinates and Layout Widgets

This chapter describes the coordinate systems that apply when positioning widgets manually, and some of the basics of FLTK layout widgets that are used to position widgets automatically.

1.5.1 The Widget Coordinate System

All widgets have constructors with `x` and `y` parameters to let the programmer specify the desired initial position of the top left corner during explicit manual layout within Fl_Window and Fl_Group container widgets.

This position is always relative to the enclosing Fl_Window, which is usually, but not always, the top-level application window, or a free-floating pop-up dialog window. In some cases it could also be a subwindow embedded in a higher-level window, as shown in the figure below.

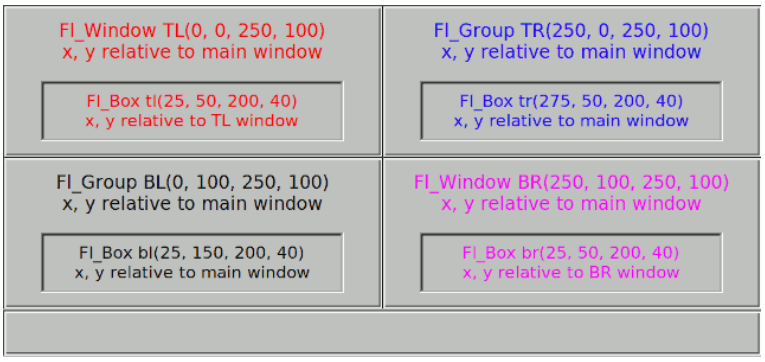


Figure 1.7 FLTK coordinate system

The positions of the TL and BR sub-windows and the TR and BL groups are all relative to the top-left corner of the main window. The positions of the boxes inside the TR and BL groups are also relative to the main window, but the boxes inside the TL and BR sub-windows are positioned relative to the enclosing sub-window.

In other words, the widget hierarchy and positions can be summarized as:

```
Fl_Window main window
  Fl_Window TL subwindow     # x, y relative to main window
    Fl_Box  tl box           # x, y relative to TL subwindow
  Fl_Window BR subwindow     # x, y relative to main window
    Fl_Box  br box           # x, y relative to BR subwindow
  Fl_Group  TR group         # x, y relative to main window
    Fl_Box  tr box           # x, y relative to main window
  Fl_Group  BL group         # x, y relative to main window
    Fl_Box  bl box           # x, y relative to main window
```

1.5.2 Layout and Container Widgets

There are four main groups of widgets derived from Fl_Group for a range of different purposes.

The first group are composite widgets that each contain a fixed set of components that work together for a specific purpose, rather than layout widgets as such, and are not discussed here.

The second group are basically containers offering the same manual layout features as Fl_Group, as described above, but which add one new capability. These widgets are Fl_Scroll, Fl_Tabs and Fl_Wizard.

The third group are layout managers that relocate and resize the child widgets added to them in order to satisfy a particular layout algorithm. These widgets are Fl_Flex, Fl_Grid, Fl_Pack, and Fl_Tile.

The final group consists of Fl_Window and its derivatives. Their special capability is that they can be top-level application windows and dialogs that interface with the operating system window manager, but can also be embedded within other windows and groups as shown in the example above. Note that the window manager may impose its own constraints on the position of top-level windows, and the `x` and `y` position parameters may be treated as hints, or even ignored. The Fl_Window class has an extra constructor that omits them.

Descriptions of layout and container widgets follow in alphabetical order.

1.5.2.1 The Fl_Flex Layout Widget

The Fl_Flex widget allows the layout of its direct children as a single row or column. If its type() is set to give the row or horizontal layout, the children are all resized to have the same height as the Fl_Flex and are moved next to each other. If set to give the column or vertical layout, the children are all resized to have the same width as the Fl_Flex and are then stacked below each other.

Widget positions (x, y) need not be given by the user because widgets are positioned inside the Fl_Flex container in the order of its children. Widget sizes can be set to (0, 0) as in Fl_Pack since they are calculated by Fl_Flex.

This is similar to Fl_Pack described below and Fl_Flex is designed to act as a drop-in replacement of Fl_Pack with some minor differences.

Other than Fl_Pack the Fl_Flex widget does **not** resize itself but resizes its children to fill the entire space of the Fl_Flex container. Single children of Fl_Flex can be set to fixed sizes to inhibit this resizing behavior. In this case the remaining space is distributed to all non-fixed widgets.

Fl_Flex widgets can be nested inside each other and with Fl_Grid in any combination.

The name Fl_Flex was inspired by the CSS 'flex' container.

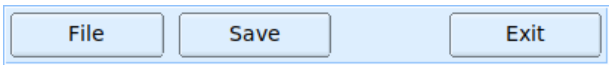


Figure 1.8 Simple Fl_Flex Layout

Fl_Flex was added in FLTK 1.4.0.

1.5.2.2 The Fl_Grid Layout Widget

Fl_Grid is the most flexible layout container in FLTK 1.4. It is based on a flexible grid of **cells** that can be assigned one widget per cell which is the *anchor* of the widget. Widgets can span multiple rows and columns and the cells can constitute a sparse matrix. Widgets can be aligned inside their cells in several ways (left, right, top, bottom) and can stretch horizontally, vertically, or both, i.e. fill the entire cell.

Widget positions (x, y) need not be given by the user because widgets are assigned to a particular grid cell by row and column number. Widget sizes can be given as their **minimal** sizes and will be resized appropriately depending on the free space.

Optional margins around all cells inside the widget border and gaps between rows and cells make the layout even more flexible.

The Fl_Grid widget should be designed with a grid (matrix) and its minimal size in mind. It is designed to **enlarge** cells and widgets in a flexible way when the Fl_Grid widget itself is created or resized.

Additional free space inside the Fl_Grid container is distributed to widgets by considering minimal row heights, column widths, sizes of widgets, and row and column *weights*. These weights are used to distribute the free space proportionally according to the row and column weights.

Fl_Grid widgets can be nested inside each other and with Fl_Flex and other subclasses of Fl_Group in any combination.

Note

We don't recommend to use Fl_Pack as child widgets although this **may** work as well.

The name Fl_Grid was inspired by the CSS 'grid' container but it has some properties in common with HTML <table> containers as well, for instance row and column spanning.

Figure 1.9 Simple Fl_Grid Layout

Fl_Grid was added in FLTK 1.4.0.

1.5.2.3 The Fl_Pack Layout Widget

The Fl_Pack widget allows the layout of its direct children as a single row, or column. If its type() is set to give the row or horizontal layout, the children are all resized to have the same height as the Fl_Pack and are moved next to each other. If set to give the column or vertical layout, the children are all resized to have the same width as the Fl_Pack and are then stacked below each other. The Fl_Pack then resizes itself to shrink-wrap itself around all of the children.

Fl_Pack widgets are often used inside an Fl_Scroll, as shown in the diagram below, to avoid having to deal with tricky resize behavior when used with nested widgets.

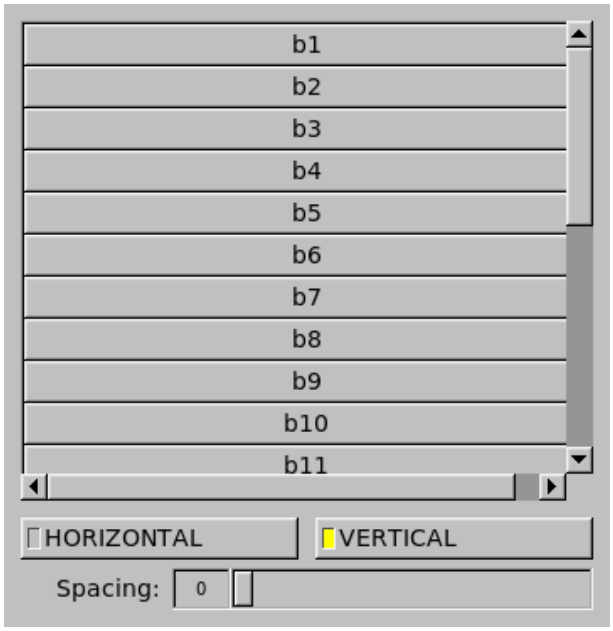


Figure 1.10 Fl_Pack test program screenshot

Since FLTK 1.4.0 Fl_Flex (described above) can in many cases be used as a drop-in replacement for Fl_Pack if this "shrink-wrap" behavior is not required. Note that the Fl_Pack layout algorithm can cause some issues because its widget size can change depending on its children and particularly because this is done late, i.e. during draw() and not as usual during resize of the window.

Note

We recommend that developers evaluate whether using Fl_Flex or Fl_Grid instead of Fl_Pack can be a better solution with more predictable and reliable resizing behavior of the overall program layout.

1.5.2.4 The Fl_Scroll Container Widget

The Fl_Scroll container widget can hold an assortment of widgets that may extend beyond its own width and height, in which case horizontal and/or vertical scrollbars may appear automatically so that you can scroll and view the entire contents.

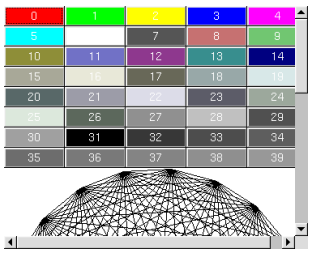


Figure 1.11 Fl_Scroll container widget

1.5.2.5 The Fl_Tabs Container Widget

The Fl_Tabs widget provides a front-to-back stack of individual panels which usually contain Fl_Group widgets and their children. The user can switch between panels by clicking on the small tabs that protrude from the panels. The appearance of each tab is determined by the child widget's label and related attributes.

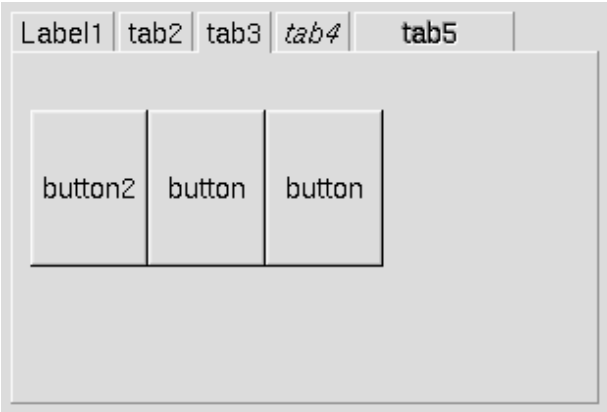


Figure 1.12 Fl_Tabs container widget

1.5.2.6 The Fl_Tile Layout Widget

The Fl_Tile widget allows the user to resize one or more of its children by dragging on the border between adjacent child widgets. However, the programmer must first explicitly layout the child widgets so that their borders exactly fill the width and height of the Fl_Tile without having any gaps between them, or at the edges. Some care is needed when initially positioning the children and setting the resizable() widget within the Fl_Tile to prevent squeezing a child to have a zero width or height. For more information see the Fl_Tile widget manual page, and How Does Resizing Work?.

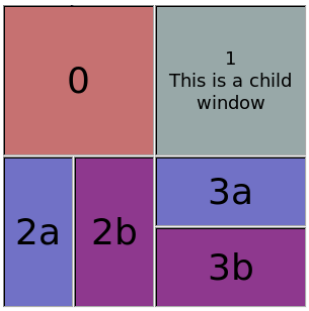


Figure 1.13 The Fl_Tile layout widget

1.5.2.7 The Fl_Wizard Container Widget

The Fl_Wizard widget derives from the Fl_Tabs class, but instead of having tabs that the user can click to select the corresponding panel, the programmer uses the prev(), next() or value() methods to show the appropriate panel. For example, the user might be able to click on "Next" and "Prev" navigation buttons or keys, as shown below.

Figure 1.14 Fl_Wizard container widget

1.6 How Does Resizing Work?

This chapter describes the basic mechanism behind the creation of resizable user interface elements in FLTK.

FLTK uses a simple, but very versatile system to resize even the most complex dialogs and interfaces. The resizing is implemented within the Fl_Group widget, and the exact resizing behavior of that group is determined by its resizable() attribute.

1.6.1 Resizing can be disabled

```
Summary:
  group = new Fl_Group(xg, yg, wg, hg, "No Resizing");
  child1 = new Fl_Box(xb, yb, wb, hb, "B"); // or other widget type
  . . .
  group->resizable((Fl_Widget*)0); // no resizing
  group->end()
```

The `resizable` may be set to the NULL pointer, which means that the group will not resize. Note that this is the default behavior for Fl_Window and Fl_Pack derived widgets, and therefore the programmer must explicitly set the window's `resizable` attribute if they want to allow the window to be resized.

1.6.2 Resizing can be simple

```
Summary:
  group = new Fl_Group(xg, yg, wg, hg, "Simple Resizing");
  child1 = new Fl_Box(xb, yb, wb, hb, "B"); // or other widget type
  . . .
  group->resizable(group); // simple proportional resizing
  group->end()
```

The `resizable` may be set to the group itself, which means that all widgets within the group will resize as the group itself is resized. This is the default behavior for Fl_Group widgets, and is shown in the diagram below.

If the group is stretched horizontally, the widths of the widgets within the group are adjusted proportionally. The same is true for vertical resizing.

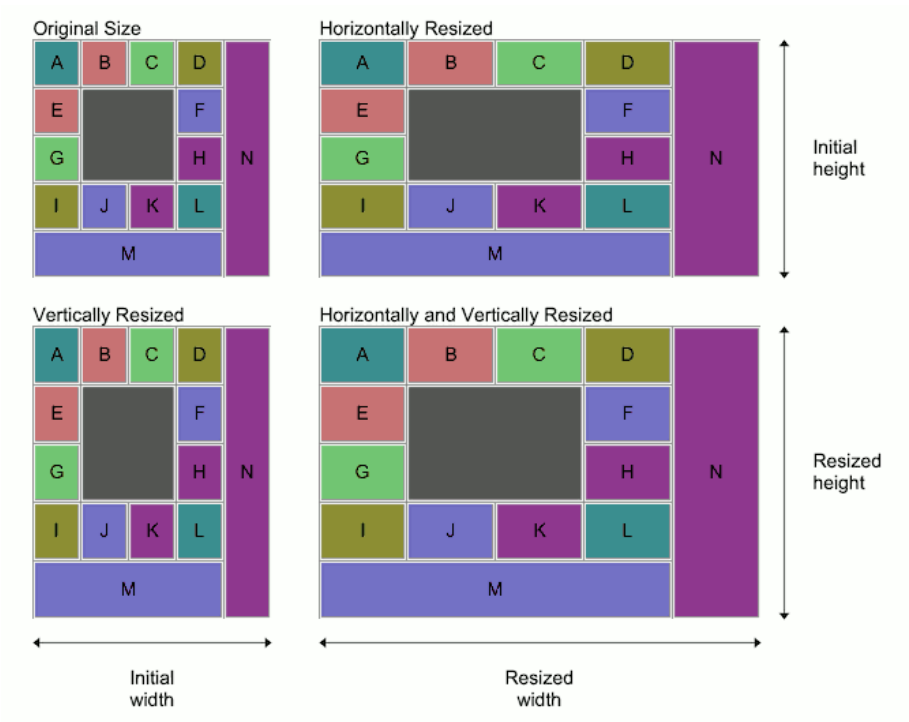


Figure 1.15 Proportional resizing example

1.6.3 Resizing can be complex

```
Summary:
  group = new Fl_Group(xg, yg, wg, hg, "Complex Resizing");
  child1 = new Fl_Box(xb, yb, wb, hb, "B"); // or other widget type
  . . .
  group->resizable(child1); // complex resizing
  group->end()
```

It is when the group's `resizable` attribute is set to one of the group's child widgets, that things become really interesting.

In the diagram below, imagine vertical lines extending from the left and right sides of the yellow widget marked "resizable", and horizontal lines extending from the top and bottom sides. Exactly which widgets resize, and by how much, is determined by which ones lie completely or partially within this cross.

The widgets marked B, C, J, K and M clearly lie completely or partially within the vertical part of the cross; the widgets marked E, F, G, H and N lie completely or partially within the horizontal part of the cross; and the widgets marked A, D, I and L do not overlap with the cross at all. The resizing behavior is as follows:

- the width and height of the `resizable` widget increase to match the change in the width and height of the group widget as it is stretched;
- the widths of those widgets that overlap with the vertical part of the cross increase proportionally as the width of the group widget increases, but their heights remain unchanged, i.e. the widgets marked B, C, J, K and M;
- the heights of those widgets that overlap with the horizontal part of the cross increase proportionally as the height of the group widget increases, but their widths remain unchanged, i.e. the widgets marked E, F, G, H and N;
- the widths and heights of the remaining widgets stay the same, i.e. the widgets marked A, D, I and L stay the same size.

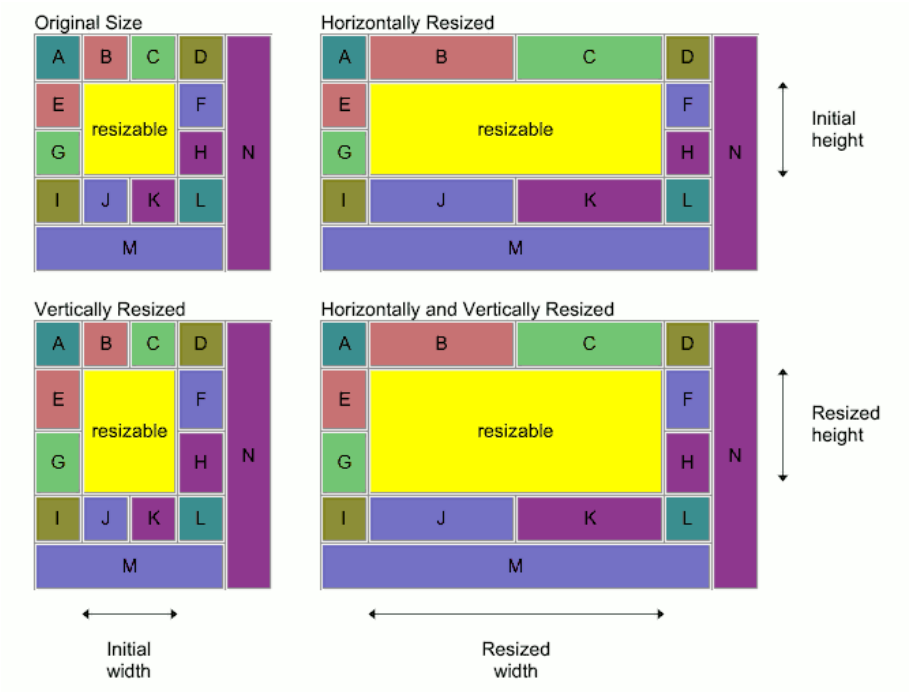


Figure 1.16 Complex resizing example

1.6.4 Practical examples

Why is this so powerful, you may ask. Well, every widget group can have a completely independent resizing strategy. By replacing one or more of the group's "normal" child widgets with another group widget where all of the above rules can be applied again, it is possible to create a hierarchy of group widgets with very complex layouts and resizing behavior.

Consider a simple dialog box, consisting of an icon box and a message area on the top and a button at the bottom right: which widget should be the `resizable` one?

Setting the `resizable` to be the icon box won't give us what we want:

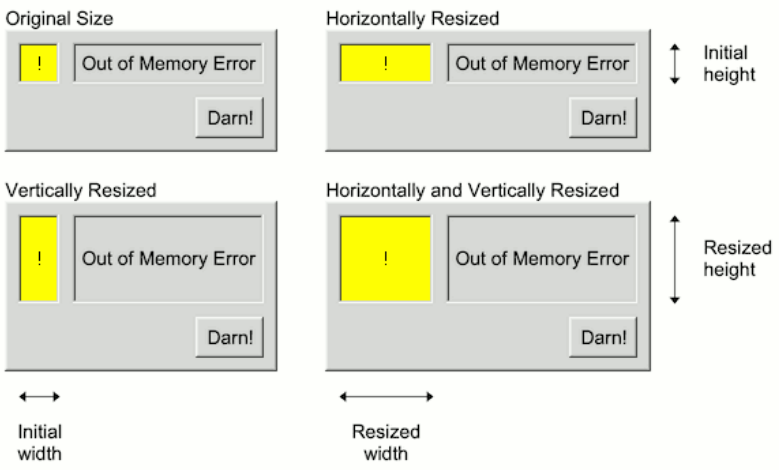


Figure 1.17 Resizing dialog example (a)

The message text area would be the logical choice so that the user can expand the dialog to see if there is more of an explanation below the short error message. This results in the behavior shown in the diagram below.

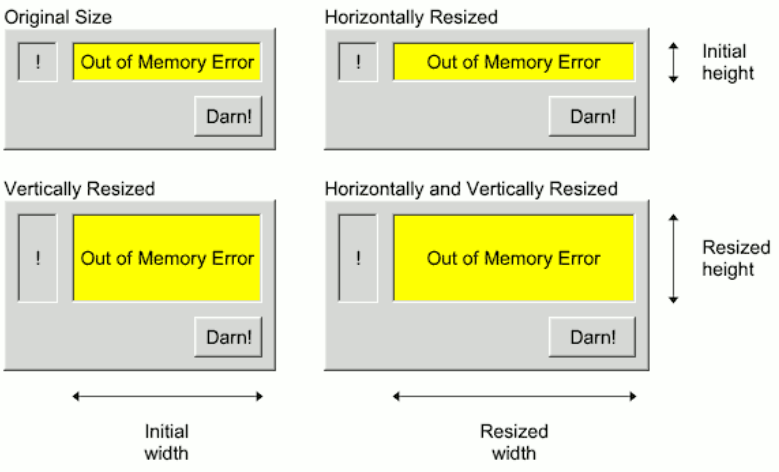


Figure 1.18 Resizing dialog example (b)

The result is close to what we want, but not quite: the text area will fully resize, the "!" icon box will resize vertically but not horizontally, which we can live with, but the "Darn!" button will - wait a minute - resize horizontally?

That's ugly. How do we stop that from happening? Simple: put it in its own group and set the `resizable` to an invisible box widget, as shown in the diagram below.

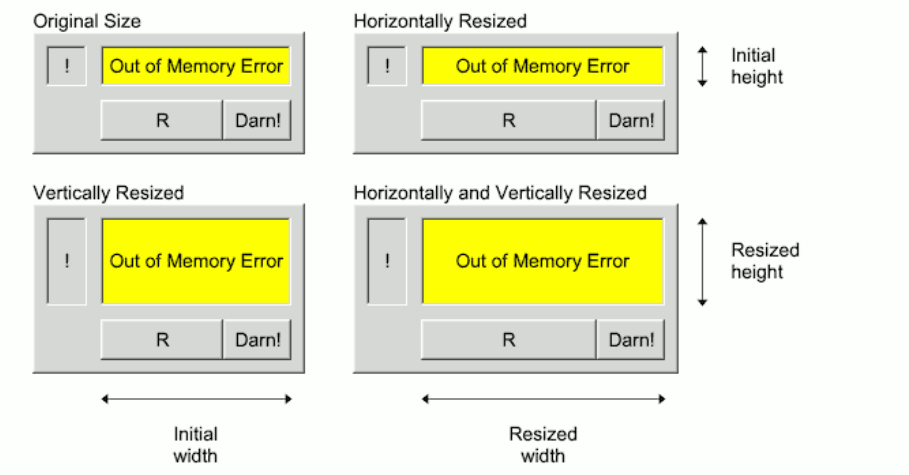


Figure 1.19 Resizing dialog example (c)

Now the invisible box, shown as "R", takes all of the horizontal resizing and the "Darn!" box will stay as it is. Here's the skeleton code:

```
dialog = new Fl_Window(300, 100);
  icon = new Fl_Box(0, 0, 50, 50, "!");
  text = new Fl_Box(50, 0, 250, 40, "Out of Memory Error");
  btns = new Fl_Group(50, 50, 250, 50); // parent group
  darn = new Fl_Button(200, 50, 100, 50, "Darn!");
     R = new Fl_Box(50, 50, 150, 50);   // "invisible" box "R"
     R->hide();                         // make sure it's invisible
  btns->resizable(R);                   // make "R" parent group resizable
  btns->end();
dialog->resizable(text);
dialog->end();
```

Imagine instead that you have a group that has a button, an input field, another button and a second input field, all next to each other, and you want the input fields to resize equally, but not the buttons. How could you achieve this?

Setting either of the input fields to be the `resizable` leaves the other one fixed, as shown below:

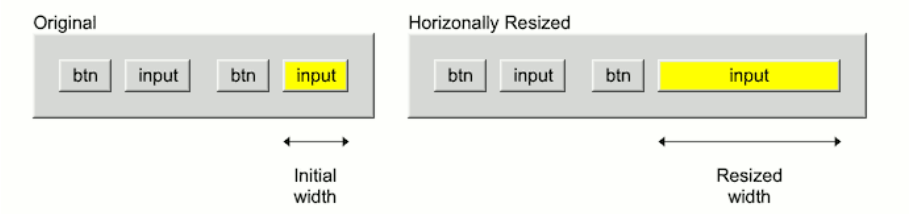


Figure 1.20 Resizing input fields example (b)

The answer is to leave the `resizable` of the group set to itself, and to create two equal size subgroups, each of which will resize equally. Add a button and input field to each subgroup, and set each subgroup's `resizable` to the input field, as shown below. Tada!

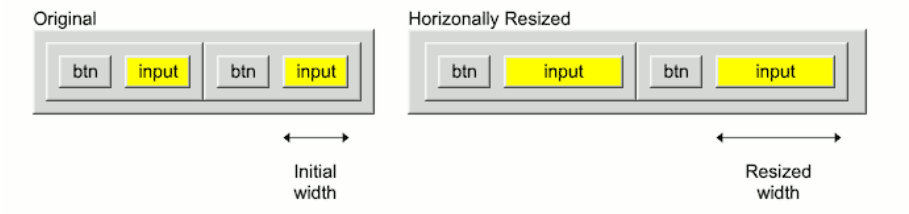


Figure 1.21 Resizing inut fields example (b)

In FLTK it is possible to solve almost any layout and resizing problem by introducing an invisible box into a group, or an extra group into the widget hierarchy. It might take some thought to achieve exactly what you want and sometimes it is necessary to introduce parallel hierarchies in order to get widgets in different groups to resize together.

Imagine you have a group containing three widgets in a row, and you want the widget in the middle to stay the same size when the group is stretched and the ones on either side and the padding between them to resize symmetrically. As described earlier, the default resizing behavior for a group results in proportional resizing of the child widgets (and also of the margins and padding between them) as shown below, which is clearly not what you want.

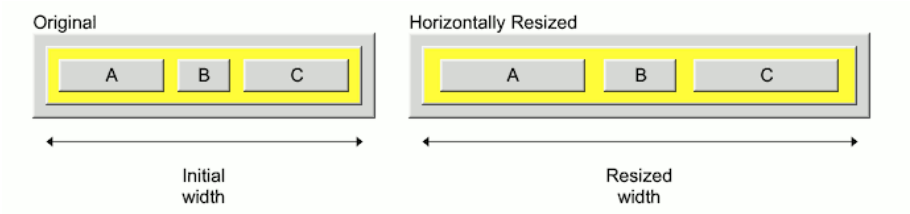


Figure 1.22 Resizing a row of widgets (a)

Simply adding a group around A and B and setting its `resizable` to A, as in the previous btn-input example, will mean that B stays the same size, but the other widgets won't resize symmetrically, so what else is needed? It isn't immediately obvious how to solve this problem, even for experienced FLTK users. This is possibly because users are generally advised to design widgets so that they don't overlap.

Albrecht Schlosser proposed an innovative technique that involves an invisible box that deliberately overlaps others to achieve the desired behavior. For the current example, this means inserting two new groups into the existing group and adding a hidden resizable widget.

The first group, shown in red below, extends from the left edge of the parent group to the middle of the gap between boxes B and C on the right. This first group contains boxes A and B, where A is the first group's `resizable` attribute.

The second group, shown in blue, extends from the right edge of the first group to the right edge of the parent group. This second group contains box C, where C is the second group's `resizable`.

The extra box widget is added to the parent group and is set as the group's `resizable`. The three `resizable` widgets are shown in yellow.

The clever bit is that this extra box widget is not horizontally aligned with any of the existing groups and widgets in the usual way, but instead overlaps the right and left parts of the two new groups by the same small amount, which means that its midpoint is aligned with the edge between the groups.

Note that, for clarity, the height of the original group has been increased to allow space for the additional annotation and to highlight the extra resizable box in the extra space at the bottom of the group. This is fine for the horizontal-only resizing shown here, but means that widgets A, B and C will never change height because the extra resizable box does not overlap them vertically. Only the padding below them will be resized.

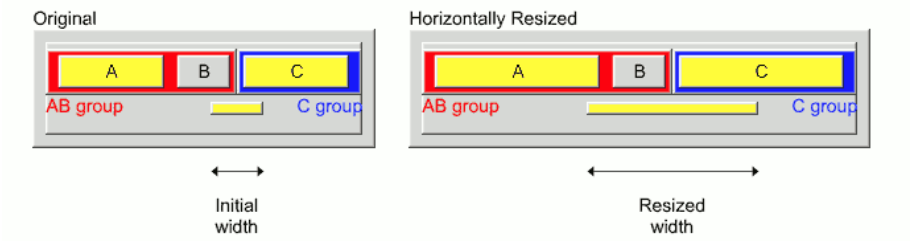


Figure 1.23 Resizing a row of widgets (b)

In a real application, you probably want to allow widgets A, B and C to resize vertically while the height of any padding or widgets above or below remains fixed, so the extra resizable box has to lie within the height of widgets A, B and C. Obviously after calling `hide()` on the box it is no longer visible, and may therefore be the same height as the other widgets, or a fraction of the height, as shown below.

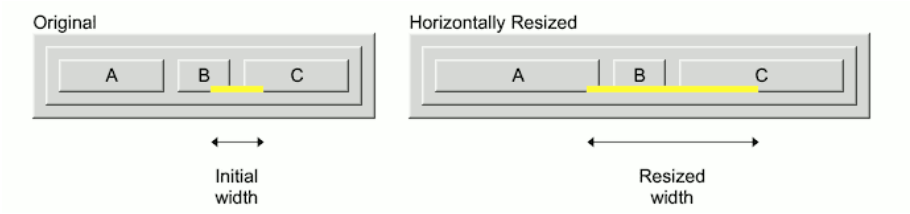


Figure 1.24 Resizing a row of widgets (c)

To summarize the key points of the new technique:

- The new resizable widget must overlap the widgets on each side by exactly the same amount.
- The width of the new resizable widget is not fixed, but should probably be a relatively small value to avoid potential problems.
- The total width of the two new groups must equal the width of the existing group and there can be no offsets or gaps between them because margins and gaps will affect the resizing behavior.
- The same principles apply to vertical resizing.

1.7 Designing a Simple Text Editor

This chapter guides you through the design of a simple FLTK-based text editor.

The complete source code for our text editor can be found in the test/editor.cxx file.

The tutorial comprises multiple chapters, and you can activate the relevant code by adjusting the TUTORIAL_↩ CHAPTER macro at the top of the source file to match the chapter number.

Each chapter builds on the previous one. The documentation, as well as the source code, can be read sequentially, maintaining a consistent program structure while introducing additional features step by step.

Note

The tutorial uses several global variables for brevity. Additionally, the order of code blocks is rather uncommon but helps to keep related features within a chapter.

1.7.1 Determining the Goals of the Text Editor

As our first step, we define what we want our text editor to do:

1. Edit a single text document.
2. Provide a menubar/menus for all functions.
3. Load from a file.
4. Save to a file.
5. Keep track of when the file has been changed.
6. Cut/copy/delete/paste menus.
7. Search and replace functionality.
8. Multiple views of the same text.
9. "C" language syntax highlighting.

1.7.2 Chapter 1: A Minimal App

Let's ensure that we can set up our build process to compile and verify our code as we add features. We begin by writing a minimal program with no other purpose than opening a window.

The code for that is barely longer than a "Hello, world" program and is marked in the source code as `TUTORIAL↩_CHAPTER = 1`.

```
#include <FL/Fl_Double_Window.H>
#include <FL/Fl.H>

Fl_Double_Window *app_window = NULL;

void tut1_build_app_window() {
  app_window = new Fl_Double_Window(640, 480, "FLTK Editor");
  app_window->xclass("fl_editor");
}

int main (int argc, char **argv) {
  tut1_build_app_window();
  app_window->show(argc, argv);
  return Fl::run();
}
```

Note: setting the xclass() of the window is not required but it is considered good style. In this case it works around a known side effect under **Windows** where leaving the default ("editor.exe") would prevent entering emojis with the emoji palette. You can ignore this for now in this tutorial.

Passing `argc` and `argv` to `Fl_Double_Window::show()` allows FLTK to parse command line options, providing the user with the ability to change the color or graphical scheme of the editor at launch time.

`Fl::run()` will return when no more windows in the app are visible. In other words, if all windows in an app are closed, hidden, or deleted. Pressing "Escape" or clicking the "Close" button in the window frame will close our only window, prompting `Fl::run()` to return, effectively ending the app.

When building FLTK from source, the `CMake` environment includes the necessary rules to build the editor. You can find more information on how to write your own `CMake` files in the `README.CMake.txt` text in the top FLTK directory.

For Linux and macOS, FLTK comes with the `fltk-config` script that generates the compiler commands for you:
`fltk-config --compile editor.cxx`

If the code compiles and links correctly, running the app will pop up an empty application window on the desktop screen. You can close the window and quit the app by pressing the 'Escape' key or by clicking the "Close" button in the window frame.

Congratulations, you've just built a minimal FLTK app.

1.7.3 Chapter 2: Adding a Menu Bar

In this chapter, we will handle the window title and add the main menu bar with a File menu and a Quit button.

We need to declare a variable to track track changes in the text, and a buffer for the current filename.

```
// remove `main()` from chapter 1, but keep the rest of the code, then add...

#include <FL/Fl_Menu_Bar.H>
#include <FL/fl_ask.H>
#include <FL/filename.H>
#include <FL/fl_string_functions.h>

Fl_Menu_Bar *app_menu_bar = NULL;
bool text_changed = false;
char app_filename[FL_PATH_MAX] = "";
```

The window title is either "FLTK Editor" if the text is not saved in any file, or the filename, followed by an * if the text changed. Note that we have two ways to set the label of a widget. `label()` will link a static text, and `copy_label()` which will copy and manage the label text.

```
void update_title() {
  const char *fname = NULL;
  if (app_filename[0])
    fname = fl_filename_name(app_filename);
  if (fname) {
    char buf[FL_PATH_MAX + 3];
    buf[FL_PATH_MAX + 2] = '\0';   // ensure that the buffer is always terminated
    if (text_changed) {
      snprintf(buf, FL_PATH_MAX+2, "%s *", fname);
    } else {
      snprintf(buf, FL_PATH_MAX+2, "%s", fname);
    }
    app_window->copy_label(buf);
  } else {
    app_window->label("FLTK Editor");
  }
}
```

Now instead of writing directly to `text_changed`, we write a function that can set and clear the flag, and update the title accordingly.

```
void set_changed(bool v) {
  if (v != text_changed) {
    text_changed = v;
    update_title();
  }
}
```

Let's do the same for changing the filename. If the new filename is NULL, the window title will revert to "FLTK Editor".

```
void set_filename(const char *new_filename) {
  if (new_filename) {
    fl_strlcpy(app_filename, new_filename, FL_PATH_MAX);
  } else {
    app_filename[0] = 0;
  }
  update_title();
}
```

But enough of managing window titles. The following code will add the first widget to our window. A menubar is created at the top and all across the main window.

```
void menu_quit_callback(Fl_Widget *, void *) { /* TODO */ }

void tut2_build_app_menu_bar() {
  app_window->begin();
  app_menu_bar = new Fl_Menu_Bar(0, 0, app_window->w(), 25);
  app_menu_bar->add("File/Quit Editor", FL_COMMAND+'q', menu_quit_callback);
  app_window->callback(menu_quit_callback);
  app_window->end();
}

int main (int argc, char **argv) {
  tut1_build_app_window();
  tut2_build_app_menu_bar();
  app_window->show(argc, argv);
  return Fl::run();
}
```

`begin()` tells FLTK to add all widgets created hereafter to our `app_window`. In this particular case, it is redundant because creating the window in the previous chapter already called `begin()` for us.

In the next line, we create the menu bar and add our first menu item to it. Menus can be constructed like file paths, with forward slashes '/' separating submenus from menu items.

Our basic callback is simple:

```
void menu_quit_callback(Fl_Widget *, void *) {
  Fl::hide_all_windows();
}
```

`Fl::hide_all_windows()` will make all windows invisible, causing `Fl::run()` to return and `main` to exit.

The next line, `app_window->callback(menu_quit_callback)` links the same `menu_quit_callback` to the `app_window` as well. Assigning the window callback removes the default "Escape" key handling and allows the `menu_quit_callback` to handle that keypress with a friendly dialog box instead of just quitting the app.

The `Fl_Widget*` parameter in the callback will either be `app_window` if called through the window callback, or `app_menu_bar` if called by one of the menu items.

One of our goals was to keep track of text changes. If we know the text changed and is unsaved, we should notify the user that she is about to lose her work. We achieve this by adding a dialog box in the Quit callback that queries if the user really wants to quit, even if text was changed:

```
void menu_quit_callback(Fl_Widget *, void *) {
  if (text_changed) {
    int c = fl_choice("Changes in your text have not been saved.\n"
                      "Do you want to quit the editor anyway?",
                      "Quit", "Cancel", NULL);
    if (c == 1) return;
  }
  Fl::hide_all_windows();
}
```

1.7.4 Chapter 3: Adding a Text Editor widget

FLTK comes with a pretty capable builtin text editing widget. We will use this `Fl_Text_Editor` widget here to allow users to edit their documents.

`Fl_Text_Editor` needs an `Fl_Text_Buffer` to do anything useful. What might seem like an unnecessary extra step is a great feature: we can assign one text buffer to multiple text editors. In a later chapter, we will use this feature to implement a split editor window.

```
#include <FL/Fl_Text_Buffer.H>
#include <FL/Fl_Text_Editor.H>

Fl_Text_Editor *app_editor = NULL;
Fl_Text_Editor *app_split_editor = NULL; // for later
Fl_Text_Buffer *app_text_buffer = NULL;

// ... callbacks go here

void tut3_build_main_editor() {
  app_window->begin();
  app_text_buffer = new Fl_Text_Buffer();
  app_text_buffer->add_modify_callback(text_changed_callback, NULL);
  app_editor = new Fl_Text_Editor(0, app_menu_bar->h(),
    app_window->w(), app_window->h() - app_menu_bar->h());
  app_editor->buffer(app_text_buffer);
  app_editor->textfont(FL_COURIER);
  app_window->resizable(app_editor);
  app_window->end();
}
```

By setting the `app_editor` to be the `resizable()` property of `app_window`, we make our application window resizable on the desktop, and we ensure that resizing the window will only resize the text editor vertically, but not our menu bar.

To keep track of changes to the document, we add a callback to the text editor that will be called whenever text is added or deleted. The text modify callback sets our `text_changed` flag if text was changed:

```
// insert before tut3_build_main_editor()
void text_changed_callback(int, int n_inserted, int n_deleted, int, const char*, void*) {
```

```
  if (n_inserted || n_deleted)
    set_changed(true);
}
```

To wrap this chapter up, we add a "File/New" menu and link it to a callback that clears the text buffer, clears the current filename, and marks the buffer as unchanged.

```
// insert before tut3_build_main_editor()
void menu_new_callback(Fl_Widget*, void*) {
  app_text_buffer->text("");
  set_changed(false);
}

// insert at the end of tut3_build_main_editor()
  ...
  // find the Quit menu and insert the New menu there
  int ix = app_menu_bar->find_index(menu_quit_callback);
  app_menu_bar->insert(ix, "New", FL_COMMAND+'n', menu_new_callback);
  ...
```

1.7.5 Chapter 4: Reading and Writing Files

In this chapter, we will add support for loading and saving text files, so we need three more menu items in the File menu: Open, Save, and Save As.

```
#include <FL/Fl_Native_File_Chooser.H>
#include <FL/platform.H>
#include <errno.h>

// ... add callbacks here

void tut4_add_file_support() {
  int ix = app_menu_bar->find_index(menu_quit_callback);
  app_menu_bar->insert(ix, "Open", FL_COMMAND+'o', menu_open_callback, NULL, FL_MENU_DIVIDER);
  app_menu_bar->insert(ix+1, "Save", FL_COMMAND+'s', menu_save_callback);
  app_menu_bar->insert(ix+2, "Save as...", FL_COMMAND+'S', menu_save_as_callback, NULL, FL_MENU_DIVIDER);
}
```

Note

> The menu shortcuts `FL_COMMAND+'s'` and `FL_COMMAND+'S'` look the same at a first glance, but the second shortcut is actually `Ctrl-Shift-S` due to the capital letter 'S'. Also, we use `FL_COMMAND` as our menu shortcut modifier key. `FL_COMMAND` translates to `FL_CTRL` on Windows and Linux, and to `FL_META` on macOS, better known as the cloverleaf, or simply "the Apple key".

We implement the Save As callback first, because we will want to call it from the Open callback later. The basic callback is only a few lines of code.

```
void menu_save_as_callback(Fl_Widget*, void*) {
  Fl_Native_File_Chooser file_chooser;
  file_chooser.title("Save File As...");
  file_chooser.type(Fl_Native_File_Chooser::BROWSE_SAVE_FILE);
  if (file_chooser.show() == 0) {
    app_text_buffer->savefile(file_chooser.filename());
    set_filename(file_chooser.filename());
    set_changed(false);
  }
}
```

However if the user has already set a file name including path information, it is the polite thing to preload the file chooser with that information. This little chunk of code will separate the file name from the path before we call `file_chooser.show()`:

```
// insert before `if (file_chooser.show()...`
  if (app_filename[0]) {
    char temp_filename[FL_PATH_MAX];
    fl_strlcpy(temp_filename, app_filename, FL_PATH_MAX);
    const char *name = fl_filename_name(temp_filename);
    if (name) {
      file_chooser.preset_file(name);
      temp_filename[name - temp_filename] = 0;
      file_chooser.directory(temp_filename);
    }
  }
```

Great. Now let's add code for our File/Save menu. If no filename was set yet, it falls back to our Save As callback. `Fl_Text_Editor::savefile()` writes the contents of our text widget into a UTF-8 encoded text file.

```
void menu_save_callback(Fl_Widget*, void*) {
  if (!app_filename[0]) {
    menu_save_as_callback(NULL, NULL);
  } else {
    app_text_buffer->savefile(file_chooser.filename());
    set_changed(false);
  }
}
```

Now that we have a save method available, we can improve the `menu_quit_callback` and offer the option to save the current modified text before quitting the app. Here is the new quit callback code that replaces the old callback:

```
void menu_quit_callback(Fl_Widget *, void *) {
  if (text_changed) {
      int r = fl_choice("The current file has not been saved.\n"
                        "Would you like to save it now?",
                        "Cancel", "Save", "Don't Save");
      if (r == 0)   // cancel
        return;
      if (r == 1) { // save
        menu_save_callback(NULL, NULL);
        return;
      }
  }
  Fl::hide_all_windows();
}
```

On to loading a new file. Let's write the function to load a file from a given file name:

```
void load(const char *filename) {
  if (app_text_buffer->loadfile(filename) == 0) {
    set_filename(filename);
    set_changed(false);
  }
}
```

A friendly app should warn the user if file operations fail. This can be done in three lines of code, so let's add an alert dialog after every `loadfile` and `savefile` call. This is exemplary for `load()`, and the code is very similar for the two other locations.

```
void load(const char *filename) {
  if (app_text_buffer->loadfile(filename) == 0) {
    set_filename(filename);
    set_changed(false);
  } else {
    fl_alert("Failed to load file\n%s\n%s",
             filename,
             strerror(errno));
  }
}
```

If the user selects our pulldown "Load" menu, we first check if the current text was modified and provide a dialog box that offers to save the changes before loading a new text file:

```
void menu_open_callback(Fl_Widget*, void*) {
  if (text_changed) {
    int r = fl_choice("The current file has not been saved.\n"
                      "Would you like to save it now?",
                      "Cancel", "Save", "Don't Save");
    if (r == 2)
      return;
    if (r == 1)
      menu_save_callback();
  }
  ...
```

If the user did not cancel the operation, we pop up a file chooser for loading the file, using similar code as in Save As.

```
...
  Fl_Native_File_Chooser file_chooser;
  file_chooser.title("Open File...");
  file_chooser.type(Fl_Native_File_Chooser::BROWSE_FILE);
...
```

Again, we preload the file chooser with the last used path and file name:

```
...
  if (app_filename[0]) {
    char temp_filename[FL_PATH_MAX];
    fl_strlcpy(temp_filename, app_filename, FL_PATH_MAX);
```

```
    const char *name = fl_filename_name(temp_filename);
    if (name) {
      file_chooser.preset_file(name);
      temp_filename[name - temp_filename] = 0;
      file_chooser.directory(temp_filename);
    }
  }
...
```

And finally, we pop up the file chooser. If the user cancels the file dialog, we do nothing and keep the current file. Otherwise, we call the `load()` function that we already wrote:

```
  if (file_chooser.show() == 0)
    load(file_chooser.filename());
}
```

We really should support two more ways to load documents from a file. Let's modify the "show and run" part of `main()` to handle command line parameters and desktop drag'n'drop operations. For that, we refactor the last two lines of `main()` into a new function:

```
// ... new function here

int main (int argc, char **argv) {
  tut1_build_app_window();
  tut2_build_app_menu_bar();
  tut3_build_main_editor();
  tut4_add_file_support();
  // ... refactor those into the new function
  // app_window->show(argc, argv);
  // return Fl::run();
  return tut4_handle_commandline_and_run(argc, argv);
}
```

Our function to show the window and run the app has a few lines of boilerplate code. `Fl::args_to_utf8()` converts the command line argument from whatever the host system provides into Unicode. `Fl::args()` goes through the list of arguments and gives `args_handler()` a chance to handle each argument. It also makes sure that FLTK specific args are still forwarded to FLTK, so `"-scheme plastic"` and `"-background #aaccff"` will draw beautiful blue buttons in a plastic look.

`fl_open_callback()` lets FLTK know what to do if a user drops a text file onto our editor icon (Apple macOS). Here, we ask it to call the `load()` function that we wrote earlier.

```
// ... args_handler here

int tut4_handle_commandline_and_run(int &argc, char **argv) {
  int i = 0;
  Fl::args_to_utf8(argc, argv);
  Fl::args(argc, argv, i, args_handler);
  fl_open_callback(load);
  app_window->show(argc, argv);
  return Fl::run();
}
```

Last work item for this long chapter: what should our `args_handler` do? We could handle additional command line options here, but for now, all we want to handle is file names and paths. Let's make this easy: if the current arg does not start with a '-', we assume it is a file name, and we call `load()`:

```
int args_handler(int argc, char **argv, int &i) {
  if (argv && argv[i] && argv[i][0]!='-') {
    load(argv[i]);
    i++;
    return 1;
  }
  return 0;
}
```

So this is our basic but quite functional text editor app in about 100 lines of code. The following chapters add some user convenience functions and show off some FLTK features including split editors and syntax highlighting.

1.7.6 Chapter 5: Cut, Copy, and Paste

The FLTK Text Editor widget comes with builtin cut, copy, and paste functionality, but as a courtesy, we should also offer these as menu items in the main menu.

In our feature list, we noted that we want to implement a split text editor. This requires that the callbacks know which text editor has the keyboard focus. Calling `Fl::focus()` may return `NULL` or other unknown widgets, so we add a little test in our callbacks:

```
void menu_cut_callback(Fl_Widget*, void* v) {
  Fl_Widget *e = Fl::focus();
  if (e && (e == app_editor || e == app_split_editor))
    Fl_Text_Editor::kf_cut(0, (Fl_Text_Editor*)e);
}
```

We can write very similar callbacks for undo, redo, copy, paste, and delete. Adding a new menu and the six menu items follows the same pattern as before. Using the Menu/Item notation will create an Edit menu for us:

```
void tut5_cut_copy_paste() {
  app_menu_bar->add("Edit/Undo",   FL_COMMAND+'z', menu_undo_callback);
  app_menu_bar->add("Edit/Redo",   FL_COMMAND+'Z', menu_redo_callback, NULL, FL_MENU_DIVIDER);
  app_menu_bar->add("Edit/Cut",    FL_COMMAND+'x', menu_cut_callback);
  app_menu_bar->add("Edit/Copy",   FL_COMMAND+'c', menu_copy_callback);
  app_menu_bar->add("Edit/Paste",  FL_COMMAND+'v', menu_paste_callback);
  app_menu_bar->add("Edit/Delete", 0,              menu_delete_callback);
}
```

1.7.7 Chapter 6: Find and Find Next

Corporate called. They want a dialog box for their users that can search for some word in the text file. We can add this functionality using a callback and a standard FLTK dialog box.

Here is some code to find a string in a text editor. The first four lines make sure that we start our search at the cursor position of the current editor window. The rest of the code searches the string and marks it if found.

```
void find_next(const char *needle) {
  Fl_Text_Editor *editor = app_editor;
  Fl_Widget *e = Fl::focus();
  if (e && e == app_split_editor)
    editor = app_split_editor;
  int pos = editor->insert_position();
  int found = app_text_buffer->search_forward(pos, needle, &pos);
  if (found) {
    app_text_buffer->select(pos, pos + (int)strlen(needle));
    editor->insert_position(pos + (int)strlen(needle));
    editor->show_insert_position();
  } else {
    fl_alert("No further occurrences of '%s' found!", needle);
  }
}
```

The callbacks are short, using the FLTK text field dialog box and the `find_next` function that we already implemented. The last searched text is saved in `last_find_text` to be reused by `menu_find_next_callback`. If no search text was set yet, or it was set to an empty text, "Find Next" will forward to `menu_find_callback` and pop up our "Find Text" dialog.

```
char last_find_text[1024] = "";

void menu_find_callback(Fl_Widget*, void* v) {
  const char *find_text = fl_input("Find in text:", last_find_text);
  if (find_text) {
    fl_strlcpy(last_find_text, find_text, sizeof(last_find_text));
    find_next(find_text);
  }
}

void menu_find_next_callback(Fl_Widget*, void* v) {
  if (last_find_text[0]) {
    find_next(last_find_text);
  } else {
    menu_find_callback(NULL, NULL);
  }
}
```

And of course we need to add two menu items to our main application menu.

```
...
app_menu_bar->add("Find/Find...",   FL_COMMAND+'f', menu_find_callback);
app_menu_bar->add("Find/Find Next", FL_COMMAND+'g', menu_find_next_callback, NULL, FL_MENU_DIVIDER);
...
```

1.7.8 Chapter 7: Replace and Replace Next

To implement the next feature, we will need to implement our own "Find and Replace" dialog box. To make this dialog box useful, it needs the following elements:

- a text input field for the text that we want to find
- a text input field for the replacement text
- a button to find the next occurrence
- a button to replace the current text and find the next occurrence
- a button to close the dialog

This is rather complex functionality, so instead of adding more global variables, we will pack this dialog into a class, derived from `Fl_Window`.

Note

The tutorial uses `Fl_Double_Window` instead of `Fl_Window` throughout. Historically, on some platforms, `Fl_Window` renders faster, but has a tendency to flicker. In today's world, this has very little relevance and FLTK optimizes both window types. `Fl_Double_Window` is recommended unless there is a specific reason to use `Fl_Window`.

Let's implement the text replacement code first:

```
char last_replace_text[1024] = "";

void replace_selection(const char *new_text) {
  Fl_Text_Editor *editor = app_editor;
  Fl_Widget *e = Fl::focus();
  if (e && e == app_split_editor)
    editor = app_split_editor;
  int start, end;
  if (app_text_buffer->selection_position(&start, &end)) {
    app_text_buffer->remove_selection();
    app_text_buffer->insert(start, new_text);
    app_text_buffer->select(start, start + (int)strlen(new_text));
    editor->insert_position(start + (int)strlen(new_text));
    editor->show_insert_position();
  }
}
```

As before, the first four lines anticipate a split editor and find the editor that has focus. The code then deletes the currently selected text, replaces it with the new text, selects the new text, and finally sets the text cursor to the end of the new text.

The Replace_Dialog class

The Replace_Dialog class holds pointers to our active UI elements as well as all the callbacks for the dialog buttons.

```
class Replace_Dialog : public Fl_Double_Window {
  Fl_Input *find_text_input;
  Fl_Input *replace_text_input;
  Fl_Button *find_next_button;
  Fl_Button *replace_and_find_button;
  Fl_Button *close_button;
public:
  Replace_Dialog(const char *label);
  void show() FL_OVERRIDE;
private:
  static void find_next_callback(Fl_Widget*, void*);
  static void replace_and_find_callback(Fl_Widget*, void*);
  static void close_callback(Fl_Widget*, void*);
};

Replace_Dialog *replace_dialog = NULL;
```

The constructor creates the dialog and marks it as "non modal". This will make the dialog hover over the application window like a toolbox window until the user closes it, allowing multiple "find and replace" operations. So here is our constructor:

```
Replace_Dialog::Replace_Dialog(const char *label)
: Fl_Double_Window(430, 110, label)
{
  find_text_input = new Fl_Input(100, 10, 320, 25, "Find:");
  replace_text_input = new Fl_Input(100, 40, 320, 25, "Replace:");
  Fl_Flex* button_field = new Fl_Flex(100, 70, w()-100, 40);
  button_field->type(Fl_Flex::HORIZONTAL);
  button_field->margin(0, 5, 10, 10);
  button_field->gap(10);
  find_next_button = new Fl_Button(0, 0, 0, 0, "Next");
  find_next_button->callback(find_next_callback, this);
  replace_and_find_button = new Fl_Button(0, 0, 0, 0, "Replace");
  replace_and_find_button->callback(replace_and_find_callback, this);
  close_button = new Fl_Button(0, 0, 0, 0, "Close");
  close_button->callback(close_callback, this);
  button_field->end();
  set_non_modal();
}
```

All buttons are created inside an `Fl_Flex` group. They will be arranged automatically by `Fl_Flex`, so there is no need to set x and y coordinates or a width or height. `button_field` will lay out the buttons for us.

Note

> There is no need to write a destructor or delete individual widgets. When we delete an instance of `Replace↩_Dialog`, all children are deleted for us.

The `show()` method overrides the window's show method. It adds some code to preload the values of the text fields for added convenience. It then pops up the dialog box by calling the original `Fl_Double_Window::show()`.

```
void Replace_Dialog::show() {
  find_text_input->value(last_find_text);
  replace_text_input->value(last_replace_text);
  Fl_Double_Window::show();
}
```

The buttons in the dialog need callbacks to be useful. If callbacks are defined within a class, they must be defined `static`, but a pointer to the class can be provided through the `user_data` field. We have done that in the constructor by adding `this` as the last argument when setting the callback, for example in `close_↩button->callback(close_callback, this);`.

The callback itself can then extract the `this` pointer with a static cast:

```
void Replace_Dialog::close_callback(Fl_Widget*, void* my_dialog) {
  Replace_Dialog *dlg = static_cast<Replace_Dialog*>(my_dialog);
  dlg->hide();
}
```

The callback for the Find button uses our already implemented `find_next` function:

```
void Replace_Dialog::find_next_callback(Fl_Widget*, void* my_dialog) {
  Replace_Dialog *dlg = static_cast<Replace_Dialog*>(my_dialog);
  fl_strlcpy(last_find_text, dlg->find_text_input->value(), sizeof(last_find_text));
  fl_strlcpy(last_replace_text, dlg->replace_text_input->value(), sizeof(last_replace_text));
  if (last_find_text[0])
    find_next(last_find_text);
}
```

The Replace button callback calls our newly implemented `replace_selection` function and then continues on to the `find_next_callback`:

```
void Replace_Dialog::replace_and_find_callback(Fl_Widget*, void* my_dialog) {
  Replace_Dialog *dlg = static_cast<Replace_Dialog*>(my_dialog);
  replace_selection(dlg->replace_text_input->value());
  find_next_callback(NULL, my_dialog);
}
```

This long chapter comes close to its end. We are missing menu items that pop up our dialog and that allow a quick "Replace and Find Next" functionality without popping up the dialog. The code is quite similar to the "Find" and "Find Next" code in the previous chapter:

```
void menu_replace_callback(Fl_Widget*, void*) {
  if (!replace_dialog)
    replace_dialog = new Replace_Dialog("Find and Replace");
```

```
  replace_dialog->show();
}

void menu_replace_next_callback(Fl_Widget*, void*) {
  if (!last_find_text[0]) {
    menu_replace_callback(NULL, NULL);
  } else {
    replace_selection(last_replace_text);
    find_next(last_find_text);
  }
}

void tut7_implement_replace() {
  app_menu_bar->add("Find/Replace...",   FL_COMMAND+'r', menu_replace_callback);
  app_menu_bar->add("Find/Replace Next", FL_COMMAND+'t', menu_replace_next_callback);
}
```

1.7.9 Chapter 8: Editor Features

Chapter 7 was long an intense. Let's relax and implement something simple here. We want menus with check boxes that can toggle some text editor features on and off:

```
void tut8_editor_features() {
  app_menu_bar->add("Window/Line Numbers", FL_COMMAND+'l', menu_linenumbers_callback, NULL, FL_MENU_TOGGLE);
  app_menu_bar->add("Window/Word Wrap", 0, menu_wordwrap_callback, NULL, FL_MENU_TOGGLE);
}
```

The `Fl_Widget` parameter in callbacks always points to the widget that causes the callback. Menu items are not derived from widgets, so to find out which menu item caused a callback, we can do this:

```
void menu_linenumbers_callback(Fl_Widget* w, void*) {
  Fl_Menu_Bar* menu = static_cast<Fl_Menu_Bar*>(w);
  const Fl_Menu_Item* linenumber_item = menu->mvalue();
  if (linenumber_item->value()) {
    app_editor->linenumber_width(40);
  } else {
    app_editor->linenumber_width(0);
  }
  app_editor->redraw();
}
```

Setting the width enables the line numbers, setting it to 0 disables the line number display. When changing the value of a widget, FLTK will make sure that the widget is redrawn to reflect the new value. When changing other attributes such as colors or fonts, FLTK assumes that many attributes are changed at the same time and leaves it to the user to call `Fl_Widget::redraw()` when done. Here we call `app_editor->redraw()` to make sure that the change in the line number setting is also drawn on screen.

Let's not forget to update the line number display for a potential split editor widget es well:

```
// add before the end of menu_linenumbers_callback
if (app_split_editor) {
  if (linenumber_item->value()) {
    app_split_editor->linenumber_width(40);
  } else {
    app_split_editor->linenumber_width(0);
  }
  app_split_editor->redraw();
}
```

The word wrap feature is activated by calling `Fl_Text_Editor::wrap_mode()` with the parameters `Fl_Text_Display::WRAP_AT_BOUNDS` and `0`. It's deactivated with `Fl_Text_Display::WRAP_NONE`. The implementation of the callback is the same as `menu_linenumbers_callback`.

1.7.10 Chapter 9: Split Editor

When editing long source code files, it can be really helpful to split the editor to view statements at the top of the text while adding features at the bottom of the text in a split text view.

FLTK can link multiple text editors to a single text buffer. Let's implement this now. This chapter will show you how to rearrange widgets in an existing widget tree.

Our initializer removes the main text editor from the widget tree and replaces it with an `Fl_Tile`. A tile can hold multiple widgets that can then be resized interactively by the user by clicking and dragging the divider between those widgets.

We start by replacing the editor widget with a tile group of the same size.

```
#include <FL/Fl_Tile.H>

Fl_Tile *app_tile = NULL;

void tut9_split_editor() {
  app_window->begin();
  app_tile = new Fl_Tile(app_editor->x(), app_editor->y(),
                         app_editor->w(), app_editor->h());
  app_window->remove(app_editor);
```

Next we add our existing editor as the first child of the tile and create another text editor `app_split_editor` as the second child of the tile, but it's hidden for now with a height of zero pixels.

Note

Creating the new `Fl_Tile` also calls `Fl_Tile::begin()`.

Adding `app_editor` to the tile would have also removed it from `app_window`, so `app_window->remove(app_editor)` in the code above is not really needed, but illustrates what we are doing.

```
app_tile->add(app_editor);
app_split_editor = new Fl_Text_Editor(app_tile->x(), app_tile->y()+app_tile->h(),
                                      app_tile->w(), 0);
app_split_editor->buffer(app_text_buffer);
app_split_editor->textfont(FL_COURIER);
app_split_editor->hide();
```

Now we clean up after ourselves and make sure that the resizables are all set correctly. Lastly, we add a menu item with a callback.

```
  app_tile->end();
  app_tile->size_range(0, 25, 25);
  app_tile->size_range(1, 25, 25);
  app_window->end();
  app_window->resizable(app_tile);
  app_tile->resizable(app_editor);
  app_menu_bar->add("Window/Split", FL_COMMAND+'i', menu_split_callback, NULL, FL_MENU_TOGGLE);
}
```

Now with all widgets in place, the callback's job is to show and resize, or hide and resize the split editor. We can implement that like here:

```
void menu_split_callback(Fl_Widget* w, void*) {
  Fl_Menu_Bar* menu = static_cast<Fl_Menu_Bar*>(w);
  const Fl_Menu_Item* splitview_item = menu->mvalue();
  if (splitview_item->value()) {
    int h_split = app_tile->h()/2;
    app_editor->size(app_tile->w(), h_split);
    app_split_editor->resize(app_tile->x(), app_tile->y() + h_split,
                             app_tile->w(), app_tile->h() - h_split);
    app_split_editor->show();
  } else {
    app_editor->size(app_tile->w(), app_tile->h());
    app_split_editor->resize(app_tile->x(), app_tile->y()+app_tile->h(),
                             app_tile->w(), 0);
    app_split_editor->hide();
  }
  app_tile->resizable(app_editor);
  app_tile->init_sizes();
  app_tile->redraw();
}
```

1.7.11 Chapter 10: Syntax Highlighting

Chapter 10 adds a lot of code to implement "C" language syntax highlighting. Not all code is duplicated here in the documentation. Please check out `test/editor.cxx` for all the details.

The Fl_Text_Editor widget supports highlighting of text with different fonts, colors, and sizes. The implementation is based on the excellent NEdit text editor core, from https://sourceforge.net/projects/nedit/, which uses a parallel "style" buffer which tracks the font, color, and size of the text that is drawn.

Styles are defined using the Fl_Text_Display::Style_Table_Entry structure defined in <FL/Fl_Text_Display.H>↩
:

```
struct Style_Table_Entry {
  Fl_Color color;
  Fl_Font  font;
  int      size;
  unsigned attr;
};
```

The `color` member sets the color for the text, the `font` member sets the FLTK font index to use, and the `size` member sets the pixel size of the text. The `attr` member is currently not used.

For our text editor we'll define 7 styles for plain code, comments, keywords, and preprocessor directives:

```
Fl_Text_Display::Style_Table_Entry styletable[] = {     // Style table
  { FL_BLACK,      FL_COURIER,        FL_NORMAL_SIZE }, // A - Plain
  { FL_DARK_GREEN, FL_COURIER_ITALIC, FL_NORMAL_SIZE }, // B - Line comments
  { FL_DARK_GREEN, FL_COURIER_ITALIC, FL_NORMAL_SIZE }, // C - Block comments
  { FL_BLUE,       FL_COURIER,        FL_NORMAL_SIZE }, // D - Strings
  { FL_DARK_RED,   FL_COURIER,        FL_NORMAL_SIZE }, // E - Directives
  { FL_DARK_RED,   FL_COURIER_BOLD,   FL_NORMAL_SIZE }, // F - Types
  { FL_BLUE,       FL_COURIER_BOLD,   FL_NORMAL_SIZE }  // G - Keywords
};
```

You'll notice that the comments show a letter next to each style - each style in the style buffer is referenced using a character starting with the letter 'A'.

You call the `highlight_data()` method to associate the style data and buffer with the text editor widget:

```
Fl_Text_Buffer *app_style_buffer;

app_editor->highlight_data(app_style_buffer, styletable,
                           sizeof(styletable) / sizeof(styletable[0]),
                           'A', style_unfinished_cb, 0);
```

Finally, you need to add a callback to the main text buffer so that changes to the text buffer are mirrored in the style buffer:

```
app_text_buffer->add_modify_callback(style_update, app_editor);
```

The `style_update()` function, like the `change_cb()` function described earlier, is called whenever text is added or removed from the text buffer. It mirrors the changes in the style buffer and then updates the style data as necessary:

```
//
// 'style_update()' - Update the style buffer...
//

void
style_update(int        pos,          // I - Position of update
             int        nInserted,    // I - Number of inserted chars
             int        nDeleted,     // I - Number of deleted chars
             int        nRestyled,    // I - Number of restyled chars
             const char *deletedText, // I - Text that was deleted
             void       *cbArg) {     // I - Callback data
  int  start,                         // Start of text
       end;                           // End of text
  char last,                          // Last style on line
       *style,                        // Style data
       *text;                         // Text data


  // If this is just a selection change, just unselect the style buffer...
  if (nInserted == 0 && nDeleted == 0) {
    app_style_buffer->unselect();
    return;
  }

  // Track changes in the text buffer...
  if (nInserted > 0) {
    // Insert characters into the style buffer...
    style = new char[nInserted + 1];
    memset(style, 'A', nInserted);
    style[nInserted] = '\0';

    app_style_buffer->replace(pos, pos + nDeleted, style);
    delete[] style;
```

```
  } else {
    // Just delete characters in the style buffer...
    app_style_buffer->remove(pos, pos + nDeleted);
  }

  // Select the area that was just updated to avoid unnecessary
  // callbacks...
  app_style_buffer->select(pos, pos + nInserted - nDeleted);

  // Re-parse the changed region; we do this by parsing from the
  // beginning of the line of the changed region to the end of
  // the line of the changed region...  Then we check the last
  // style character and keep updating if we have a multi-line
  // comment character...
  start = app_text_buffer->line_start(pos);
  end   = app_text_buffer->line_end(pos + nInserted - nDeleted);
  text  = app_text_buffer->text_range(start, end);
  style = app_style_buffer->text_range(start, end);
  last  = style[end - start - 1];

  style_parse(text, style, end - start);

  app_style_buffer->replace(start, end, style);
  ((Fl_Text_Editor *)cbArg)->redisplay_range(start, end);

  if (last != style[end - start - 1]) {
    // The last character on the line changed styles, so reparse the
    // remainder of the buffer...
    free(text);
    free(style);

    end   = app_text_buffer->length();
    text  = app_text_buffer->text_range(start, end);
    style = app_style_buffer->text_range(start, end);

    style_parse(text, style, end - start);

    app_style_buffer->replace(start, end, style);
    ((Fl_Text_Editor *)cbArg)->redisplay_range(start, end);
  }

  free(text);
  free(style);
}
```

The `style_parse()` function scans a copy of the text in the buffer and generates the necessary style characters for display. It assumes that parsing begins at the start of a line:

```
//
// 'style_parse()' - Parse text and produce style data.
//

void
style_parse(const char *text,
            char       *style,
            int        length) {
  char       current;
  int        col;
  int        last;
  char       buf[255],
             *bufptr;
  const char *temp;

  for (current = *style, col = 0, last = 0; length > 0; length --, text ++) {
    if (current == 'A') {
      // Check for directives, comments, strings, and keywords...
      if (col == 0 && *text == '#') {
        // Set style to directive
        current = 'E';
      } else if (strncmp(text, "//", 2) == 0) {
        current = 'B';
      } else if (strncmp(text, "/*", 2) == 0) {
        current = 'C';
      } else if (strncmp(text, "\\\"", 2) == 0) {
        // Quoted quote...
        *style++ = current;
        *style++ = current;
        text ++;
        length --;
        col += 2;
        continue;
      } else if (*text == '\"') {
        current = 'D';
      } else if (!last && islower(*text)) {
        // Might be a keyword...
        for (temp = text, bufptr = buf;
             islower(*temp) && bufptr < (buf + sizeof(buf) - 1);
```

```
            *bufptr++ = *temp++);

        if (!islower(*temp)) {
          *bufptr = '\0';

          bufptr = buf;

          if (bsearch(&bufptr, code_types,
                      sizeof(code_types) / sizeof(code_types[0]),
                      sizeof(code_types[0]), compare_keywords)) {
            while (text < temp) {
              *style++ = 'F';
              text ++;
              length --;
              col ++;
            }

            text --;
            length ++;
            last = 1;
            continue;
          } else if (bsearch(&bufptr, code_keywords,
                             sizeof(code_keywords) / sizeof(code_keywords[0]),
                             sizeof(code_keywords[0]), compare_keywords)) {
            while (text < temp) {
              *style++ = 'G';
              text ++;
              length --;
              col ++;
            }

            text --;
            length ++;
            last = 1;
            continue;
          }
        }
      }
    } else if (current == 'C' && strncmp(text, "*/", 2) == 0) {
      // Close a C comment...
      *style++ = current;
      *style++ = current;
      text ++;
      length --;
      current = 'A';
      col += 2;
      continue;
    } else if (current == 'D') {
      // Continuing in string...
      if (strncmp(text, "\\\"", 2) == 0) {
        // Quoted end quote...
        *style++ = current;
        *style++ = current;
        text ++;
        length --;
        col += 2;
        continue;
      } else if (*text == '\"') {
        // End quote...
        *style++ = current;
        col ++;
        current = 'A';
        continue;
      }
    }

    // Copy style info...
    if (current == 'A' && (*text == '{' || *text == '}')) *style++ = 'G';
    else *style++ = current;
    col ++;

    last = isalnum(*text) || *text == '.';

    if (*text == '\n') {
      // Reset column and possibly reset the style
      col = 0;
      if (current == 'B' || current == 'E') current = 'A';
    }
  }
}
```

1.8 Drawing Things in FLTK

This chapter covers the drawing functions that are provided with FLTK.

1.8.1 When Can You Draw Things in FLTK?

There are only certain places you can execute FLTK code that draws to the computer's display. Calling these functions at other places will result in undefined behavior!

- The most common place is inside the virtual Fl_Widget::draw() method. To write code here, you must subclass one of the existing Fl_Widget classes and implement your own version of draw().
- You can also create custom boxtypes and labeltypes. These involve writing small procedures that can be called by existing Fl_Widget::draw() methods. These "types" are identified by an 8-bit index that is stored in the widget's `box()`, `labeltype()`, and possibly other properties.
- You can call Fl_Window::make_current() to do incremental update of a widget. Use Fl_Widget::window() to find the window.

In contrast, code that draws to other drawing surfaces than the display (i.e., instances of derived classes of the Fl_Surface_Device class, except Fl_Display_Device, such as Fl_Printer and Fl_Copy_Surface) can be executed at any time as follows:

1. Make your surface the new current drawing surface calling the Fl_Surface_Device::push_current(Fl_Surface_Device∗) function.
2. Make a series of calls to any of the drawing functions described below; these will operate on the new current drawing surface;
3. Set the current drawing surface back to its previous state calling Fl_Surface_Device::pop_current().

1.8.2 What Units Do FLTK Functions Use?

Before version 1.4 all graphical quantities used by FLTK were in pixel units: a window of width 500 units was 500 pixels wide, a line of length 10 units was 10 pixels long, lines of text written using a 14-point font were 14 pixels below each other. This organization is not sufficient to support GUI apps that can be drawn on screens of varying pixel density, especially on High-DPI screens, because widgets become very small and text becomes unreadable.

FLTK version 1.4 introduces a new feature, a screen-specific **scale factor** which is a float number with a typical value in the 1-2.5 range and is used as follows: any graphical element with an FLTK value of v units is drawn on the screen with $v * scale$ units. Thus, a window with width 500 units is 500∗scale pixels wide, a line of length 10 units is 10∗scale pixels long, lines of text written using a 14-point font are 14∗scale pixels below each other. Consider a system with two screens, one with regular DPI and one with a twice higher DPI. If the first screen's scale factor is set to 1 and that of the second screen to 2, the GUI of any FLTK app appears equally sized on the two screens.

FLTK uses several units to measure graphical elements:

- All quantities used by the public FLTK API to measure graphical elements (e.g., window widths, line lengths, font sizes, clipping regions, image widths and heights) are in **FLTK units** except if it's explicitly documented another unit is used. FLTK units are both platform- and DPI-independent. An example of FLTK API using another unit is Fl_Gl_Window::pixel_w().

- Just before drawing to a screen, the library internally multiplies all quantities expressed in FLTK units by the current value of the scale factor for the screen in use and obtains quantities in **drawing units**. The current scale factor value, for an Fl_Window named *window*, is given by
```
int nscreen = window->screen_num();  // the screen where window is mapped
float s = Fl::screen_scale(nscreen); // this screen's scale factor
```
  One drawing unit generally corresponds to one screen pixel ...
- ... but not on macOS and for retina displays, where one drawing unit corresponds to two pixels.
- ... and not with the Wayland platform, where one drawing unit may correspond to 1, 2, or 3 pixels according to the current value of the Wayland-defined, integer-valued scale factor.

At application start time, FLTK attempts to detect the adequate scale factor value for each screen of the system. Here is how that's done under the X11, Windows, and Wayland platforms. If the resulting scale factor is not satisfactory, and also under the macOS platform, it's possible to set the `FLTK_SCALING_FACTOR` environmental variable to the desired numerical value (e.g., 1.75) and any FLTK app will start scaled with that value. Furthermore, it's possible to change the scale factor value of any screen at run time with ctrl/+/-/0/ keystrokes which enlarge, shrink, and reset, respectively, all FLTK windows on a screen and their content. Under macOS, the corresponding GUI scaling shortcuts are cmd/+/-/0/.

GUI rescaling involves also image drawing: the screen area covered by the drawn image contains a number of pixels that grows with the scale factor. When FLTK draws images, it maps the image data (the size of these data is given by Fl_Image::data_w() and Fl_Image::data_h()) to the screen area whose size (in FLTK units) is given by Fl_Image::w() and Fl_Image::h(). How exactly such mapping is performed depends on the image type, the platform and some hardware features. The most common case for Fl_RGB_Image's is that FLTK uses a scaled drawing system feature that directly maps image data to screen pixels. An important feature of FLTK for image drawing is the Fl_Image::scale() member function, new in FLTK version 1.4. This function controls the image drawing size (in FLTK units, given by Fl_Image::w() and Fl_Image::h()) independently from the size of the image data (given by Fl_Image::data_w() and Fl_Image::data_h()). An image with large enough data size can thus be drawn at the full resolution of the screen even when the screen area covered by the image grows following the GUI scale factor.

The Fl_Image_Surface class is intended to create an Fl_RGB_Image from a series of FLTK drawing operations. The Fl_Image_Surface constructor allows to control whether the size in pixels of the resulting image matches the FLTK units used when performing drawing operations, or matches the number of pixels corresponding to these FLTK units given the current value of the scale factor. The first result is obtained with `new Fl_Image_Surface(w, h)`, the second with `new Fl_Image_Surface(w, h, 1)`.

When drawing to Fl_Printer or Fl_PostScript_File_Device, the drawing unit is initially one point, that is, 1/72 of an inch. This unit is changed by calls to Fl_Paged_Device::scale().

1.8.3 Drawing Functions

To use the drawing functions you must first include the <FL/fl_draw.H> header file. FLTK provides the following types of drawing functions:

- Boxes
- Clipping
- Colors
- Color Contrast
- Line Dashes and Thickness
- Drawing Fast Shapes
- Drawing Complex Shapes

- Drawing Text
- Fonts
- Character Encoding
- Drawing Overlays
- Drawing Images
- Direct Image Drawing
- Direct Image Reading
- Image Classes
- Offscreen Drawing

1.8.3.1 Boxes

FLTK provides three functions that can be used to draw boxes for buttons and other UI controls. Each function uses the supplied upper-lefthand corner and width and height to determine where to draw the box.

void fl_draw_box(Fl_Boxtype b, int x, int y, int w, int h, Fl_Color c)

The `fl_draw_box`() function draws a standard boxtype `b` in the specified color `c`.

void fl_frame(const char *s, int x, int y, int w, int h)
void fl_frame2(const char *s, int x, int y, int w, int h)

The `fl_frame`() and `fl_frame2`() functions draw a series of line segments around the given box. The string `s` must contain groups of 4 letters which specify one of 24 standard grayscale values, where 'A' is black and 'X' is white. The results of calling these functions with a string that is not a multiple of 4 characters in length are undefined.

The only difference between `fl_frame`() and `fl_frame2`() is the order of the line segments:

- For `fl_frame`() the order of each set of 4 characters is: top, left, bottom, right.
- For `fl_frame2`() the order of each set of 4 characters is: bottom, right, top, left.

Note that fl_frame(Fl_Boxtype b) is described in the Box Types section.

1.8.3.2 Clipping

You can limit all your drawing to a rectangular region by calling `fl_push_clip`(), and put the drawings back by using `fl_pop_clip`(). This rectangle is measured in FLTK units and is unaffected by the current transformation matrix.

In addition, the system may provide clipping when updating windows which may be more complex than a simple rectangle.

void fl_push_clip(int x, int y, int w, int h)

Intersect the current clip region with a rectangle and push this new region onto the stack.

void fl_push_no_clip()

Pushes an empty clip region on the stack so nothing will be clipped.

void fl_pop_clip()

Restore the previous clip region.

Note: You must call `fl_pop_clip`() once for every time you call `fl_push_clip`(). If you return to FLTK with the clip stack not empty unpredictable results occur.

int fl_not_clipped(int x, int y, int w, int h)

Returns non-zero if any of the rectangle intersects the current clip region. If this returns 0 you don't have to draw the object.

Note: Under X this returns 2 if the rectangle is partially clipped, and 1 if it is entirely inside the clip region.

int fl_clip_box(int x, int y, int w, int h, int &X, int &Y, int &W, int &H)

Intersect the rectangle `x,y,w,h` with the current clip region and returns the bounding box of the result in `X,Y,W,H`. Returns non-zero if the resulting rectangle is different than the original. This can be used to limit the necessary drawing to a rectangle. `W` and `H` are set to zero if the rectangle is completely outside the region.

void fl_clip_region(Fl_Region r)
Fl_Region fl_clip_region()

Replace the top of the clip stack with a clipping region of any shape. Fl_Region is an operating system specific type. The second form returns the current clipping region.

1.8.3.3 Colors

FLTK manages colors as 32-bit unsigned integers, encoded as RGBI. When the "RGB" bytes are non-zero, the value is treated as RGB. If these bytes are zero, the "I" byte will be used as an index into the colormap. Colors with both "RGB" set and an "I" >0 are reserved for special use.

Values from 0 to 255, i.e. the "I" index value, represent colors from the FLTK standard colormap and are allocated as needed on screens without TrueColor support. The **Fl_Color** enumeration type defines the standard colors and color cube for the first 256 colors. All of these are named with symbols in <FL/Enumerations.H>. Example:

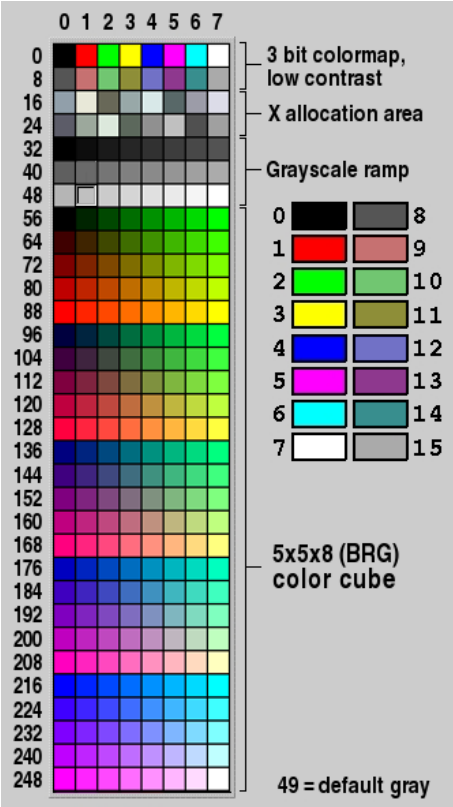


Figure 1.25 FLTK default colormap (Fl_Color 0x00 - 0xff)

Color values greater than 255 are treated as 24-bit RGB values. These are mapped to the closest color supported by the screen, either from one of the 256 colors in the FLTK colormap or a direct RGB value on TrueColor screens.

File FL/rgb_colors.H defines 490 constants of type Fl_Color than can be used to identify many colors by a name. The names of these constants (example: FL_RGB_ALICE_BLUE) begin with the "FL_RGB_" prefix and use the English-language color names of Xorg's "share/X11/rgb.txt" file and their values. Include <FL/rgb_colors.H> in your program to gain access to these color names.

Fl_Color fl_rgb_color(uchar r, uchar g, uchar b)
Fl_Color fl_rgb_color(uchar grayscale)

Generate Fl_Color out of specified 8-bit RGB values or one 8-bit grayscale value.

void fl_color(Fl_Color c)
void fl_color(int c)

Sets the color for all subsequent drawing operations. Please use the first form: the second form is only provided for back compatibility.

For colormapped displays, a color cell will be allocated out of `fl_colormap` the first time you use a color. If the colormap fills up then a least-squares algorithm is used to find the closest color.

Fl_Color fl_color()

Returns the last color that was set using `fl_color`(). This can be used for state save/restore.

void fl_color(uchar r, uchar g, uchar b)

Set the color for all subsequent drawing operations. The closest possible match to the RGB color is used. The RGB color is used directly on TrueColor displays. For colormap visuals the nearest index in the gray ramp or color cube is used.

unsigned Fl::get_color(Fl_Color i)
void Fl::get_color(Fl_Color i, uchar &red, uchar &green, uchar &blue)

Generate RGB values from a colormap index value `i`. The first returns the RGB as a 32-bit unsigned integer, and the second decomposes the RGB into three 8-bit values.

Fl::get_system_colors()
Fl::foreground()
Fl::background()
Fl::background2()

The first gets color values from the user preferences or the system, and the other routines are used to apply those values.

Fl::own_colormap()
Fl::free_color(Fl_Color i, int overlay)
Fl::set_color(Fl_Color i, unsigned c)

`Fl::own_colormap()` is used to install a local colormap [X11 only].

`Fl::free_color()` and `Fl::set_color()` are used to remove and replace entries from the colormap.

There are two predefined graphical interfaces for choosing colors. The function fl_show_colormap() shows a table of colors and returns an Fl_Color index value. The Fl_Color_Chooser widget provides a standard RGB color chooser.

As the Fl_Color encoding maps to a 32-bit unsigned integer representing RGBI, it is also possible to specify a color using a hex constant as a color map index:

```
// COLOR MAP INDEX
color(0x000000II)
         ------ |
            |   |
            |   Color map index (8 bits)
           Must be zero

button->color(0x000000ff);                // colormap index #255 (FL_WHITE)
```

or specify a color using a hex constant for the RGB components:

```
// RGB COLOR ASSIGNMENTS
color(0xRRGGBB00)
          | | | |
          | | | Must be zero
          | | Blue (8 bits)
          | Green (8 bits)
          Red (8 bits)

button->color(0xff000000);                // RGB: red
button->color(0x00ff0000);                // RGB: green
button->color(0x0000ff00);                // RGB: blue
button->color(0xffffff00);                // RGB: white
```

Note

If TrueColor is not available, any RGB colors will be set to the nearest entry in the colormap.

1.8.3.4 Color Contrast

Although these are not real "drawing" functions, creating readable contrast is essential in a good GUI design. FLTK tries to help with this by providing fl_contrast() and related functions.

The basic function is Fl_Color fl_contrast(Fl_Color fg, Fl_Color bg, int context, int size);

The parameters `context` and `size` are optional and reserved for future use (since FLTK 1.4.0).

The return value can be used to substitute the foreground color `fg` used for drawing (usually the "text" or "label" color) on a particular background color `bg` with either black (FL_BLACK) or white (FL_WHITE). This is useful if the background color is not known or can be changed by the user or a system "theme".

FLTK calculates the contrast between `fg` and `bg` and returns the same color (`fg`) if the contrast is considered sufficient or one of FL_BLACK or FL_WHITE if the contrast of the given foreground color would be insufficient. Then either FL_BLACK or FL_WHITE is chosen, whichever has the higher contrast with the background color.

Example, may be used in a widget's draw() method:

```
Fl_Color bg = color();          // background color of the widget
Fl_Color fg = FL_BLUE;          // the chosen foreground (drawing) color
fl_color(fl_contrast(fg, bg));  // set the drawing color
fl_rect(..);                    // draw a rectangle with sufficient contrast
```

FLTK 1.4.0 introduced a new contrast algorithm which is superior to the one used up to FLTK 1.3.x. You can use

fl_contrast_mode(FL_CONTRAST_LEGACY);

early in your program to select the old behavior if you really need strict backwards compatibility. This is discouraged because the new algorithm is much better with regard to human contrast perception. The default mode since FLTK 1.4.0 is

fl_contrast_mode(FL_CONTRAST_CIELAB);

For more info please see the linked documentation of these functions.

Additionally the old and new contrast calculations can be fine tuned with the new function (since 1.4.0)

fl_contrast_level(int level);

This is not recommended but can be useful for some border cases. Please refer to the documentation of fl_contrast_level().

Finally, developers can define their own contrast calculation function with

void fl_contrast_function(Fl_Contrast_Function *f);

Please see the documentation for details.

1.8.3.5 Line Dashes and Thickness

FLTK supports drawing of lines with different styles and widths.

void fl_line_style(int style, int width, char* dashes)

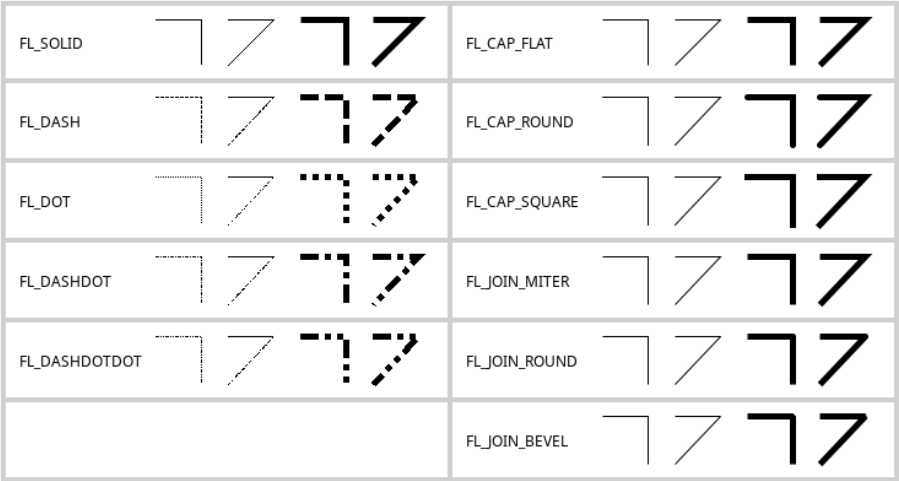


Figure 1.26 fl_line_style() styles

Set how to draw lines (the "pen"). If you change this it is your responsibility to set it back to the default with `fl_line_style(0)`.

`style` is a bitmask which is a bitwise-OR of the following values. If you don't specify a dash type you will get a solid line. If you don't specify a cap or join type you will get a system-defined default of whatever value is fastest.

- `FL_SOLID` solid line
- `FL_DASH` 75% dashed line
- `FL_DOT` 50% pixel dotted
- `FL_DASHDOT` dash / dot pattern
- `FL_DASHDOTDOT` dash / two dot pattern
- `FL_CAP_FLAT` end is flat
- `FL_CAP_ROUND` end is round
- `FL_CAP_SQUARE` extends past end point 1/2 line width
- `FL_JOIN_MITER` line join extends to a point
- `FL_JOIN_ROUND` line join is rounded
- `FL_JOIN_BEVEL` line join is flat (tidied)

`width` is the number of FLTK units thick to draw the lines. Zero results in the system-defined default, which on both X and Windows is somewhat different and nicer than 1.

`dashes` is a pointer to an array of dash lengths, measured in FLTK units. The first location is how long to draw a solid portion, the next is how long to draw the gap, then the solid, etc. It is terminated with a zero-length entry. A `NULL` pointer or a zero-length array results in a solid line. Odd array sizes are not supported and result in undefined behavior.

Note

- Full functionality is not available under Windows 95, 98, and Me due to the reduced drawing functionality these operating systems provide.
- Because of how line styles are implemented on Windows systems, you *must* set the line style *after* setting the drawing color. If you set the color after the line style you will lose the line style settings!
- The dashes array does not work under Windows 95, 98, or Me, since those operating systems do not support complex line styles.

1.8.3.6 Drawing Fast Shapes

These functions are used to draw almost all the FLTK widgets. They draw on exact pixel boundaries and are as fast as possible. Their behavior is duplicated exactly on all platforms FLTK is ported. It is undefined whether these are affected by the transformation matrix, so you should only call these while the matrix is set to the identity matrix (the default).

void fl_point(int x, int y)

Draw a single pixel at the given coordinates.

void fl_rectf(int x, int y, int w, int h)
void fl_rectf(int x, int y, int w, int h, Fl_Color c)

Color a rectangle that exactly fills the given bounding box.

void fl_rectf(int x, int y, int w, int h, uchar r, uchar g, uchar b)

Color a rectangle with "exactly" the passed `r,g,b` color. On screens with less than 24 bits of color this is done by drawing a solid-colored block using fl_draw_image() so that the correct color shade is produced.

void fl_rect(int x, int y, int w, int h)
void fl_rect(int x, int y, int w, int h, Fl_Color c)

Draw a 1-pixel border *inside* this bounding box.

void fl_rounded_rect(int x, int y, int w, int h, int radius) void fl_rounded_rectf(int x, int y, int w, int h, int radius)

Draw an outlined or filled rectangle with rounded corners.

void fl_line(int x, int y, int x1, int y1)
void fl_line(int x, int y, int x1, int y1, int x2, int y2)

Draw one or two lines between the given points.

void fl_loop(int x, int y, int x1, int y1, int x2, int y2)
void fl_loop(int x, int y, int x1, int y1, int x2, int y2, int x3, int y3)

Outline a 3 or 4-sided polygon with lines.

void fl_polygon(int x, int y, int x1, int y1, int x2, int y2)
void fl_polygon(int x, int y, int x1, int y1, int x2, int y2, int x3, int y3)

Fill a 3 or 4-sided polygon. The polygon must be convex.

void fl_xyline(int x, int y, int x1)
void fl_xyline(int x, int y, int x1, int y2)
void fl_xyline(int x, int y, int x1, int y2, int x3)

Draw horizontal and vertical lines. A horizontal line is drawn first, then a vertical, then a horizontal.

void fl_yxline(int x, int y, int y1)
void fl_yxline(int x, int y, int y1, int x2)
void fl_yxline(int x, int y, int y1, int x2, int y3)

Draw vertical and horizontal lines. A vertical line is drawn first, then a horizontal, then a vertical.

void fl_arc(int x, int y, int w, int h, double a1, double a2)
void fl_pie(int x, int y, int w, int h, double a1, double a2)

Draw ellipse sections using integer coordinates. These functions match the rather limited circle drawing code provided by X and Windows. The advantage over using fl_arc() with floating point coordinates is that they are faster because they often use the hardware, and they draw much nicer small circles, since the small sizes are often hard-coded bitmaps.

If a complete circle is drawn it will fit inside the passed bounding box. The two angles are measured in degrees counter-clockwise from 3'oclock and are the starting and ending angle of the arc, `a2` must be greater or equal to `a1`.

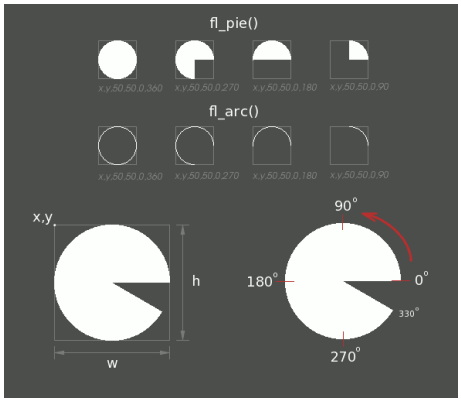


Figure 1.27 fl_pie() and fl_arc()

`fl_arc`() draws a series of lines to approximate the arc. Notice that the integer version of `fl_arc`() has a different number of arguments to the other fl_arc() function described later in this chapter.

`fl_pie`() draws a filled-in pie slice. This slice may extend outside the line drawn by `fl_arc`(); to avoid this use `w-1` and `h-1`.

void fl_scroll(int X, int Y, int W, int H, int dx, int dy, void (*draw_area)(void*, int,int,int,int), void* data)

Scroll a rectangle and draw the newly exposed portions. The contents of the rectangular area is first shifted by `dx` and `dy` FLTK units. The callback is then called for every newly exposed rectangular area,

1.8.3.7 Drawing Complex Shapes

The complex drawing functions let you draw arbitrary shapes with 2-D linear transformations. The functionality matches that found in the Adobe® PostScript™ language. The exact pixels that are filled are less defined than for the fast drawing functions so that FLTK can take advantage of drawing hardware. On both X and Windows the transformed vertices are rounded to integers before drawing the line segments: this severely limits the accuracy of these functions for complex graphics, so use OpenGL when greater accuracy and/or performance is required.

void fl_load_matrix(double a,double b,double c,double d,double x,double y) void fl_load_identity()

Set the current transformation.

void fl_push_matrix()
void fl_pop_matrix()

Save and restore the current transformation. The maximum depth of the stack is 32 entries.

void fl_scale(double x,double y)
void fl_scale(double x)
void fl_translate(double x,double y)
void fl_rotate(double d)
void fl_mult_matrix(double a,double b,double c,double d,double x,double y)

Concatenate another transformation onto the current one. The rotation angle is in degrees (not radians) and is counter-clockwise.

double fl_transform_x(double x, double y)
double fl_transform_y(double x, double y)
double fl_transform_dx(double x, double y)
double fl_transform_dy(double x, double y)
void fl_transformed_vertex(double xf, double yf)

Transform a coordinate or a distance using the current transformation matrix. After transforming a coordinate pair, it can be added to the vertex list without any further translations using `fl_transformed_vertex`().

void fl_begin_points()
void fl_end_points()

Start and end drawing a list of points. Points are added to the list with `fl_vertex`().

void fl_begin_line()
void fl_end_line()

Start and end drawing lines.

void fl_begin_loop()
void fl_end_loop()

Start and end drawing a closed sequence of lines.

void fl_begin_polygon()
void fl_end_polygon()

Start and end drawing a convex filled polygon.

void fl_begin_complex_polygon()
void fl_gap()
void fl_end_complex_polygon()

Start and end drawing a complex filled polygon. This polygon may be concave, may have holes in it, or may be several disconnected pieces. Call `fl_gap`() to separate loops of the path. It is unnecessary but harmless to call `fl_gap`() before the first vertex, after the last one, or several times in a row.

`fl_gap`() should only be called between `fl_begin_complex_polygon`() and `fl_end_complex↩_polygon`(). To outline the polygon, use `fl_begin_loop`() and replace each `fl_gap`() with a `fl_↩end_loop`();fl_begin_loop() pair.

Note: For portability, you should only draw polygons that appear the same whether "even/odd" or "non-zero" winding rules are used to fill them. Holes should be drawn in the opposite direction of the outside loop.

void fl_vertex(double x,double y)

Add a single vertex to the current path.

void fl_curve(double X0, double Y0, double X1, double Y1, double X2, double Y2, double X3, double Y3)

Add a series of points on a Bézier curve to the path. The curve ends (and two of the points are) at `X0,Y0` and `X3,Y3`.

void fl_arc(double x, double y, double r, double start, double end)

Add a series of points to the current path on the arc of a circle; you can get elliptical paths by using scale and rotate before calling `fl_arc`(). The center of the circle is given by `x` and `y`, and `r` is its radius. `fl_arc`() takes `start` and `end` angles that are measured in degrees counter-clockwise from 3 o'clock. If `end` is less than `start` then it draws the arc in a clockwise direction.

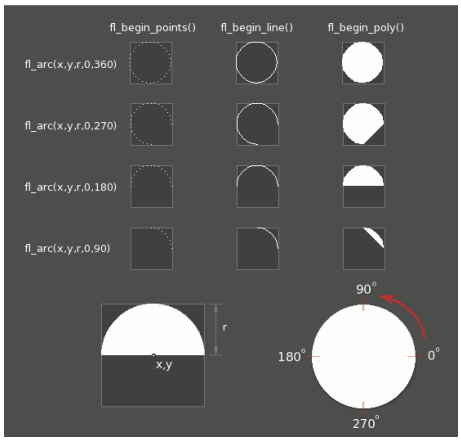


Figure 1.28 fl_arc(x,y,r,a1,a2)

void fl_circle(double x, double y, double r)

`fl_circle(x,y,r)` is equivalent to `fl_arc(x,y,r,0,360)` but may be faster. It must be the *only* thing in the path: if you want a circle as part of a complex polygon you must use `fl_arc`().

Note: `fl_circle`() draws incorrectly if the transformation is both rotated and non-square scaled.

1.8.3.8 Drawing Text

All text is drawn in the current font. It is undefined whether this location or the characters are modified by the current transformation.

void fl_draw(const char *, int x, int y)
void fl_draw(const char *, int n, int x, int y)

Draw a nul-terminated string or an array of `n` bytes starting at the given location. In both cases, the text must be UTF-8 encoded. Text is aligned to the left and to the baseline of the font. To align to the bottom, subtract `fl_descent`() from `y`. To align to the top, subtract `fl_descent`() and add `fl_height`(). This version of `fl_draw`() provides direct access to the text drawing function of the underlying OS. It does not apply any special handling to control characters.

void fl_rtl_draw(const char *str, int n, int x, int y)

Draw a UTF-8 string of length n bytes right to left starting at the given x, y location.

void fl_draw(const char* str, int x, int y, int w, int h, Fl_Align align, Fl_Image* img, int draw_symbols, int spacing)

Fancy string drawing function which is used to draw all the labels. The string is formatted and aligned inside the passed box. Handles '\t' and '\n', expands all other control characters to ^X, and aligns inside or against the edges of the box described by `x`, `y`, `w` and `h`. See Fl_Widget::align() for values for `align`. The value `FL_ALIGN_INSIDE` is ignored, as this function always prints inside the box. Parameter `spacing` controls the space between text and image.

If `img` is provided and is not `NULL`, the image is drawn above or below the text as specified by the `align` value.

The `draw_symbols` argument specifies whether or not to look for symbol names starting with the "@" character.

void fl_measure(const char *str, int& w, int& h, int draw_symbols)

Measure how wide and tall the string will be when printed by the `fl_draw(...align)` function. This includes leading/trailing white space in the string, kerning, etc.

If the incoming `w` is non-zero it will wrap to that width.

This will probably give unexpected values unless you have called fl_font() explicitly in your own code. Refer to the full documentation for fl_measure() for details on usage and how to avoid common pitfalls.

See also

fl_text_extents() – measure the 'inked' area of a string

fl_width() – measure the width of a string or single character

fl_height() – measure the height of the current font

fl_descent() – the height of the descender for the current font

int fl_height()

Recommended minimum line spacing for the current font. You can also just use the value of `size` passed to fl_font().

See also

fl_text_extents(), fl_measure(), fl_width(), fl_descent()

int fl_descent()

Recommended distance above the bottom of a `fl_height`() tall box to draw the text at so it looks centered vertically in that box.

double fl_width(const char* txt)
double fl_width(const char* txt, int n)
double fl_width(unsigned int unicode_char)

Return the width of a nul-terminated string, a sequence of `n` characters, or a single character in the current font.

See also

fl_measure(), fl_text_extents(), fl_height(), fl_descent()

void fl_text_extents(const char* txt, int& dx, int& dy, int& w, int& h)

Determines the minimum dimensions of a nul-terminated string, ie. the 'inked area'.

Given a string "txt" drawn using fl_draw(txt, x, y) you would determine its extents in FLTK units on the display using fl_text_extents(txt, dx, dy, wo, ho) such that a bounding box that exactly fits around the inked area of the text could be drawn with fl_rect(x+dx, y+dy, wo, ho).

Refer to the full documentation for fl_text_extents() for details on usage.

See also

fl_measure(), fl_width(), fl_height(), fl_descent()

const char* fl_shortcut_label(int shortcut)

Unparse a shortcut value as used by Fl_Button or Fl_Menu_Item into a human-readable string like "Alt+N". This only works if the shortcut is a character key or a numbered function key. If the shortcut is zero an empty string is returned. The return value points at a static buffer that is overwritten with each call.

1.8.3.9 Fonts

FLTK supports a set of standard fonts based on the Times, Helvetica/Arial, Courier, and Symbol typefaces, as well as custom fonts that your application may load. Each font is accessed by an index into a font table.

Initially only the first 16 faces are filled in. There are symbolic names for them: FL_HELVETICA, FL_TIMES, FL↩_COURIER, and modifier values FL_BOLD and FL_ITALIC which can be added to these, and FL_SYMBOL and FL_ZAPF_DINGBATS. Faces greater than 255 cannot be used in Fl_Widget labels, since Fl_Widget stores the index as a byte.

One important thing to note about 'current font' is that there are so many paths through the GUI event handling code as widgets are partially or completely hidden, exposed and then re-drawn and therefore you can not guarantee that 'current font' contains the same value that you set on the other side of the event loop. Your value may have been superseded when a widget was redrawn. You are strongly advised to set the font explicitly before you draw any text or query the width and height of text strings, etc.

void fl_font(int face, int size)

> Set the current font, which is then used by the routines described above. You may call this outside a draw context if necessary to call fl_width(), but on X this will open the display.

> The font is identified by a `face` and a `size`. The size of the font is measured in FLTK units and not "points". Lines should be spaced `size` FLTK units apart or more.

int fl_font()
int fl_size()

> Returns the face and size set by the most recent call to `fl_font(a,b)`. This can be used to save/restore the font.

1.8.3.10 Character Encoding

FLTK 1.3 and later versions expect all text in Unicode UTF-8 encoding. UTF-8 is ASCII compatible for the first 128 characters. International characters are encoded in multibyte sequences.

FLTK expects individual characters, characters that are not part of a string, in UCS-4 encoding, which is also ASCII compatible, but requires 4 bytes to store a Unicode character.

FLTK can draw accurately any Unicode-supported script for which the system contains relevant fonts. Under X11 platforms, this requires to build the library with the FLTK_USE_PANGO CMake option turned On.

Plain text drawing starting at a user-given coordinate is well supported by FLTK, including for right-to-left scripts. Further text-related operations (i.e., selection, formatting, input, and editing) are functional with left-to-right scripts only.

For more information about character encodings, see the chapter on Unicode and UTF-8 Support.

1.8.3.11 Drawing Overlays

These functions allow you to draw interactive selection rectangles without using the overlay hardware. FLTK will XOR a single rectangle outline over a window.

void fl_overlay_rect(int x, int y, int w, int h)
void fl_overlay_clear()

`fl_overlay_rect`() draws a selection rectangle, erasing any previous rectangle by XOR'ing it first. `fl_overlay_clear`() will erase the rectangle without drawing a new one.

Using these functions is tricky. You should make a widget with both a `handle()` and `draw()` method. `draw()` should call `fl_overlay_clear`() before doing anything else. Your `handle()` method should call `window()->make_current()` and then `fl_overlay_rect`() after FL_DRAG events, and should call `fl_overlay_clear`() after a FL_RELEASE event.

1.8.4 Drawing Images

To draw images, you can either do it directly from data in your memory, or you can create a Fl_Image object. The advantage of drawing directly is that it is more intuitive, and it is faster if the image data changes more often than it is redrawn. The advantage of using the object is that FLTK will cache translated forms of the image (on X it uses a server pixmap) and thus redrawing is *much* faster.

1.8.4.1 Direct Image Drawing

The behavior when drawing images when the current transformation matrix is not the identity is not defined, so you should only draw images when the matrix is set to the identity.

void fl_draw_image(const uchar *buf,int X,int Y,int W,int H,int D,int L)
void fl_draw_image_mono(const uchar *buf,int X,int Y,int W,int H,int D,int L)

Draw an 8-bit per color RGB or luminance image. The pointer points at the "r" data of the top-left pixel. Color data must be in `r,g,b` order. The top left corner is given by `X` and `Y` and the size of the image is given by `W` and `H`. `D` is the delta to add to the pointer between pixels, it may be any value greater or equal to `3`, or it can be negative to flip the image horizontally. `L` is the delta to add to the pointer between lines (if 0 is passed it uses `W*D`). and may be larger than `W*D` to crop data, or negative to flip the image vertically.

It is highly recommended that you put the following code before the first show() of *any* window in your program to get rid of the dithering if possible:

```
Fl::visual(FL_RGB);
```

Gray scale (1-channel) images may be drawn. This is done if `abs(D)` is less than 3, or by calling `fl_draw_image_mono`(). Only one 8-bit sample is used for each pixel, and on screens with different numbers of bits for red, green, and blue only gray colors are used. Setting `D` greater than 1 will let you display one channel of a color image.

Note: The X version does not support all possible visuals. If FLTK cannot draw the image in the current visual it will abort. FLTK supports any visual of 8 bits or less, and all common TrueColor visuals up to 32 bits.

typedef void (∗Fl_Draw_Image_Cb)(void ∗data,int x,int y,int w,uchar ∗buf)
void fl_draw_image(Fl_Draw_Image_Cb cb,void ∗data,int X,int Y,int W,int H,int D)
void fl_draw_image_mono(Fl_Draw_Image_Cb cb,void ∗data,int X,int Y,int W,int H,int D)

Call the passed function to provide each scan line of the image. This lets you generate the image as it is being drawn, or do arbitrary decompression of stored data, provided it can be decompressed to individual scan lines easily.

The callback is called with the `void*` user data pointer which can be used to point at a structure of information about the image, and the `x`, `y`, and `w` of the scan line desired from the image. 0,0 is the upper-left corner of the image, *not* `X, Y`. A pointer to a buffer to put the data into is passed. You must copy `w` pixels from scanline `y`, starting at pixel `x`, to this buffer.

Due to cropping, less than the whole image may be requested. So `x` may be greater than zero, the first `y` may be greater than zero, and `w` may be less than `W`. The buffer is long enough to store the entire `W*D` pixels, this is for convenience with some decompression schemes where you must decompress the entire line at once: decompress it into the buffer, and then if `x` is not zero, copy the data over so the `x'th` pixel is at the start of the buffer.

You can assume the `y's` will be consecutive, except the first one may be greater than zero.

If `D` is 4 or more, you must fill in the unused bytes with zero.

int fl_draw_pixmap(char∗ const∗ data, int x, int y, Fl_Color bg)
int fl_draw_pixmap(const char∗ const∗ cdata, int x, int y, Fl_Color bg)

Draws XPM image data, with the top-left corner at the given position. The image is dithered on 8-bit displays so you won't lose color space for programs displaying both images and pixmaps. This function returns zero if there was any error decoding the XPM data.

To use an XPM, do:

```
#include "foo.xpm"
...
fl_draw_pixmap(foo, X, Y);
```

Transparent colors are replaced by the optional Fl_Color argument. To draw with true transparency you must use the Fl_Pixmap class.

int fl_measure_pixmap(char∗ const∗ data, int &w, int &h)
int fl_measure_pixmap(const char∗ const∗ cdata, int &w, int &h)

An XPM image contains the dimensions in its data. This function finds and returns the width and height. The return value is non-zero if the dimensions were parsed ok and zero if there was any problem.

1.8.4.2 Direct Image Reading

FLTK provides a single function for reading from the current window or off-screen buffer into a RGB(A) image buffer.

uchar* fl_read_image(uchar *p, int X, int Y, int W, int H, int alpha)

Read a RGB(A) image from the current window or off-screen buffer. The `p` argument points to a buffer that can hold the image and must be at least `W*H*3` bytes when reading RGB images and `W*H*4` bytes when reading RGBA images. If `NULL`, `fl_read_image`() will create an array of the proper size which can be freed using `delete`[].

The `alpha` parameter controls whether an alpha channel is created and the value that is placed in the alpha channel. If 0, no alpha channel is generated.

1.8.4.3 Direct Image Saving

FLTK allows to save an array of pixel values to a file either in PNG or JPEG formats, provided the app is linked to the `fltk_images` library:

int fl_write_png(const char *filename, const char *pixels, int w, int h, int d, int ld)

Saves an array of pixels to a PNG file.

int fl_write_jpeg(const char *filename, const char *pixels, int w, int h, int d, int ld)

Saves an array of pixels to a JPEG file.

1.8.4.4 Image Classes

FLTK provides a base image class called Fl_Image which supports creating, copying, and drawing images of various kinds, along with some basic color operations. Images can be used as labels for widgets using the `image()` and `deimage()` methods or drawn directly. Images can be drawn scaled to any size, independently from the size of the image's data (see Fl_Image::scale()).

The Fl_Image class does almost nothing by itself, but is instead supported by three basic image types:

- Fl_Bitmap
- Fl_Pixmap
- Fl_RGB_Image

The Fl_Bitmap class encapsulates a mono-color bitmap image. The `draw()` method draws the image using the current drawing color.

The Fl_Pixmap class encapsulates a colormapped image. The `draw()` method draws the image using the colors in the file, and masks off any transparent colors automatically.

The Fl_RGB_Image class encapsulates a full-color (or grayscale) image with 1 to 4 color components. Images with an even number of components are assumed to contain an alpha channel that is used for transparency. The transparency provided by the draw() method is either a 24-bit blend against the existing window contents or a "screen door" transparency mask, depending on the platform and screen color depth.

char fl_can_do_alpha_blending()

`fl_can_do_alpha_blending()` will return 1, if your platform supports true alpha blending for RGBA images, or 0, if FLTK will use screen door transparency.

FLTK also provides several image classes based on the three standard image types for common file formats:

- Fl_GIF_Image
- Fl_Anim_GIF_Image
- Fl_JPEG_Image
- Fl_PNG_Image
- Fl_PNM_Image
- Fl_XBM_Image
- Fl_XPM_Image
- Fl_SVG_Image
- Fl_BMP_Image
- Fl_ICO_Image

Each of these image classes loads a named file of the corresponding format. The Fl_Shared_Image class can be used to load any type of image file - the class examines the file and constructs an image of the appropriate type.

Finally, FLTK provides a special image class called Fl_Tiled_Image to tile another image object in the specified area. This class can be used to tile a background image in a Fl_Group widget, for example.

virtual Fl_Image* Fl_Image::copy()
virtual Fl_Image* Fl_Image::copy(int W, int H) const

The `copy()` method creates a copy of the image. The second form specifies the new size of the image - the image is resized using the nearest-neighbor algorithm (this is the default).

Note

As of FLTK 1.3.3 the image resizing algorithm can be changed. See Fl_Image::RGB_scaling(Fl_RGB_Scaling method)

virtual void Fl_Image::draw(int x, int y, int w, int h, int ox, int oy)

The `draw()` method draws the image object. `x,y,w,h` indicates the destination rectangle. `ox,oy,w,h` is the source rectangle. This source rectangle is copied to the destination. The source rectangle may extend outside the image, i.e. `ox` and `oy` may be negative and `w` and `h` may be bigger than the image, and this area is left unchanged.

Note

See exceptions for Fl_Tiled_Image::draw() regarding arguments `ox`, `oy`, `w`, and `h`.

virtual void Fl_Image::draw(int x, int y)

Draws the image with the upper-left corner at `x, y`. This is the same as doing `img->draw(x, y, img->w(), img->h(), 0, 0)` where img is a pointer to any Fl_Image type.

1.8.5 Offscreen Drawing

Sometimes it can be very useful to generate a complex drawing in memory first and copy it to the screen at a later point in time. This technique can significantly reduce the amount of repeated drawing. Offscreen drawing functions are declared in <FL/fl_draw.H>.

Fl_Double_Window uses offscreen rendering to avoid flickering on systems that don't support double-buffering natively.

FLTK can draw into an offscreen buffer at any time. There is no need to wait for an Fl_Widget::draw() to occur.

Note

In FLTK 1.3.x and earlier versions all offscreen drawing functions described below were implemented as macros and created certain temporary variables to save context information. You needed to create local scope blocks with curly braces { ... } if you used offscreen functions more than once in a function or method. This is no longer necessary since offscreen drawing is now implemented in real functions (no macros).

Example:

```
Fl_Offscreen oscr = fl_create_offscreen(120, 120);
fl_begin_offscreen(oscr);
fl_color(FL_WHITE);
fl_rectf(0, 0, 120, 120);
fl_end_offscreen();
// other code here
fl_begin_offscreen(oscr);
fl_color(FL_BLACK);
fl_rectf(10, 10, 100, 100);
fl_end_offscreen();
// other code here
fl_delete_offscreen(oscr);
```

Fl_Offscreen fl_create_offscreen(int w, int h)

Create an RGB offscreen buffer containing as many pixels as in a screen area of size `w`,h FLTK units.

void fl_delete_offscreen(Fl_Offscreen)

Delete a previously created offscreen buffer. All drawings are lost.

void fl_begin_offscreen(Fl_Offscreen)

Send all subsequent drawing commands to this offscreen buffer.

void fl_end_offscreen()

Quit sending drawing commands to this offscreen buffer.

void fl_copy_offscreen(int x, int y, int w, int h, Fl_Offscreen osrc, int srcx, int srcy)

Copy a rectangular area of the size `w*h` from `srcx`,srcy in the offscreen buffer into the current drawing surface at `x`,y.

void fl_rescale_offscreen(Fl_Offscreen &osrc)

Adapts the offscreen's size in pixels to a changed value of the scale factor while keeping the offscreen's graphical content.

1.9 Handling Events

This chapter discusses the FLTK event model and how to handle events in your program or widget.

1.9.1 The FLTK Event Model

Every time a user moves the mouse pointer, clicks a button, or presses a key, an event is generated and sent to your application. Events can also come from other programs like the window manager.

Events are identified by the integer argument passed to a `handle()` method that overrides the Fl_Widget::handle() virtual method. Other information about the most recent event is stored in static locations and acquired by calling the Fl::event_∗() methods. This static information remains valid until the next event is read from the window system, so it is ok to look at it outside of the `handle()` method, for instance in a callback.

Event numbers can be converted to their actual names using the fl_eventnames[] array (std::map since FLTK 1.5) defined in #include <FL/names.h>; see next chapter for details.

In the next chapter, the MyClass::handle() example shows how to override the Fl_Widget::handle() method to accept and process specific events.

1.9.2 Mouse Events

Classic mice provide two or three buttons:

- a primary (left) button, typically used to point and click on objects
- a secondary (right) button, typically used to open special menus etc.
- a middle button for other special events.

The primary and secondary mouse buttons can usually be inverted by system setup functions for left handed usage. FLTK always sends the mouse button number according to the "logical" mouse button, i.e. such a setup is transparent for application programmers.

The middle button was later replaced by a scroll wheel to allow scrolling text or other objects in a window or widget vertically. Pushing the scroll wheel down acts like pushing the middle mouse button.

If the scroll wheel is used to scroll while the Shift key is held, the scrolling direction is horizontal rather than vertical.

Some mice may even have a "scroll ball" or similar (touch) feature. These mice support horizontal and vertical scrolling simultaneously.

Newer mice add even more buttons, called "side buttons":

- a "back" button, typically used to "go back", e.g. in browsers
- a "forward" button, typically used to "go forward", e.g. in browsers

FLTK handles up to five buttons since version 1.3.10 across all supported platforms. Note that Windows doesn't support more than five buttons whereas other platforms may support more. FLTK ignores other buttons as long as they send "mouse button" events.

Some gaming mice with five or more buttons may send their button clicks as keyboard events which will be handled like normal text input by FLTK. This has not been tested though.

1.9.2.1 FL_PUSH

A mouse button has gone down with the mouse pointing at this widget. You can find out what button by calling Fl::event_button(). You find out the mouse position by calling Fl::event_x() and Fl::event_y().

A widget indicates that it *"wants"* the mouse click by returning non-zero from its `handle()` method, as in the MyClass::handle() example. It will then become the Fl::pushed() widget and will get `FL_DRAG` and the matching `FL_RELEASE` events. If `handle()` returns zero then FLTK will try sending the `FL_PUSH` to another widget.

1.9.2.2 FL_DRAG

The mouse has moved with at least one button held down. The current button state is in Fl::event_state(). The mouse position is in Fl::event_x() and Fl::event_y().

In order to receive `FL_DRAG` events, the widget must return non-zero when handling `FL_PUSH`.

Moving the mouse keeps sending FL_DRAG events as long as at least one button is held down. If any buttons are pushed and released during the drag operation FLTK sends FL_PUSH and FL_RELEASE events even while dragging. The current button status can be found in Fl::event_state() and in Fl::event_buttons().

When the last button has been released FLTK sends FL_MOVE events again. Note: The button released last need not necessarily be the one that started the drag operation.

Since FLTK 1.4.0 dragging is supported for all five supported mouse buttons, for instance the user can drag the mouse while the "back" button is held down. Warning: This may or may not be compatible with other applications for drag-and-drop operations **between** applications and lead to unexpected behavior.

1.9.2.3 FL_RELEASE

A mouse button has been released. You can find out what button by calling Fl::event_button().

In order to receive the `FL_RELEASE` event, the widget must return non-zero when handling `FL_PUSH`.

1.9.2.4 FL_MOVE

The mouse has moved without any mouse buttons held down. This event is sent to the Fl::belowmouse() widget.

In order to receive `FL_MOVE` events, the widget must return non-zero when handling `FL_ENTER`.

1.9.2.5 FL_MOUSEWHEEL

The user has moved the mouse wheel. The Fl::event_dx() and Fl::event_dy() methods can be used to find the amount to scroll horizontally (dx) and vertically (dy). On mice with a single scroll wheel holding the Shift key on the keyboard while scrolling sends horizontal scroll events.

1.9.3 Focus Events

1.9.3.1 FL_ENTER

The mouse has been moved to point at this widget. This can be used for highlighting feedback. If a widget wants to highlight or otherwise track the mouse, it indicates this by returning non-zero from its `handle()` method. It then becomes the Fl::belowmouse() widget and will receive `FL_MOVE` and `FL_LEAVE` events.

1.9.3.2 FL_LEAVE

The mouse has moved out of the widget.

In order to receive the `FL_LEAVE` event, the widget must return non-zero when handling `FL_ENTER`.

1.9.3.3 FL_FOCUS

This indicates an *attempt* to give a widget the keyboard focus.

If a widget wants the focus, it should change itself to display the fact that it has the focus, and return non-zero from its `handle()` method. It then becomes the Fl::focus() widget and gets `FL_KEYDOWN`, `FL_KEYUP`, and `FL_UNFOCUS` events.

The focus will change either because the window manager changed which window gets the focus, or because the user tried to navigate using tab, arrows, or other keys. You can check Fl::event_key() to figure out why it moved. For navigation it will be the key pressed and for interaction with the window manager it will be zero.

1.9.3.4 FL_UNFOCUS

This event is sent to the previous Fl::focus() widget when another widget gets the focus or the window loses focus.

1.9.4 Keyboard Events

1.9.4.1 FL_KEYBOARD, FL_KEYDOWN, FL_KEYUP

A key was pressed (FL_KEYDOWN) or released (FL_KEYUP). FL_KEYBOARD is a synonym for FL_KEYDOWN, and both names are used interchangeably in this documentation.

The key can be found in Fl::event_key(). The text that the key should insert can be found with Fl::event_text() and its length is in Fl::event_length().

If you use the key, then `handle()` should return 1. If you return zero then FLTK assumes you ignored the key and will then attempt to send it to a parent widget. If none of them want it, it will change the event into a `FL_SHORTCUT` event. FL_KEYBOARD events are also generated by the character palette/map.

To receive `FL_KEYBOARD` events you must also respond to the `FL_FOCUS` and `FL_UNFOCUS` events by returning 1. This way FLTK knows whether to bother sending your widget keyboard events. (Some widgets don't need them, e.g. Fl_Box.)

If you are writing a text-editing widget you may also want to call the Fl::compose() function to translate individual keystrokes into characters.

`FL_KEYUP` events are sent to the widget that currently has focus. This is not necessarily the same widget that received the corresponding `FL_KEYDOWN` event because focus may have changed between events.

Todo Add details on how to detect repeating keys, since on some X servers a repeating key will generate both FL_KEYUP and FL_KEYDOWN, such that to tell if a key is held, you need Fl::event_key(int) to detect if the key is being held down during FL_KEYUP or not.

1.9.4.2 FL_SHORTCUT

If the Fl::focus() widget is zero or ignores an `FL_KEYBOARD` event then FLTK tries sending this event to every widget it can, until one of them returns non-zero. `FL_SHORTCUT` is first sent to the Fl::belowmouse() widget, then its parents and siblings, and eventually to every widget in the window, trying to find an object that returns non-zero. FLTK tries really hard to not to ignore any keystrokes!

You can also make "global" shortcuts by using Fl::add_handler(). A global shortcut will work no matter what windows are displayed or which one has the focus.

Since version 1.4, FLTK has 3 default global shortcuts (`Ctrl/+/-/0/` [`Cmd/+/-/0/` under macOS]) that change the value of the GUI scaling factor. `Ctrl+` zooms-in all app windows of the focussed display (all displays under macOS); `Ctrl-` zooms-out these windows; `Ctrl` 0 restores the initial value of the scaling factor. If any window of the display is fullscreen or maximized, scaling shortcuts have no effect. It's possible to deactivate FLTK's default scaling shortcuts with function Fl::keyboard_screen_scaling().

Option Fl::OPTION_SIMPLE_ZOOM_SHORTCUT can facilitate the typing necessary to trigger the zoom-in operation with those keyboard layouts where character '+' is located in the shifted position of its key: when this option is On it's not necessary to press also the Shift key to zoom-in.

These scaling shortcuts are installed when the FLTK library opens the display. They have a lower priority than any shortcut defined in any menu and than any user-provided event handler (see Fl::add_handler()) installed after FLTK opened the display. Therefore, if a menu item of an FLTK app is given FL_COMMAND+'+' as shortcut, that item's callback rather than FLTK's default zooming-in routine is triggered when `Ctrl+` (`Cmd+` under macOS) is pressed.

FLTK sends the FL_ZOOM_EVENT when the scaling factor value changes, to which a callback can be associated with Fl::add_handler(). By default, FLTK displays the new scaling factor value in a yellow, transient window. This can be changed with option Fl::OPTION_SHOW_SCALING.

1.9.5 Widget Events

1.9.5.1 FL_DEACTIVATE

This widget is no longer active, due to deactivate() being called on it or one of its parents. Please note that although active() may still return true for this widget after receiving this event, it is only truly active if active() is true for both it and all of its parents. (You can use active_r() to check this).

1.9.5.2 FL_ACTIVATE

This widget is now active, due to activate() being called on it or one of its parents.

1.9.5.3 FL_HIDE

This widget is no longer visible, due to hide() being called on it or one of its parents, or due to a parent window being minimized.

See also

FL_HIDE

1.9.5.4 FL_SHOW

This widget is visible again, due to show() being called on it or one of its parents, or due to a parent window being restored. *A child Fl_Window will respond to this by actually creating the window if not done already, so if you subclass a window, be sure to pass `FL_SHOW` to the base class `handle()` method!*

Note

The events in this chapter ("Widget Events"), i.e. FL_ACTIVATE, FL_DEACTIVATE, FL_SHOW, and FL_HIDE, are the only events deactivated and invisible widgets can usually get, depending on their states. Under certain circumstances, there may also be FL_LEAVE or FL_UNFOCUS events delivered to deactivated or hidden widgets.

1.9.6 Clipboard Events

1.9.6.1 FL_PASTE

You should get this event some time after you call Fl::paste(). The contents of Fl::event_text() is the text to insert and the number of characters is in Fl::event_length().

1.9.6.2 FL_SELECTIONCLEAR

The Fl::selection_owner() will get this event before the selection is moved to another widget. This indicates that some other widget or program has claimed the selection. Motif programs used this to clear the selection indication. Most modern programs ignore this.

1.9.7 Drag and Drop Events

FLTK supports drag and drop of text and files from any application on the desktop to an FLTK widget. Text is transferred using UTF-8 encoding.

See Fl::dnd() for drag and drop from an FLTK widget.

The drag and drop data is available in Fl::event_text() at the concluding `FL_PASTE`. On some platforms, the event text is also available for the `FL_DND_*` events, however application must not depend on that behavior because it depends on the protocol used on each platform.

`FL_DND_*` events cannot be used in widgets derived from Fl_Group or Fl_Window.

1.9.7.1 Dropped filenames

Files are received as a list of full path and file names.

- On some X11 platforms, files are received as a URL-encoded UTF-8 string, that is, non-ASCII bytes (and a few others such as space and %) are replaced by the 3 bytes "%XY" where XY are the byte's hexadecimal value. The fl_decode_uri() function can be used to transform in-place the received string into a proper UTF-8 string. On these platforms, strings corresponding to dropped files are further prepended by `file://` (or other prefixes such as `computer://`).
- Other X11 situations put all dropped filenames in a single line, separated by spaces.
- On non-X11 platforms, including Wayland, files dropped are received one pathname per line, with no `'\n'` after the last pathname.

1.9.7.2 FL_DND_ENTER

The mouse has been moved to point at this widget. A widget that is interested in receiving drag'n'drop data must return 1 to receive `FL_DND_DRAG`, `FL_DND_LEAVE` and `FL_DND_RELEASE` events.

1.9.7.3 FL_DND_DRAG

The mouse has been moved inside a widget while dragging data. A widget that is interested in receiving drag'n'drop data should indicate the possible drop position.

1.9.7.4 FL_DND_LEAVE

The mouse has moved out of the widget.

1.9.7.5 FL_DND_RELEASE

The user has released the mouse button dropping data into the widget. When the receiving widget's handle() method gets the FL_DND_RELEASE event, it should return 1 to accept the dragged data. Processing of this event must not use code that would make unrelated events be sent to the application (opening a dialog window for example) or that would communicate with the dragging process. The next event received by the handle() method will then be an FL_PASTE event. The handle() method should process this FL_PASTE event rapidly to prevent the dragging process from failing with a timeout error.

1.9.8 Other events

1.9.8.1 FL_SCREEN_CONFIGURATION_CHANGED

Sent whenever the screen configuration changes (a screen is added/removed, a screen resolution is changed, screens are moved). Use Fl::add_handler() to be notified of this event.

1.9.8.2 FL_FULLSCREEN

The application window has been changed from normal to fullscreen, or from fullscreen to normal. If you are using a X window manager which supports Extended Window Manager Hints, this event will not be delivered until the change has actually happened.

1.9.9 Fl::event_*() methods

FLTK keeps the information about the most recent event in static storage. This information is good until the next event is processed. Thus it is valid inside `handle()` and `callback()` methods.

These are all trivial inline functions and thus very fast and small:

- Fl::event_button()
- Fl::event_clicks()
- Fl::event_dx()
- Fl::event_dy()
- Fl::event_inside()
- Fl::event_is_click()
- Fl::event_key()
- Fl::event_length()
- Fl::event_state()
- Fl::event_text()
- Fl::event_x()
- Fl::event_x_root()
- Fl::event_y()
- Fl::event_y_root()
- Fl::get_key()
- Fl::get_mouse()
- Fl::test_shortcut()

1.9.10 Event Propagation

Widgets receive events via the virtual handle() function. The argument indicates the type of event that can be handled. The widget must indicate if it handled the event by returning 1. FLTK will then remove the event and wait for further events from the host. If the widget's handle function returns 0, FLTK may redistribute the event based on a few rules.

Most events are sent directly to the `handle()` method of the Fl_Window that the window system says they belong to. The window (actually the Fl_Group that Fl_Window is a subclass of) is responsible for sending the events on to any child widgets. To make the Fl_Group code somewhat easier, FLTK sends some events (`FL_DRAG`, `FL_RELEASE`, `FL_KEYBOARD`, `FL_SHORTCUT`, `FL_UNFOCUS`, and `FL_LEAVE`) directly to leaf widgets. These procedures control those leaf widgets:

- Fl::add_handler()
- Fl::belowmouse()
- Fl::focus()
- Fl::grab()

- Fl::modal()
- Fl::pushed()
- Fl_Widget::take_focus()

FLTK propagates events along the widget hierarchy depending on the kind of event and the status of the UI. Some events are injected directly into the widgets, others may be resent as new events to a different group of receivers.

Mouse click events are first sent to the window that caused them. The window then forwards the event down the hierarchy until it reaches the widget that is below the click position. If that widget uses the given event, the widget is marked "pushed" and will receive all following mouse motion (FL_DRAG) events until the mouse button is released.

Mouse motion (FL_MOVE) events are sent to the Fl::belowmouse() widget, i.e. the widget that returned 1 on the last FL_ENTER event.

Mouse wheel events are sent to the window that caused the event. The window propagates the event down the tree, first to the widget that is below the mouse pointer, and if that does not succeed, to all other widgets in the group. This ensures that scroll widgets work as expected with the widget furthest down in the hierarchy getting the first opportunity to use the wheel event, but also giving scroll bars, that are not directly below the mouse a chance.

Keyboard events are sent directly to the widget that has keyboard focus. If the focused widget rejects the event, it is resent as a shortcut event, first to the top-most window, then to the widget below the mouse pointer, propagating up the hierarchy to all its parents. Those send the event also to all widgets that are not below the mouse pointer. Now if that did not work out, the shortcut is sent to all registered shortcut handlers.

If we are still unsuccessful, the event handler flips the case of the shortcut letter and starts over. Finally, if the key is "escape", FLTK sends a close event to the top-most window.

All other events are pretty much sent right away to the window that created the event.

Widgets can "grab" events. The grabbing window gets all events exclusively, but usually by the same rules as described above.

Windows can also request exclusivity in event handling by making the window modal.

1.9.11 FLTK Compose-Character Sequences

The character composition done by Fl_Input widget requires that you call the Fl::compose() function if you are writing your own text editor widget.

Currently, all characters made by single key strokes with or without modifier keys, or by system-defined character compose sequences (that can involve dead keys or a compose key) can be input. You should call Fl::compose() in case any enhancements to this processing are done in the future. The interface has been designed to handle arbitrary UTF-8 encoded text.

The following methods are provided for character composition:

- Fl::compose()
- Fl::compose_reset()

Under Mac OS X, FLTK "previews" partially composed sequences.

1.10 Adding and Extending Widgets

This chapter describes how to add your own widgets or extend existing widgets in FLTK.

1.10.1 Subclassing

New widgets are created by *subclassing* an existing FLTK widget, typically Fl_Widget for controls and Fl_Group for composite widgets.

A control widget typically interacts with the user to receive and/or display a value of some sort.

A composite widget holds a list of child widgets and handles moving, sizing, showing, or hiding them as needed. Fl_Group is the main composite widget class in FLTK, and all of the other composite widgets (Fl_Pack, Fl_Scroll, Fl_Tabs, Fl_Tile, Fl_Window, Fl_Flex, Fl_Grid, etc.) are subclasses of it.

You can also subclass other existing widgets to provide a different look or user-interface. For example, the button widgets are all subclasses of Fl_Button since they all interact with the user via a mouse button click. The only difference is the code that draws the face of the button.

1.10.2 Making a Subclass of Fl_Widget

Your subclasses can directly descend from Fl_Widget or any subclass of Fl_Widget. Fl_Widget has only four virtual methods, and overriding some or all of these may be necessary.

1.10.3 The Constructor

The constructor should have the following arguments:

```
MyClass(int x, int y, int w, int h, const char *label = 0);
```

This will allow the class to be used in FLUID without problems.

The constructor must call the constructor for the base class and pass the same arguments:

```
MyClass::MyClass(int x, int y, int w, int h, const char *label)
  : Fl_Widget(x, y, w, h, label) {
  // do initialization stuff...
}
```

Fl_Widget's protected constructor sets `x()`, `y()`, `w()`, `h()`, and `label()` to the passed values and initializes the other instance variables to:

```
type(0);
box(FL_NO_BOX);
color(FL_BACKGROUND_COLOR);
selection_color(FL_BACKGROUND_COLOR);
labeltype(FL_NORMAL_LABEL);
labelstyle(FL_NORMAL_STYLE);
labelsize(FL_NORMAL_SIZE);
labelcolor(FL_FOREGROUND_COLOR);
align(FL_ALIGN_CENTER);
callback(default_callback,0);
flags(ACTIVE|VISIBLE);
image(0);
deimage(0);
```

1.10.4 Protected Methods of Fl_Widget

The following methods are provided for subclasses to use:

- clear_visible()
- damage()
- draw_box()
- draw_focus()
- draw_label()
- set_flag()
- set_visible()
- test_shortcut()
- type()

void Fl_Widget::damage(uchar mask)
void Fl_Widget::damage(uchar mask, int x, int y, int w, int h)
uchar Fl_Widget::damage()

The first form indicates that a partial update of the object is needed. The bits in mask are OR'd into damage(). Your `draw()` routine can examine these bits to limit what it is drawing. The public method Fl_Widget::redraw() simply does `Fl_Widget::damage(FL_DAMAGE_ALL)`, but the implementation of your widget can call the public `damage(n)`.

The second form indicates that a region is damaged. If only these calls are done in a window (no calls to `damage(n)`) then FLTK will clip to the union of all these calls before drawing anything. This can greatly speed up incremental displays. The mask bits are OR'd into `damage()` unless this is a Fl_Window widget.

The third form returns the bitwise-OR of all `damage(n)` calls done since the last `draw()`.

When redrawing your widgets you should look at the damage bits to see what parts of your widget need redrawing. The `handle()` method can then set individual damage bits to limit the amount of drawing that needs to be done:

```
int MyClass::handle(int event) {
  // ...
  if (change_to_part1) damage(1);
  if (change_to_part2) damage(2);
  if (change_to_part3) damage(4);
}

void MyClass::draw() {
  if (damage() & FL_DAMAGE_ALL) {
    // ... draw frame/box and other static stuff ...
  }

  if (damage() & (FL_DAMAGE_ALL | 1)) draw_part1();
  if (damage() & (FL_DAMAGE_ALL | 2)) draw_part2();
  if (damage() & (FL_DAMAGE_ALL | 4)) draw_part3();
}
```

Todo Clarify Fl_Window::damage(uchar) handling - seems confused/wrong? ORing value doesn't match setting behavior in Fl_Widget.H!

void Fl_Widget::draw_box() const
void Fl_Widget::draw_box(Fl_Boxtype t, Fl_Color c) const

The first form draws this widget's `box()`, using the dimensions of the widget. The second form uses `t` as the box type and `c` as the color for the box.

void Fl_Widget::draw_focus()
void Fl_Widget::draw_focus(Fl_Boxtype t, int x, int y, int w, int h) const

Draws a focus box inside the widget's bounding box. The second form allows you to specify a different bounding box.

void Fl_Widget::draw_label() const
void Fl_Widget::draw_label(int x, int y, int w, int h) const
void Fl_Widget::draw_label(int x, int y, int w, int h, Fl_Align align) const

The first form is the usual function for a `draw()` method to call to draw the widget's label. It does not draw the label if it is supposed to be outside the box (on the assumption that the enclosing group will draw those labels).

The second form uses the passed bounding box instead of the widget's bounding box. This is useful so "centered" labels are aligned with some feature, like a moving slider.

The third form draws the label anywhere. It acts as though `FL_ALIGN_INSIDE` has been forced on so the label will appear inside the passed bounding box. This is designed for parent groups to draw labels with.

void Fl_Widget::set_flag(int c)

Calling `set_flag(SHORTCUT_LABEL)` modifies the behavior of draw_label() so that '&' characters cause an underscore to be printed under the next letter.

void Fl_Widget::set_visible()
void Fl_Widget::clear_visible()

Fast inline versions of Fl_Widget::hide() and Fl_Widget::show(). These do not send the `FL_HIDE` and `FL_SHOW` events to the widget.

int Fl_Widget::test_shortcut()
static int Fl_Widget::test_shortcut(const char *s)

The first version tests Fl_Widget::label() against the current event (which should be a `FL_SHORTCUT` event). If the label contains a '&' character and the character after it matches the keypress, this returns true. This returns false if the `SHORTCUT_LABEL` flag is off, if the label is `NULL`, or does not have a '&' character in it, or if the keypress does not match the character.

The second version lets you do this test against an arbitrary string.

uchar Fl_Widget::type() const
void Fl_Widget::type(uchar t)

The property Fl_Widget::type() can return an arbitrary 8-bit identifier, and can be set with the protected method `type(uchar t)`. This value had to be provided for Forms compatibility, but you can use it for any purpose you want. Try to keep the value less than 100 to not interfere with reserved values.

FLTK does not use RTTI (Run Time Typing Information) to enhance portability. But this may change in the near future if RTTI becomes standard everywhere.

If you don't have RTTI you can use the clumsy FLTK mechanism, by having `type()` use a unique value. These unique values must be greater than the symbol `FL_RESERVED_TYPE` (which is 100) and less than `FL_WINDOW` (unless you make a subclass of Fl_Window). Look through the header files for `FL_RESERVED_TYPE` to find an unused number. If you make a subclass of Fl_Window you must use `FL_WINDOW + n` (where `n` must be in the range 1 to 7).

1.10.5 Handling Events

The virtual method Fl_Widget::handle(int event) is called to handle each event passed to the widget. It can:

- Change the state of the widget.
- Call Fl_Widget::redraw() if the widget needs to be redisplayed.
- Call Fl_Widget::damage(uchar c) if the widget needs a partial-update (assuming you provide support for this in your draw() method).
- Call Fl_Widget::do_callback() if a callback should be generated.
- Call Fl_Widget::handle() on child widgets.

Events are identified by the integer argument. Other information about the most recent event is stored in static locations and acquired by calling the Fl::event_*() methods. This information remains valid until another event is handled.

Here is a sample `handle()` method for a widget that acts as a pushbutton and also accepts the keystroke `'x'` to cause the callback:

```
int MyClass::handle(int event) {
  switch (event) {
    case FL_PUSH:
      highlight = 1;
      redraw();
      return 1;
    case FL_DRAG: {
      int t = Fl::event_inside(this);
      if (t != highlight) {
        highlight = t;
        redraw();
      }
    }
      return 1;
    case FL_RELEASE:
```

```
      if (highlight) {
        highlight = 0;
        redraw();
        do_callback();
        // never do anything after a callback, as the callback
        // may delete the widget!
      }
      return 1;
    case FL_SHORTCUT:
      if (Fl::event_key() == 'x') {
        do_callback();
        return 1;
      }
      return 0;
    default:
      return Fl_Widget::handle(event);
  }
}
```

You must return non-zero if your `handle()` method uses the event. If you return zero, the parent widget will try sending the event to another widget.

For debugging purposes, event numbers can be printed as their actual event names using the fl_eventnames[] array (std::map since FLTK 1.5), e.g.:

```
#include <FL/names.h>          // defines fl_eventnames[]
[..]
int MyClass::handle(int e) {
    printf("Event was %s (%d)\n", fl_eventnames[e], e);      // e.g. "Event was FL_PUSH (1)"
    [..]
```

1.10.6 Drawing the Widget

The `draw()` virtual method is called when FLTK wants you to redraw your widget. It will be called if and only if `damage()` is non-zero, and `damage()` will be cleared to zero after it returns. The `draw()` method should be declared protected so that it can't be called from non-drawing code.

The `damage()` value contains the bitwise-OR of all the `damage(n)` calls to this widget since it was last drawn. This can be used for minimal update, by only redrawing the parts whose bits are set. FLTK will turn on the `FL_DAMAGE_ALL` bit if it thinks the entire widget must be redrawn, e.g. for an expose event.

Expose events (and the damage(mask,x,y,w,h) function described above) will cause `draw()` to be called with FLTK's clipping turned on. You can greatly speed up redrawing in some cases by testing `fl_not_clipped(x,y,w,h)` or `fl_clip_box`() and skipping invisible parts.

Besides the protected methods described above, FLTK provides a large number of basic drawing functions, which are described in the chapter Drawing Things in FLTK.

1.10.7 Resizing the Widget

The `resize(x,y,w,h)` method is called when the widget is being resized or moved. The arguments are the new position, width, and height. `x()`, `y()`, `w()`, and `h()` still remain the old size. You must call `resize()` on your base class with the same arguments to get the widget size to actually change.

This should *not* call `redraw()`, at least if only the `x()` and `y()` change. This is because composite widgets like Fl_Scroll may have a more efficient way of drawing the new position.

1.10.8 Making a Composite Widget

A "composite" widget contains one or more "child" widgets. To make a composite widget you should subclass Fl_Group. It is possible to make a composite object that is not a subclass of Fl_Group, but you'll have to duplicate the code in Fl_Group anyways.

Instances of the child widgets may be included in the parent:

```
class MyClass : public Fl_Group {
  Fl_Button the_button;
  Fl_Slider the_slider;
  ...
};
```

The constructor has to initialize these instances. They are automatically added to the group, since the Fl_Group constructor does Fl_Group::begin(). *Don't forget to call Fl_Group::end() or use the Fl_End pseudo-class:*

```
MyClass::MyClass(int x, int y, int w, int h) :
  Fl_Group(x, y, w, h),
  the_button(x + 5, y + 5, 100, 20),
  the_slider(x, y + 50, w, 20)
{
  ...(you could add dynamically created child widgets here)...
  end(); // don't forget to do this!
}
```

The child widgets need callbacks. These will be called with a pointer to the children, but the widget itself may be found in the `parent()` pointer of the child. Usually these callbacks can be static private methods, with a matching private method:

```
void MyClass::static_slider_cb(Fl_Widget* v, void *) { // static method
  ((MyClass*)(v->parent())->slider_cb();
}
void MyClass::slider_cb() { // normal method
  use(the_slider->value());
}
```

If you make the `handle()` method, you can quickly pass all the events to the children using the Fl_Group::handle() method. You don't need to override `handle()` if your composite widget does nothing other than pass events to the children:

```
int MyClass::handle(int event) {
  if (Fl_Group::handle(event)) return 1;
  ... handle events that children don't want ...
}
```

If you override `draw()` you need to draw all the children. If `redraw()` or `damage()` is called on a child, `damage(FL_DAMAGE_CHILD)` is done to the group, so this bit of `damage()` can be used to indicate that a child needs to be drawn. It is fastest if you avoid drawing anything else in this case:

```
void MyClass::draw() {
  Fl_Widget *const*a = array();
  if (damage() == FL_DAMAGE_CHILD) { // only redraw some children
    for (int i = children(); i--; a++) {
      update_child(**a);
    }
  } else { // total redraw
    // ... draw background graphics ...
    // now draw all the children atop the background:
    for (int i = children(); i--; a++) {
      draw_child(**a);
      draw_outside_label(**a); // you may not need to do this
    }
  }
}
```

Fl_Group provides some protected methods to make drawing easier:

- draw_child()
- draw_children()
- draw_outside_label()
- update_child()

void Fl_Group::draw_child(Fl_Widget &widget) const

This will force the child's `damage()` bits all to one and call `draw()` on it, then clear the `damage()`. You should call this on all children if a total redraw of your widget is requested, or if you draw something (like a background box) that damages the child. Nothing is done if the child is not `visible()` or if it is clipped.

void Fl_Group::draw_children()

A convenience function that draws all children of the group. This is useful if you derived a widget from Fl_Group and want to draw a special border or background. You can call `draw_children()` from the derived `draw()` method after drawing the box, border, or background.

void Fl_Group::draw_outside_label(const Fl_Widget &widget) const

Draw the labels that are *not* drawn by draw_label(). If you want more control over the label positions you might want to call `child->draw_label(x,y,w,h,a)`.

void Fl_Group::update_child(Fl_Widget& widget) const

Draws the child only if its `damage()` is non-zero. You should call this on all the children if your own damage is equal to `FL_DAMAGE_CHILD`. Nothing is done if the child is not `visible()` or if it is clipped.

1.10.9 Cut and Paste Support

FLTK provides routines to cut and paste UTF-8 encoded text between applications:

- Fl::copy()
- Fl::paste()
- Fl::selection()
- Fl::selection_owner()

It is also possible to copy and paste image data between applications:

- Fl_Copy_Surface
- Fl::clipboard_contains()
- Fl::paste()

It may be possible to cut/paste other kinds of data by using Fl::add_handler(). Note that handling events beyond those provided by FLTK may be operating system specific. See Operating System Issues for more details.

1.10.10 Drag And Drop Support

FLTK provides routines to drag and drop UTF-8 encoded text between applications:

Drag'n'drop operations are initiated by copying data to the clipboard and calling the function Fl::dnd().

Drop attempts are handled via the following events, already described under Drag and Drop Events in a previous chapter:

- `FL_DND_ENTER`
- `FL_DND_DRAG`
- `FL_DND_LEAVE`
- `FL_DND_RELEASE`
- `FL_PASTE`

1.10.11 Making a subclass of Fl_Window

You may want your widget to be a subclass of Fl_Window, Fl_Double_Window, or Fl_Gl_Window. This can be useful if your widget wants to occupy an entire window, and can also be used to take advantage of system-provided clipping, or to work with a library that expects a system window ID to indicate where to draw.

Subclassing Fl_Window is almost exactly like subclassing Fl_Group, and in fact you can easily switch a subclass back and forth. Watch out for the following differences:

1. Fl_Window is a subclass of Fl_Group so *make sure your constructor calls* `end()` unless you actually want children added to your window.
2. When handling events and drawing, the upper-left corner is at 0,0, not `x(),y()` as in other Fl_Widget's. For instance, to draw a box around the widget, call `draw_box(0,0,w(),h())`, rather than `draw_box(x(),y(),w(),h())`.

You may also want to subclass Fl_Window in order to get access to different visuals or to change other attributes of the windows. See the Operating System Issues chapter for more information.

1.11 Using OpenGL

This chapter discusses using FLTK for your OpenGL applications.

1.11.1 Using OpenGL in FLTK

The easiest way to make an OpenGL display is to subclass Fl_Gl_Window. Your subclass must implement a `draw()` method which uses OpenGL calls to draw the display. Your main program should call `redraw()` when the display needs to change, and (somewhat later) FLTK will call `draw()`.

With a bit of care you can also use OpenGL to draw into normal FLTK windows (see Using OpenGL in Normal FLTK Windows below). This allows you to use Gouraud shading for drawing your widgets. To do this you use the gl_start() and gl_finish() functions around your OpenGL code.

You must include FLTK's `<FL/gl.h>` header file. It will include the file `<GL/gl.h>` (on macOS: `<OpenGL/gl.h>`), define some extra drawing functions provided by FLTK, and include the `<windows.h>` header file needed by Windows applications.

Some simple coding rules (see OpenGL and support of HighDPI displays) allow to write cross-platform code that will support OpenGL run on HighDPI displays (including the 'retina' displays of Apple hardware).

1.11.2 Making a Subclass of Fl_Gl_Window

To make a subclass of Fl_Gl_Window, you must provide:

- A class definition.
- A `draw()` method.
- A `handle()` method if you need to receive input from the user.

If your subclass provides static controls in the window, they must be redrawn whenever the `FL_DAMAGE_ALL` bit is set in the value returned by `damage()`.

1.11.2.1 Defining the Subclass

To define the subclass you just subclass the Fl_Gl_Window class:

```
class MyWindow : public Fl_Gl_Window {
  void draw();
  int handle(int);

public:
  MyWindow(int X, int Y, int W, int H, const char *L)
    : Fl_Gl_Window(X, Y, W, H, L) {}
};
```

The `draw()` and `handle()` methods are described below. Like any widget, you can include additional private and public data in your class (such as scene graph information, etc.)

1.11.2.2 The draw() Method

The `draw()` method is where you actually do your OpenGL drawing:

```
void MyWindow::draw() {
  if (!valid()) {
    ... set up projection, viewport, etc ...
    ... window size is in w() and h().
    ... valid() is turned on by FLTK after draw() returns
  }
  ... draw ...
}
```

1.11.2.3 The handle() Method

The `handle()` method handles mouse and keyboard events for the window:

```
int MyWindow::handle(int event) {
  switch(event) {
  case FL_PUSH:
    ... mouse down event ...
    ... position in Fl::event_x() and Fl::event_y()
    return 1;
  case FL_DRAG:
    ... mouse moved while down event ...
    return 1;
  case FL_RELEASE:
    ... mouse up event ...
    return 1;
  case FL_FOCUS :
  case FL_UNFOCUS :
    ... Return 1 if you want keyboard events, 0 otherwise
    return 1;
  case FL_KEYBOARD:
    ... keypress, key is in Fl::event_key(), ascii in Fl::event_text()
    ... Return 1 if you understand/use the keyboard event, 0 otherwise...
    return 1;
  case FL_SHORTCUT:
```

```
    ... shortcut, key is in Fl::event_key(), ascii in Fl::event_text()
    ... Return 1 if you understand/use the shortcut event, 0 otherwise...
    return 1;
  default:
    // pass other events to the base class...
    return Fl_Gl_Window::handle(event);
  }
}
```

When `handle()` is called, the OpenGL context is not set up! If your display changes, you should call `redraw()` and let `draw()` do the work. Don't call any OpenGL drawing functions from inside `handle()` !

You can call *some* OpenGL stuff like hit detection and texture loading functions by doing:

```
case FL_PUSH:
  make_current();      // make OpenGL context current
  if (!valid()) {

    ... set up projection exactly the same as draw ...

    valid(1);          // stop it from doing this next time
  }
  ... ok to call NON-DRAWING OpenGL code here, such as hit
  detection, loading textures, etc...
```

Your main program can now create one of your windows by doing `new MyWindow(...)`.

You can also use your new window class in FLUID by:

1. Putting your class definition in a `MyWindow.H` file.
2. Creating a Fl_Box widget in FLUID.
3. In the widget panel fill in the "class" field with `MyWindow`. This will make FLUID produce constructors for your new class.
4. In the "Extra Code" field put `#include "MyWindow.H"`, so that the FLUID output file will compile.

You must put `glwindow->show()` in your main code after calling `show()` on the window containing the Open↩ GL window.

1.11.3 OpenGL and support of HighDPI displays

HighDPI displays (including the so-called 'retina' displays of Apple hardware) are supported by FLTK in such a way that 1 unit of an FLTK quantity (say, the value given by Fl_Gl_Window::w()) corresponds to more than 1 pixel on the display. Consequently, when a program specifies the width and height of the OpenGL viewport, it is necessary to use an API that returns quantities expressed in pixels. That can be done as follows:

```
Fl_Gl_Window *glw = ...;
glViewport(0, 0, glw->pixel_w(), glw->pixel_h());
```

which makes use of the Fl_Gl_Window::pixel_w() and Fl_Gl_Window::pixel_h() methods giving the size in pixels of an Fl_Gl_Window that is potentially mapped to a HighDPI display.

Method Fl_Gl_Window::pixels_per_unit() can be useful when the OpenGL code depends on the pixel dimension of the GL scene. This occurs, e.g., if a window's handle() method uses Fl::event_x() and Fl::event_y() whose returned values should be multiplied by Fl_Gl_Window::pixels_per_unit() to obtain the adequate pixel units. This method may also be useful, for example, to adjust the width of a line in a high resolution GL scene.

1.11.4 Using OpenGL in Normal FLTK Windows

Note

Drawing both with OpenGL and Quartz in a normal FLTK window is not possible with the macOS platform. This technique is therefore not useful under macOS because it permits nothing more than what is possible with class Fl_Gl_Window.

You can put OpenGL code into the `draw()` method, as described in Drawing the Widget in the previous chapter, or into the code for a boxtype or other places with some care.

Most importantly, before you show *any* windows, including those that don't have OpenGL drawing, you **must** initialize FLTK so that it knows it is going to use OpenGL. You may use any of the symbols described for `Fl_Gl_Window::mode()` to describe how you intend to use OpenGL:

```
Fl::gl_visual(FL_RGB);
```

You can then put OpenGL drawing code anywhere you can draw normally by surrounding it with gl_start() and gl_finish() to set up, and later release, an OpenGL context with an orthographic projection so that 0,0 is the lower-left corner of the window and each pixel is one unit. The current clipping is reproduced with OpenGL `glScissor()` commands. These functions also synchronize the OpenGL graphics stream with the drawing done by other X, Windows, or FLTK functions.

```
gl_start();
... put your OpenGL code here ...
gl_finish();
```

The same context is reused each time. If your code changes the projection transformation or anything else you should use `glPushMatrix()` and `glPopMatrix()` functions to put the state back before calling `gl_finish()`.

You may want to use `Fl_Window::current()->h()` to get the drawable height so that you can flip the Y coordinates.

Unfortunately, there are a bunch of limitations you must adhere to for maximum portability:

- You must choose a default visual with Fl::gl_visual().
- You cannot pass `FL_DOUBLE` to Fl::gl_visual().
- You cannot use Fl_Double_Window or Fl_Overlay_Window.

Do *not* call `gl_start()` or `gl_finish()` when drawing into an Fl_Gl_Window !

1.11.5 Using FLTK widgets in OpenGL Windows

FLTK widgets can be added to `Fl_Gl_Window`s just as they would be added to `Fl_Window`s. They are rendered as an overlay over the user defined OpenGL graphics using 'fl_..' graphics calls that are implemented in GL.

`Fl_Gl_Window` does not add subsequent widgets as children by default as `Fl_Window` does. Call `myGlWindow->begin()` after creating the GL window to automatically add following widgets. Remember to call `myGlWindow->end()`.

```
class My_Gl_Window : public Fl_Gl_Window {
...
  void draw();
...
};

...
```

```
myGlWindow = new My_Gl_Window(0, 0, 500, 500);
myGlWindow->begin();
myButton = new Fl_Button(10, 10, 120, 24, "Hello!");
myGlWindow->end();
...

void My_Gl_Window::draw() {
  // ... user GL drawing code
  Fl_Gl_Window::draw(); // Draw FLTK child widgets.
}
```

Users can draw into the overlay by using GL graphics calls as well as all `fl_...` graphics calls from the "Drawing Fast Shapes" section.

```
void My_Gl_Window::draw() {
  // ... user GL drawing code
  Fl_Gl_Window::draw_begin(); // Set up 1:1 projection
  Fl_Window::draw();          // Draw FLTK children
  fl_color(FL_RED);
  fl_rect(10, 10, 100, 100);
  Fl_Gl_Window::draw_end();   // Restore GL state
}
```

Widgets can be drawn with transparencies by assigning an alpha value to a colormap entry and using that color in the widget.

```
Fl::set_color(FL_FREE_COLOR, 255, 255, 0, 127); // 50% transparent yellow
myGlWindow = new My_Gl_Window(0, 0, 500, 500);
myGlWindow->begin();
myButton = new Fl_Button(10, 10, 120, 24, "Hello!");
myButton->box(FL_BORDER_BOX);
myButton->color(FL_FREE_COLOR);
myGlWindow->end();
```

Transparencies can also be set directly when drawing. This can be used to create custom box types and RGB overlay drawings with an alpha channel.

```
fl_color(0, 255, 0, 127); // 50% transparent green
fl_rectf(10, 10, 100, 100);
fl_color(FL_RED); // back to opaque red
fl_rect(20, 20, 80, 80);
```

1.11.6 OpenGL Drawing Functions

FLTK provides some useful OpenGL drawing functions. They can be freely mixed with any OpenGL calls, and are defined by including `<FL/gl.h>` which you should include instead of the OpenGL header `<GL/gl.h>`.

void gl_color(Fl_Color)

> Sets the current OpenGL color to a FLTK color. *For color-index modes it will use `fl_xpixel(c)`, which is only right if this window uses the default colormap!*

void gl_rect(int x, int y, int w, int h)
void gl_rectf(int x, int y, int w, int h)

> Outlines or fills a rectangle with the current color. If Fl_Gl_Window::ortho() has been called, then the rectangle will exactly fill the pixel rectangle passed.

void gl_font(Fl_Font fontid, int size)

Sets the current OpenGL font to the same font you get by calling fl_font().

int gl_height()
int gl_descent()
float gl_width(const char *s)
float gl_width(const char *s, int n)
float gl_width(uchar c)

Returns information about the current OpenGL font.

void gl_draw(const char *s)
void gl_draw(const char *s, int n)

Draws a nul-terminated string or an array of `n` characters in the current OpenGL font at the current raster position.

void gl_draw(const char *s, int x, int y)
void gl_draw(const char *s, int n, int x, int y)
void gl_draw(const char *s, float x, float y)
void gl_draw(const char *s, int n, float x, float y)

Draws a nul-terminated string or an array of `n` characters in the current OpenGL font at the given position.

void gl_draw(const char *s, int x, int y, int w, int h, Fl_Align)

Draws a string formatted into a box, with newlines and tabs expanded, other control characters changed to ^X, and aligned with the edges or center. Exactly the same output as fl_draw().

1.11.7 Speeding up OpenGL

Performance of Fl_Gl_Window may be improved on some types of OpenGL implementations, in particular MESA and other software emulators, by setting the `GL_SWAP_TYPE` environment variable. This variable declares what is in the backbuffer after you do a swapbuffers.

- `setenv GL_SWAP_TYPE COPY`

  This indicates that the back buffer is copied to the front buffer, and still contains its old data. This is true of many hardware implementations. Setting this will speed up emulation of overlays, and widgets that can do partial update can take advantage of this as `damage()` will not be cleared to -1.

- `setenv GL_SWAP_TYPE NODAMAGE`

  This indicates that nothing changes the back buffer except drawing into it. This is true of MESA and Win32 software emulation and perhaps some hardware emulation on systems with lots of memory.

- All other values for `GL_SWAP_TYPE`, and not setting the variable, cause FLTK to assume that the back buffer must be completely redrawn after a swap.

This is easily tested by running the gl_overlay demo program and seeing if the display is correct when you drag another window over it or if you drag the window off the screen and back on. You have to exit and run the program again for it to see any changes to the environment variable.

1.11.8 Using OpenGL Optimizer with FLTK

`OpenGL Optimizer` is a scene graph toolkit for OpenGL available from Silicon Graphics for IRIX and Microsoft Windows. It allows you to view large scenes without writing a lot of OpenGL code.

OptimizerWindow Class Definition

To use `OpenGL Optimizer` with FLTK you'll need to create a subclass of Fl_Gl_Widget that includes several state variables:

```
class OptimizerWindow : public Fl_Gl_Window {
  csContext *context_; // Initialized to 0 and set by draw()...
  csDrawAction *draw_action_; // Draw action...
  csGroup *scene_; // Scene to draw...
  csCamara *camera_; // Viewport for scene...

  void draw();

public:
  OptimizerWindow(int X, int Y, int W, int H, const char *L)
    : Fl_Gl_Window(X, Y, W, H, L) {
      context_ = (csContext *)0;
      draw_action_ = (csDrawAction *)0;
      scene_ = (csGroup *)0;
      camera_ = (csCamera *)0;
    }

  void scene(csGroup *g) { scene_ = g; redraw(); }

  void camera(csCamera *c) {
    camera_ = c;
    if (context_) {
      draw_action_->setCamera(camera_);
      camera_->draw(draw_action_);
      redraw();
    }
  }
};
```

The camera() Method

The `camera()` method sets the camera (projection and viewpoint) to use when drawing the scene. The scene is redrawn after this call.

The draw() Method

The `draw()` method performs the needed initialization and does the actual drawing:

```
void OptimizerWindow::draw() {
  if (!context_) {
    // This is the first time we've been asked to draw; create the
    // Optimizer context for the scene...

#ifdef _WIN32
    context_ = new csContext((HDC)fl_getHDC());
    context_->ref();
    context_->makeCurrent((HDC)fl_getHDC());
#else
    context_ = new csContext(fl_display, fl_visual);
    context_->ref();
    context_->makeCurrent(fl_display, fl_window);
#endif // _WIN32

    ... perform other context setup as desired ...

    // Then create the draw action to handle drawing things...

    draw_action_ = new csDrawAction;
    if (camera_) {
      draw_action_->setCamera(camera_);
      camera_->draw(draw_action_);
    }
  } else {
#ifdef _WIN32
    context_->makeCurrent((HDC)fl_getHDC());
#else
    context_->makeCurrent(fl_display, fl_window);
#endif // _WIN32
  }

  if (!valid()) {
    // Update the viewport for this context...
    context_->setViewport(0, 0, w(), h());
  }

  // Clear the window...
  context_->clear(csContext::COLOR_CLEAR | csContext::DEPTH_CLEAR,
                  0.0f,         // Red
                  0.0f,         // Green
                  0.0f,         // Blue
                  1.0f);        // Alpha

  // Then draw the scene (if any)...
  if (scene_)
    draw_action_->apply(scene_);
}
```

The scene() Method

The `scene()` method sets the scene to be drawn. The scene is a collection of 3D objects in a `csGroup`. The scene is redrawn after this call.

1.11.9 Using OpenGL 3.0 (or higher versions)

The examples subdirectory contains OpenGL3test.cxx, a toy program showing how to use OpenGL 3.0 (or higher versions) with FLTK in a cross-platform fashion. It contains also OpenGL3-glut-test.cxx which shows how to use FLTK's GLUT compatibility and OpenGL 3.

To access OpenGL 3.0 (or higher versions), use the `FL_OPENGL3` flag when calling Fl_Gl_Window::mode(int a) or glutInitDisplayMode().

On the Windows and Linux platforms, FLTK creates contexts implementing the highest OpenGL version supported by the hardware. Such contexts may also be compatible with lower OpenGL versions. Access to functions from OpenGL versions above 1.1 requires to load function pointers at runtime on these platforms. FLTK recommends to use the GLEW library to perform this. It is therefore necessary to install the GLEW library (see below).

On the macOS platform, MacOS 10.7 or above is required; GLEW is possible but not necessary. FLTK creates contexts for OpenGL versions 1 and 2 without the FL_OPENGL3 flag and for OpenGL versions 3.2 and above (**but not below**) with it.

GLEW installation (Linux and Windows platforms)

FLTK needs a header file, `GL/glew.h`, and a library, `libGLEW.*` or equivalent, to support OpenGL 3 and above.
These can be obtained for most Linux distributions by installing package libglew-dev.
For the Windows platform :

- the header and a Visual Studio static library (glew32.lib) can be downloaded from http://glew.sourceforge.net/ ;
- a MinGW-style static library (libglew32.a) can be built from source (same web site) with the make command. Alternatively, pre-built files are available for these architectures :
  - x86: download files glew.h and libglew32.a;
  - x86_64: install GLEW as an MSYS2 package with command :
    `pacman -S mingw-w64-x86_64-glew`

Source-level changes for OpenGL 3:

- Put this in all OpenGL-using source files (instead of, or before if needed, #include <FL/gl.h>, and before #include <FL/glut.h> if you use GLUT):
```
#if defined(__APPLE__)
#  include <OpenGL/gl3.h> // defines OpenGL 3.0+ functions
#else
#  if defined(_WIN32)
#    define GLEW_STATIC 1
#  endif
#  include <GL/glew.h>
#endif
```
- Add the `FL_OPENGL3` flag when calling Fl_Gl_Window::mode(int a) or glutInitDisplayMode().
- Put this in the `handle(int event)` member function of the first to be created among your Fl_Gl_Window-derived classes:
```
#ifndef __APPLE__
    static int first = 1;
    if (first && event == FL_SHOW && shown()) {
      first = 0;
      make_current();
      glewInit(); // defines pters to functions of OpenGL V 1.2 and above
    }
#endif
```
- Alternatively, if you use GLUT, put
```
#ifndef __APPLE__
  glewInit(); // defines pters to functions of OpenGL V 1.2 and above
#endif
```
  after the first glutCreateWindow() call.

If GLEW is installed on the Mac OS development platform, it is possible to use the same code for all platforms, with one exception: put
```
#ifdef __APPLE__
glewExperimental = GL_TRUE;
#endif
```
before the glewInit() call.

Testing for success of the glewInit() call

Testing whether the glewInit() call is successful is to be done as follows:
```
#include <FL/platform.H> // defines FLTK_USE_WAYLAND under the Wayland platform
#include <FL/Fl.H> // for Fl::warning()
#ifndef __APPLE__
#  if defined(_WIN32)
#    define GLEW_STATIC 1
#  endif
#  include <GL/glew.h>

  GLenum err = glewInit(); // defines pters to functions of OpenGL V 1.2 and above
#  ifdef FLTK_USE_WAYLAND
      // glewInit returns GLEW_ERROR_NO_GLX_DISPLAY with Wayland
      if (fl_wl_display() && err == GLEW_ERROR_NO_GLX_DISPLAY) err = GLEW_OK;
#  endif
  if (err != GLEW_OK) Fl::warning("glewInit() failed returning %u", err);
#endif // ! __APPLE__
```

Changes in the build process

Link with libGLEW.so (with X11 or Wayland), libglew32.a (with MinGW) or glew32.lib (with MS Visual Studio); no change is needed on the Mac OS platform.

1.12 FLTK Runtime Options

In this chapter, we will cover how to access and alter settings for applications created using FLTK, both as an administrator and as a regular user.

Subchapters:

- Runtime Options
- Obtaining Current Settings
- Administrative Tool
- List of Options

1.12.1 Runtime Options

FLTK keeps track of various aspects of the user interface in a system-wide database. Users have the ability to set their own preferences and override default or system settings. For instance, FLTK will display a dotted rectangle around the widget with current focus. This might not be desirable for users who do not use keyboard navigation and do not need the rectangle. This can be turned off by setting the OPTION_VISIBLE_FOCUS option to 'off' for that user, which will disable the focus rectangle in all FLTK-based applications.

1.12.2 Obtaining Current Settings

Options are kept in preference files using the signature `Fl_Preferences::CORE_SYSTEM, "fltk.org", "fltk"` for system-wide settings and `Fl_Preferences::CORE_USER, "fltk.org", "fltk"` for individual users. They can be accessed by using the function `bool Fl::option(Fl_Option opt)`. If an application needs to temporarily override user or system settings, it can use the function `void option(Fl_Option opt, bool val)`.

To make changes to options permanently, FLTK provides an administrative tool called fltk-options.

1.12.3 Administrative Tool

`fltk-options` is a hybrid app that is part of FLTK and can be installed on the target system. It includes an up-to-date man page.

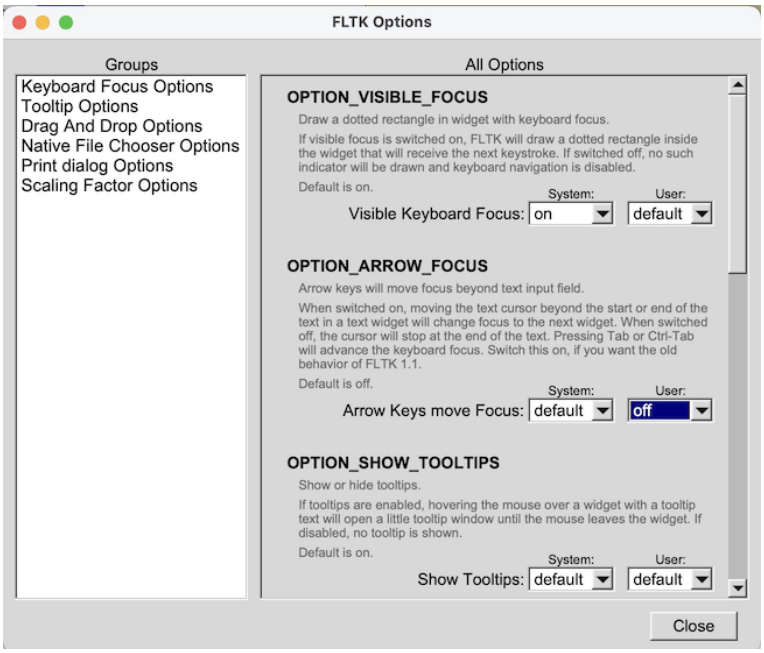


Figure 1.29 fltk-options Application

When fltk-options is called without any command-line arguments, it opens in interactive mode and provides a user interface to view and alter all system and current user options.

Starting the tool from a shell, the command-line options `-S` and `-U` can be used to display or change system or user options. On MS-Windows, `fltk-options` is also available as `fltk-options-cmd.exe`.

1.12.4 List of Options

Calling `fltk-options --help` gives a list of all available commands, and options and their values. `fltk-options --help OPTION` prints more detailed information for OPTION if available. In interactive mode, tooltips provide this additional information.

A full list of options can be found in the manual at Fl::Fl_Option.

1.13 Advanced FLTK

This chapter explains advanced programming and design topics that will help you to get the most out of FLTK.

1.13.1 Multithreading

FLTK can be used to implement a GUI for a multithreaded application but, as with multithreaded programming generally, there are some concepts and caveats that must be kept in mind.

Key amongst these is that, for many of the target platforms on which FLTK is supported, only the `main()` thread of the process is permitted to handle system events, create or destroy windows and open or close windows. Further, only the `main()` thread of the process can safely write to the display.

To support this in a portable way, all FLTK `draw()` methods are executed in the `main()` thread. A worker thread may update the state of an existing widget, but it may not do any rendering directly, nor create or destroy a window. (**NOTE:** A special case exists for Fl_Gl_Window where it can, with suitable precautions, be possible to safely render to an existing GL context from a worker thread.)

Creating portable threads

We do not provide a threading interface as part of the library. A simple example showing how threads can be implemented, for all supported platforms, can be found in `test/threads.h` and `test/threads.cxx`.

FLTK has been used with a variety of thread interfaces, so if the simple example shown in `test/threads.cxx` does not cover your needs, you might want to select a third-party library that provides the features you require.

1.13.2 FLTK multithread locking - Fl::lock() and Fl::unlock()

In a multithreaded program, drawing of widgets (in the `main()` thread) happens asynchronously to widgets being updated by worker threads, so no drawing can occur safely whilst a widget is being modified (and no widget should be modified whilst drawing is in progress).

FLTK supports multithreaded applications using a locking mechanism internally. This allows a worker thread to lock the rendering context, preventing any drawing from taking place, whilst it changes the value of its widget.

Note

> The converse is also true; whilst a worker thread holds the lock, the `main()` thread may not be able to process any drawing requests, nor service any events. So a worker thread that holds the FLTK lock **must** contrive to do so for the shortest time possible or it could impair operation of the application.

The lock operates broadly as follows.

Using the FLTK library, the `main()` thread holds the lock whenever it is processing events or redrawing the display. It acquires (locks) and releases (unlocks) the FLTK lock automatically and no "user intervention" is required. Indeed, a function that runs in the context of the `main()` thread ideally should **not** acquire / release the FLTK lock explicitly. (Though note that the lock calls are recursive, so calling Fl::lock() from a thread that already holds the lock, including the `main()` thread, is benign. The only constraint is that every call to Fl::lock() **must** be balanced by a corresponding call to Fl::unlock() to ensure the lock count is preserved.)

The `main()` thread **must** call Fl::lock() **once** before any windows are shown, to enable the internal lock (it is "off" by default since it is not useful in single-threaded applications) but thereafter the `main()` thread lock is managed by the library internally.

A worker thread, when it wants to alter the value of a widget, can acquire the lock using Fl::lock(), update the widget, then release the lock using Fl::unlock(). Acquiring the lock ensures that the worker thread can update the widget, without any risk that the `main()` thread will attempt to redraw the widget whilst it is being updated.

Note that acquiring the lock is a blocking action; the worker thread will stall for as long as it takes to acquire the lock. If the `main()` thread is engaged in some complex drawing operation this may block the worker thread for a long time, effectively serializing what ought to be parallel operations. (This frequently comes as a surprise to coders less familiar with multithreaded programming issues; see the discussion of "lockless programming" later for strategies for managing this.)

To incorporate the locking mechanism in the library, FLTK must be compiled with FLTK_USE_PTHREADS=On set during the build process. This is the default since version 1.3. IDE-based versions of FLTK are automatically compiled with the locking mechanism incorporated if possible.

1.13.3 Simple multithreaded examples using Fl::lock

In `main()`, call Fl::lock() once before Fl::run() or Fl::wait() to enable the lock and start the runtime multithreading support for your program. All callbacks and derived functions like `handle()` and `draw()` will now be properly locked.

This might look something like this:

```
int main(int argc, char **argv) {
  /* Create your windows and widgets here */

  Fl::lock(); /* "start" the FLTK lock mechanism */

  /* show your window */
  main_win->show(argc, argv);

  /* start your worker threads */
  ... start threads ...

  /* Run the FLTK main loop */
  int result = Fl::run();

  /* terminate any pending worker threads */
  ... stop threads ...

  return result;
}
```

You can start as many threads as you like. From within a thread (other than the `main()` thread) FLTK calls must be wrapped with calls to Fl::lock() and Fl::unlock():

```
void my_thread(void) {
  while (thread_still_running) {
    /* do thread work */
    ...
    /* compute new values for widgets */
    ...

    Fl::lock();       // acquire the lock
    my_widget->update(values);
    Fl::unlock();     // release the lock; allow other threads to access FLTK again
    Fl::awake();      // use Fl::awake() to signal main thread to refresh the GUI
  }
}
```

Note

To trigger a refresh of the GUI from a worker thread, the worker code should call Fl::awake()

Using Fl::awake callback messages

You can also request that the `main()` thread call a function on behalf of the worker thread by using Fl::awake(Fl_Awake_Handler cb, void* userdata).

The `main()` thread will execute the callback "as soon as possible" when next processing the pending events. This can be used by a worker thread to perform operations (for example showing or hiding windows) that are prohibited in a worker thread.

```
void do_something_cb(void *userdata) {
  // Will run in the context of the main thread
  ... do_stuff ...
}

// running in worker thread
void *data;                       // "data" is a pointer to your user data
Fl::awake(do_something_cb, data); // call to execute cb in main thread
```

Note

The `main()` thread will execute the Fl_Awake_Handler callback `do_something_cb` asynchronously to the worker thread, at some short but indeterminate time after the worker thread registers the request. When it executes the Fl_Awake_Handler callback, the `main()` thread will use the contents of `*userdata` **at the time of execution**, not necessarily the contents that `*userdata` had at the time that the worker thread posted the callback request. The worker thread should therefore contrive **not** to alter the contents of `*userdata` once it posts the callback, since the worker thread does not know when the `main()` thread will consume that data. It is often useful that `userdata` point to a struct, one member of which the `main()` thread can modify to indicate that it has consumed the data, thereby allowing the worker thread to re-use or update `userdata`.

Warning

The Fl::awake(void* message) call has been deprecated because the API was not sufficient to ensure the deliver of all message or the order of messages. The cal still exists for back compatibility, but should be repleaced with a call to Fl::awake(), Fl::awake(handler, user_data), or Fl::awake_once(handler, user_data).

1.13.4 FLTK multithreaded "lockless programming"

The simple multithreaded examples shown above, using the FLTK lock, work well for many cases where multiple threads are required. However, when that model is extended to more complex programs, it often produces results that the developer did not anticipate.

A typical case might go something like this. A developer creates a program to process a huge data set. The program has a `main()` thread and 7 worker threads and is targeted to run on an 8-core computer. When it runs, the program divides the data between the 7 worker threads, and as they process their share of the data, each thread updates its portion of the GUI with the results, locking and unlocking as they do so.

But when this program runs, it is much slower than expected and the developer finds that only one of the eight CPU cores seems to be utilised, despite there being 8 threads in the program. What happened?

The threads in the program all run as expected, but they end up being serialized (that is, not able to run in parallel) because they all depend on the single FLTK lock. Acquiring (and releasing) that lock has an associated cost, and is a **blocking** action if the lock is already held by any other worker thread or by the `main()` thread.

If the worker threads are acquiring the lock "too often", then the lock will **always** be held **somewhere** and every attempt by any other thread (even `main()`) to lock will cause that other thread (including `main()`) to block. And blocking `main()` also blocks event handling, display refresh...

As a result, only one thread will be running at any given time, and the multithreaded program is effectively reduced to being a (complicated and somewhat less efficient) single thread program.

A "solution" is for the worker threads to lock "less often", such that they do not block each other or the `main()` thread. But judging what constitutes locking "too often" for any given configuration, and hence will block, is a very tricky question. What works well on one machine, with a given graphics card and CPU configuration may behave very differently on another target machine.

There are "interesting" variations on this theme, too: for example it is possible that a "faulty" multithreaded program such as described above will work adequately on a single-core machine (where all threads are inherently serialized anyway and so are less likely to block each other) but then stall or even deadlock in unexpected ways on a multicore machine when the threads do interfere with each other. (I have seen this - it really happens.)

The "better" solution is to avoid using the FLTK lock so far as possible. Instead, the code should be designed so that the worker threads do not update the GUI themselves and therefore never need to acquire the FLTK lock. This would be FLTK multithreaded "lockless programming".

There are a number of ways this can be achieved (or at least approximated) in practice but the most direct approach is for the worker threads to make use of the Fl::awake(Fl_Awake_Handler cb, void* userdata) method so that GUI updates can all run in the context of the `main()` thread, alleviating the need for the worker thread to ever lock. The onus is then on the worker threads to manage the `userdata` so that it is delivered safely to the `main()` thread, but there are many ways that can be done.

Note

Using Fl::awake is not, strictly speaking, entirely "lockless" since the awake handler mechanism incorporates resource locking internally to protect the queue of pending awake messages. These resource locks are held transiently and generally do not trigger the pathological blocking issues described here.

However, aside from using Fl::awake, there are many other ways that a "lockless" design can be implemented, including message passing, various forms of IPC, etc.

If you need high performing multithreaded programming, then take some time to study the options and understand the advantages and disadvantages of each; we can't even begin to scratch the surface of this huge topic here!

And of course occasional, sparse, use of the FLTK lock from worker threads will do no harm; it is "excessive" locking (whatever that might be) that triggers the failing behaviour.

It is always a Good Idea to update the GUI at the lowest rate that is acceptable when processing bulk data (or indeed, in all cases!) Updating at a few frames per second is probably adequate for providing feedback during a long calculation. At the upper limit, anything faster than the frame rate of your monitor and the updates will never even be displayed; why waste CPU computing pixels that you will never show?

1.13.5 FLTK multithreaded Constraints

FLTK supports multiple platforms, some of which allow only the `main()` thread to handle system events and open or close windows. The safe thing to do is to adhere to the following rules for threads on all operating systems:

- Don't `show()` or `hide()` anything that contains Fl_Window based widgets from a worker thread. This includes any windows, dialogs, file choosers, subwindows or widgets using Fl_Gl_Window. Note that this constraint also applies to non-window widgets that have tooltips, since the tooltip will contain a Fl_Window object. The safe and portable approach is **never** to call `show()` or `hide()` on any widget from the context of a worker thread. Instead you can use the Fl_Awake_Handler variant of Fl::awake() to request the `main()` thread to create, destroy, show or hide the widget on behalf of the worker thread.

- Don't call Fl::run(), Fl::wait(), Fl::flush(), Fl::check() or any related methods that will handle system messages from a worker thread

- Don't intermix use of Fl::awake(Fl_Awake_Handler cb, void∗ userdata) and Fl::awake(void∗ message) calls in the same program as they may interact unpredictably on some platforms; choose one or other style of Fl::awake(<thing>) mechanism and use that. (Intermixing calls to Fl::awake() should be safe with either however.)

- Starting with FLTK 1.4, it's possible to start (or cancel) a timer from a worker thread under the condition that the call to Fl::add_timeout (or Fl::remove_timeout) is wrapped in Fl::lock() and Fl::unlock().

- Don't change window decorations or titles from a worker thread

- The `make_current()` method will probably not work well for regular windows, but should always work for a Fl_Gl_Window to allow for high speed rendering on graphics cards with multiple pipelines. Managing thread-safe access to the GL pipelines is left as an exercise for the reader! (And may be target specific...)

See also: Fl::lock(), Fl::unlock(), Fl::awake(), Fl::awake(Fl_Awake_Handler cb, void∗ userdata), Fl::awake(void∗ message), Fl::thread_message().

1.14 Unicode and UTF-8 Support

FLTK provides comprehensive Unicode support through UTF-8 encoding, allowing your applications to handle international text and be easily localized for users worldwide.

1.14.1 Overview

Starting with version 1.3.0, FLTK uses UTF-8 as its primary text encoding. This means:

- All text in FLTK is expected to be UTF-8 encoded
- Your application can display text in any language
- File operations work correctly with international filenames
- Most existing ASCII code continues work unchanged

Note

Unicode support in FLTK is functional but still evolving. Some advanced features like bidirectional text and complex script shaping are not yet implemented.

1.14.2 Quick Start

International code test for HTML and PDF:

```
// This is a test
// 日本語テストテキスト
// 中文测试文本
// Ελληνικό κείμενο δοκιμής
```

For most applications, you simply need to ensure your text is UTF-8 encoded:

```
// These all work automatically with UTF-8:
Fl_Window window(400, 300, "Hallo Römer");  // Mixed ASCII and German
button->label("Café");                      // Accented characters
fl_fopen("pièce.txt", "r");
```

Note

FLTK supports most Unicode character sets, including Chinese, Cyrillic, Greek, and many more. However the Doxygen-to-pdf pipeline does not easily support those. So I limited this document to German and French examples.

1.14.3 What is Unicode and UTF-8?

Unicode is a standard that assigns a unique number to every character used in human languages - from Latin letters to Chinese characters to emoji. Each character has a "code point" like U+0041 for 'A' or U+00DF for 'ß'.

Note that some characters may be composed of multiple consecutive codepoints. FLTK supports only a subset of the more complex Unicode concepts.

UTF-8 is a way to store Unicode codepoints as a sequence of bytes, called an octet. UTF-8 is backward-compatible with ASCII and efficient for most text:

- ASCII characters (like 'A') use 1 byte
- European accented characters use 2 bytes
- Most other characters (Chinese, Arabic, etc.) use 3 bytes
- Rare characters and emoji may use 4 bytes

FLTK chose UTF-8 because it works well with existing C string functions and doesn't break legacy ASCII code.

1.14.4 Unicode Functions in FLTK

1.14.4.1 Text Validation and Analysis

Functions to check and analyze UTF-8 text:

fl_utf8test() - Check if a string contains valid UTF-8

```
const char* text = "Hello 世界";
int result = fl_utf8test(text, strlen(text));
// Returns: 0=invalid, 1=ASCII, 2=2-byte chars, 3=3-byte chars, 4=4-byte chars
```

fl_utf8len() - Get the length in bytes of a UTF-8 octet

```
char ch = '\xE4';  // First byte of a 3-byte UTF-8 sequence
int len = fl_utf8len(ch);  // Returns 3 (or -1 if invalid)
```

fl_utf8locale() - Check if system uses UTF-8 encoding

```
if (fl_utf8locale()) {
    // System uses UTF-8, no conversion needed
} else {
    // May need to convert from local encoding
}
```

fl_utf_nb_char() - Count Unicode codepoints by decoding UTF-8 octets

```
const char* text = "Téléphone";
int char_count = fl_utf_nb_char((unsigned char*)text, strlen(text));
// Returns 9 (number of characters, not bytes)
```

fl_utf8bytes() / fl_unichar_to_utf8_size() - Get number of bytes needed to create a UTF-8 octet for a Unicode codepoint

```
unsigned int unicode_char = 0x4E2D;  // Chinese character 'zhong'
int bytes_needed = fl_utf8bytes(unicode_char);  // Returns 3
```

fl_nonspacing() - Check if codepoint is non-spacing (combining character)

Non-spacing codepoints typically combine this codepoint with the folowing codepoint into a single glyph. Don't line-break after a nonspacing codepoint. A more detailed description and links to the Unicode standard can be found in the fl_nonspacing() docs.

1.14.4.2 Text Conversion

Functions to convert between encodings:

fl_utf8decode() / fl_utf8encode() - Convert one UTF-8 octet into a Unicode codepoint or a Unicode codepoint into a UTF-8 octet.

```
// Decode UTF-8 to Unicode code point
const char* utf8_char = "";
int len;
unsigned int unicode = fl_utf8decode(utf8_char, utf8_char + 3, &len);
// unicode = 0x4E2D, len = 3

// Encode Unicode codepoint to UTF-8
char buffer[5];
int bytes = fl_utf8encode(0x4E2D, buffer);  // Returns 3
buffer[bytes] = '\0';  // Now buffer contains ""
```

fl_utf8froma() / fl_utf8toa() - Convert between UTF-8 and single-byte ISO-8859-1 encoding

```
// Convert ISO-8859-1 to UTF-8
char utf8_buffer[200];
fl_utf8froma(utf8_buffer, sizeof(utf8_buffer), "café", 4);

// Convert UTF-8 to ISO-8859-1 (non-representable chars become '?')
char text_buffer[100];
fl_utf8toa("café", 5, text_buffer, sizeof(text_buffer));
```

fl_utf8fromwc() / fl_utf8towc() - Convert between UTF-8 and wide characters

```
// Convert wide string to UTF-8
wchar_t wide_text[] = L"Hello ";
char utf8_buffer[100];
fl_utf8fromwc(utf8_buffer, sizeof(utf8_buffer), wide_text, wcslen(wide_text));

// Convert UTF-8 to wide string
const char* utf8_text = "Hello ";
wchar_t wide_buffer[50];
fl_utf8towc(utf8_text, strlen(utf8_text), wide_buffer, 50);
```

fl_utf8toUtf16() - Convert UTF-8 to UTF-16

```
const char* utf8_text = "Hello ";
unsigned short utf16_buffer[100];
unsigned int result = fl_utf8toUtf16(utf8_text, strlen(utf8_text),
                                     utf16_buffer, 100);
// Converts to UTF-16, handling surrogate pairs on Windows
```

fl_utf2mbcs() - Convert UTF-8 to local multibyte encoding

```
const char* utf8_text = "Hello ";
char* local_text = fl_utf2mbcs(utf8_text);
// Converts to system's local encoding (Windows CP, etc.)
// Remember to free the returned pointer
free(local_text);
```

fl_utf8from_mb() / fl_utf8to_mb() - Convert between UTF-8 and local multibyte

```
// Convert from local multibyte to UTF-8
char utf8_buffer[200];
fl_utf8from_mb(utf8_buffer, sizeof(utf8_buffer), local_text, strlen(local_text));

// Convert from UTF-8 to local multibyte
char local_buffer[200];
fl_utf8to_mb(utf8_text, strlen(utf8_text), local_buffer, sizeof(local_buffer));
```

1.14.4.3 Text Navigation

Functions to move through UTF-8 text safely:

fl_utf8back() / fl_utf8fwd() - Find codepoint boundaries

```
const char* text = "Café";
const char* start = text;
const char* end = text + strlen(text);
const char* e_pos = text + 3;  // Points to 'é'

// Move to previous codepoint
const char* c_pos = fl_utf8back(e_pos, start, end);  // Points to 'f'

// Move to next codepoint
const char* next_pos = fl_utf8fwd(e_pos, start, end);  // Points after 'é'
```

1.14.4.4 String Operations

UTF-8 aware string functions:

fl_utf8strlen() - Count Unicode codepoints in a UTF-8 encoded text

```
const char* text = "Café";  // 5 bytes, 4 characters
int chars = fl_utf8strlen(text);  // Returns 4
int bytes = strlen(text);         // Returns 5
```

fl_utf_strcasecmp() / fl_utf_strncasecmp() - Compare strings ignoring case. Please note the list of limitations in FLTK's Unicode support at the bottom of this page.

```
int result = fl_utf_strcasecmp("Café", "CAFÉ");  // Returns 0 (equal)
int result2 = fl_utf_strncasecmp("Café", "CAFÉ", 2);  // Compare first 2 chars
```

fl_tolower() / fl_toupper() - Convert case for individual Unicode characters

```
unsigned int lower_a = fl_tolower(0x41);   // 'A' -> 'a' (0x61)
unsigned int upper_e = fl_toupper(0xE9);   // 'é' -> 'É' (0xC9)
```

fl_utf_tolower() / fl_utf_toupper() - Convert case for UTF-8 strings

```
const char* text = "Café";
char lower_buffer[20];
fl_utf_tolower((unsigned char*)text, strlen(text), lower_buffer);
// lower_buffer now contains "café"
```

1.14.4.5 File Operations

Cross-platform file functions that handle UTF-8 filenames:

Basic file operations:

```
// These work with international filenames on all platforms:
FILE* f = fl_fopen("sœur.txt", "r");              // Open file
int fd = fl_open("gummibär.bin", O_RDONLY);       // Open with file descriptor
int result = fl_stat("français.dat", &stat_buf); // Get file info
```

File access and properties:

```
fl_access("château.txt", R_OK);           // Check if file is readable
fl_chmod("fête.dat", 0644);               // Change file permissions
fl_unlink("Küche.tmp");                   // Delete file
fl_rename("Äpfel.txt", "Birnen.txt");     // Rename file
```

Directory operations:

```
fl_mkdir("Straßen", 0755);                // Create directory
fl_rmdir("éléphant");                     // Remove directory
char current_dir[1024];
fl_getcwd(current_dir, sizeof(current_dir));  // Get current directory
```

Path operations:

```
fl_make_path("animaux/éléphant");           // Create directory path
fl_make_path_for_file("Tiere/Mäuse/Stuart.txt"); // Create path for file
```

Process and system operations:

```
fl_execvp("löschen", argv);                  // Execute program
fl_system("echo 'Hallo Rüdiger!'");          // Execute system command
char* value = fl_getenv("crème");            // Get environment variable
```

1.14.5 Best Practices

1.14.5.1 File Handling

- Always use fl_fopen(), fl_open(), etc. for file operations with international names
- Test with international filenames during development

1.14.5.2 String Processing

- Use fl_utf8strlen() instead of strlen() for character counts
- Use fl_utf8fwd()/fl_utf8back() when iterating through text character by character
- Validate user input with fl_utf8test() if accepting external data
- Be careful when truncating strings - use character boundaries, not arbitrary byte positions

1.14.5.3 Display and UI

- Test your interface with text in various languages (especially long German words or wide Asian characters)
- Consider that text length varies greatly between languages when designing layouts
- Ensure your chosen fonts support the characters you need to display

1.14.5.4 Performance Notes

- ASCII text has no performance overhead compared to single-byte encodings
- UTF-8 functions are optimized for common cases (ASCII and Western European text)
- File operations may be slightly slower on Windows due to UTF-16 conversion

1.14.6 Common Issues and Solutions

1.14.6.1 "My international text shows up as question marks"

Solution: Ensure your text is UTF-8 encoded and your font supports the characters. If reading from files, verify they're saved as UTF-8.

1.14.6.2 "File operations fail with international names"

Solution: Use FLTK's Unicode file functions instead of standard C functions:

```
// Instead of:
FILE* f = fopen("Löwe.txt", "r");  // May fail on Windows

// Use:
FILE* f = fl_fopen("Löwe.txt", "r");  // Works correctly
```

1.14.6.3 "String length calculations are wrong"

Solution: Use UTF-8 aware functions:

```
// Wrong - counts bytes, not characters:
int len = strlen("Café");  // Returns 5

// Correct - counts characters:
int len = fl_utf8strlen("Café");  // Returns 4
```

1.14.6.4 "Text gets corrupted when I truncate it"

Solution: Don't truncate UTF-8 strings at arbitrary byte positions:

```
// Wrong - may cut in middle of character:
char truncated[10];
strncpy(truncated, utf8_text, 9);

// Correct - find proper character boundary:
const char* end = utf8_text;
int char_count = 0;
while (char_count < max_chars && *end) {
    end = fl_utf8fwd(end, utf8_text, utf8_text + strlen(utf8_text));
    char_count++;
}
int safe_length = end - utf8_text;
```

1.14.7 Error Handling

FLTK handles invalid UTF-8 sequences gracefully using configurable behavior:

Error handling modes (compile-time configuration):

- **ERRORS_TO_CP1252** (default): Treats bytes 0x80-0x9F as CP1252 characters
- **STRICT_RFC3629**: Strict UTF-8 validation according to RFC 3629
- **ERRORS_TO_ISO8859_1** (default): Invalid bytes returned as-is, otherwise returns Unicode replacement character (U+FFFD)

Note

You can configure these with compiler flags like -DERRORS_TO_CP1252=0

This design allows FLTK to handle legacy text files that mix encodings, making it more robust in real-world scenarios.

1.14.8 Current Limitations

FLTK's Unicode support covers most common use cases but has some limitations:

Text Processing:

- No automatic text normalization (combining characters are treated separately)
- No complex script shaping (may affect some Arabic, Indic scripts)
- No bidirectional text support (right-to-left languages like Arabic/Hebrew)

Character Range:

- Full Unicode range supported (U+000000 to U+10FFFF)
- Some legacy APIs may be limited to 16-bit characters (Basic Multilingual Plane)

Sorting and Comparison:

- String comparison is byte-based, not linguistically correct
- Use system locale functions for proper collation when needed for sorting

Composed Characters:

- Composed characters (base + combining accents) are treated as separate characters
- No automatic character composition or decomposition

Most applications won't encounter these limitations in practice. The Unicode support in FLTK is sufficient for displaying and processing international text in the majority of real-world scenarios.

1.15 Constants and Enumerations

Note

This file is not actively maintained any more, but is left here as a reference, until the doxygen documentation is completed.

See also

FL/Enumerations.H.

This appendix lists the enumerations provided in the <FL/Enumerations.H> header file, organized by section. Constants whose value are zero are marked with "(0)", this is often useful to know when programming.

1.15.1 Version Numbers

The FLTK version number is stored in a number of compile-time constants. Assuming FLTK version 1.2.3, this would be:

- FL_MAJOR_VERSION - The major release number = 1
- FL_MINOR_VERSION - The minor release number = 2
- FL_PATCH_VERSION - The patch release number = 3
- FL_VERSION - [Deprecated] A combined floating-point version number for the major, minor, and patch release numbers = 1.0203
- FL_API_VERSION - A combined integer version number for the major, minor, and patch release numbers = 10203 (use this instead of FL_VERSION, if possible)
- FL_ABI_VERSION - A combined integer version number for the application binary interface (ABI) major, minor, and patch release numbers, would be 10200 (default), see note below.

Note

The ABI version (FL_ABI_VERSION) is usually constant throughout one major/minor release version, for instance 10200 if FL_API_VERSION is 10203. Hence the ABI is constant if only the patch version is changed. You can change this with CMake though if you want the latest enhancements (called "ABI features").
See CHANGES.txt, README.CMake.txt, and README.abi-version.txt.

1.15.2 Events

Events are identified by an Fl_Event enumeration value. The following events are currently defined:

- FL_NO_EVENT - No event (or an event FLTK does not understand) occurred (0).
- FL_PUSH - A mouse button was pushed.
- FL_RELEASE - A mouse button was released.
- FL_ENTER - The mouse pointer entered a widget.
- FL_LEAVE - The mouse pointer left a widget.

- FL_DRAG - The mouse pointer was moved with a button pressed.
- FL_FOCUS - A widget should receive keyboard focus.
- FL_UNFOCUS - A widget loses keyboard focus.
- FL_KEYBOARD - A key was pressed.
- FL_CLOSE - A window was closed.
- FL_MOVE - The mouse pointer was moved with no buttons pressed.
- FL_SHORTCUT - The user pressed a shortcut key.
- FL_DEACTIVATE - The widget has been deactivated.
- FL_ACTIVATE - The widget has been activated.
- FL_HIDE - The widget has been hidden.
- FL_SHOW - The widget has been shown.
- FL_PASTE - The widget should paste the contents of the clipboard.
- FL_SELECTIONCLEAR - The widget should clear any selections made for the clipboard.
- FL_MOUSEWHEEL - The horizontal or vertical mousewheel was turned.
- FL_DND_ENTER - The mouse pointer entered a widget dragging data.
- FL_DND_DRAG - The mouse pointer was moved dragging data.
- FL_DND_LEAVE - The mouse pointer left a widget still dragging data.
- FL_DND_RELEASE - Dragged data is about to be dropped.
- FL_SCREEN_CONFIGURATION_CHANGED - The screen configuration (number, positions) was changed.
- FL_FULLSCREEN - The fullscreen state of the window has changed.

1.15.3 Callback "When" Conditions

The following constants determine when a callback is performed:

- FL_WHEN_NEVER - Never call the callback (0).
- FL_WHEN_CHANGED - Do the callback only when the widget value changes.
- FL_WHEN_NOT_CHANGED - Do the callback whenever the user interacts with the widget.
- FL_WHEN_RELEASE - Do the callback when the button or key is released and the value changes.
- FL_WHEN_ENTER_KEY - Do the callback when the user presses the ENTER key and the value changes.
- FL_WHEN_RELEASE_ALWAYS - Do the callback when the button or key is released, even if the value doesn't change.
- FL_WHEN_ENTER_KEY_ALWAYS - Do the callback when the user presses the ENTER key, even if the value doesn't change.

1.15.4 Fl::event_button() Values

The following constants define the button numbers for FL_PUSH and FL_RELEASE events:

- FL_LEFT_MOUSE - the left mouse button
- FL_MIDDLE_MOUSE - the middle mouse button
- FL_RIGHT_MOUSE - the right mouse button
- FL_BACK_MOUSE - the back mouse button (side button 1)
- FL_FORWARD_MOUSE - the forward mouse button (side button 2)

1.15.5 Fl::event_key() Values

The following constants define the non-ASCII keys on the keyboard for FL_KEYBOARD and FL_SHORTCUT events:

- FL_Button - A mouse button; use `Fl_Button + n` for mouse button `n`.
- FL_BackSpace - The backspace key.
- FL_Tab - The tab key.
- FL_Enter - The enter key.
- FL_Pause - The pause key.
- FL_Scroll_Lock - The scroll lock key.
- FL_Escape - The escape key.
- FL_Home - The home key.
- FL_Left - The left arrow key.
- FL_Up - The up arrow key.
- FL_Right - The right arrow key.
- FL_Down - The down arrow key.
- FL_Page_Up - The page-up key.
- FL_Page_Down - The page-down key.
- FL_End - The end key.
- FL_Print - The print (or print-screen) key.
- FL_Insert - The insert key.
- FL_Menu - The menu key.
- FL_Num_Lock - The num lock key.
- FL_KP - One of the keypad numbers or keys; use `FL_KP + 'n'` for number `n` and, say, `FL_KP + '*'`.
- FL_KP_Enter - The enter key on the keypad.
- FL_F - One of the function keys; use `FL_F + n` for function key `n`.
- FL_Shift_L - The lefthand shift key.

- FL_Shift_R - The righthand shift key.
- FL_Control_L - The lefthand control key.
- FL_Control_R - The righthand control key.
- FL_Caps_Lock - The caps lock key.
- FL_Meta_L - The left meta/Windows key.
- FL_Meta_R - The right meta/Windows key.
- FL_Alt_L - The left alt key.
- FL_Alt_R - The right alt key.
- FL_Delete - The delete key.
- FL_Alt_Gr - The AltGr key on some international keyboards.

1.15.6 Fl::event_state() Values

The following constants define bits in the Fl::event_state() value:

- FL_SHIFT - One of the shift keys is down.
- FL_CAPS_LOCK - The caps lock is on.
- FL_CTRL - One of the ctrl keys is down.
- FL_ALT - One of the alt keys is down.
- FL_NUM_LOCK - The num lock is on.
- FL_META - One of the meta/Windows keys is down.
- FL_COMMAND - An alias for FL_CTRL on Windows, X11 and Wayland, or FL_META on MacOS X.
- FL_CONTROL - An alias for FL_META on Windows, X11 and Wayland, or FL_CTRL on MacOS X.
- FL_SCROLL_LOCK - The scroll lock is on.
- FL_BUTTON1 - Mouse button 1 is pushed.
- FL_BUTTON2 - Mouse button 2 is pushed.
- FL_BUTTON3 - Mouse button 3 is pushed.
- FL_BUTTON4 - Mouse button 4 (back) is pushed.
- FL_BUTTON5 - Mouse button 5 (forward) is pushed.
- FL_BUTTONS - Any mouse button (1..5) is pushed.
- FL_BUTTON(n) - Mouse button n (where $n > 0$) is pushed.

1.15.7 Alignment Values

The following constants define bits that can be used with Fl_Widget::align() to control the positioning of the label:

- FL_ALIGN_CENTER - The label is centered (0).
- FL_ALIGN_TOP - The label is top-aligned.
- FL_ALIGN_BOTTOM - The label is bottom-aligned.
- FL_ALIGN_LEFT - The label is left-aligned.
- FL_ALIGN_RIGHT - The label is right-aligned.
- FL_ALIGN_CLIP - The label is clipped to the widget.
- FL_ALIGN_WRAP - The label text is wrapped as needed.
- FL_ALIGN_TOP_LEFT - The label appears at the top of the widget, aligned to the left.
- FL_ALIGN_TOP_RIGHT - The label appears at the top of the widget, aligned to the right.
- FL_ALIGN_BOTTOM_LEFT - The label appears at the bottom of the widget, aligned to the left.
- FL_ALIGN_BOTTOM_RIGHT - The label appears at the bottom of the widget, aligned to the right.
- FL_ALIGN_LEFT_TOP - The label appears to the left of the widget, aligned at the top. Outside labels only.
- FL_ALIGN_RIGHT_TOP - The label appears to the right of the widget, aligned at the top. Outside labels only.
- FL_ALIGN_LEFT_BOTTOM - The label appears to the left of the widget, aligned at the bottom. Outside labels only.
- FL_ALIGN_RIGHT_BOTTOM - The label appears to the right of the widget, aligned at the bottom. Outside labels only.
- FL_ALIGN_INSIDE - 'or' this with other values to put label inside the widget.
- FL_ALIGN_TEXT_OVER_IMAGE - Label text will appear above the image.
- FL_ALIGN_IMAGE_OVER_TEXT - Label text will be below the image.
- FL_ALIGN_IMAGE_NEXT_TO_TEXT - The image will appear to the left of the text.
- FL_ALIGN_TEXT_NEXT_TO_IMAGE - The image will appear to the right of the text.
- FL_ALIGN_IMAGE_BACKDROP - The image will be used as a background for the widget.

1.15.8 Fonts

The following constants define the standard FLTK fonts:

- FL_HELVETICA - Helvetica (or Arial) normal (0).
- FL_HELVETICA_BOLD - Helvetica (or Arial) bold.
- FL_HELVETICA_ITALIC - Helvetica (or Arial) oblique.
- FL_HELVETICA_BOLD_ITALIC - Helvetica (or Arial) bold-oblique.
- FL_COURIER - Courier normal.
- FL_COURIER_BOLD - Courier bold.

- FL_COURIER_ITALIC - Courier italic.
- FL_COURIER_BOLD_ITALIC - Courier bold-italic.
- FL_TIMES - Times roman.
- FL_TIMES_BOLD - Times bold.
- FL_TIMES_ITALIC - Times italic.
- FL_TIMES_BOLD_ITALIC - Times bold-italic.
- FL_SYMBOL - Standard symbol font.
- FL_SCREEN - Default monospaced screen font.
- FL_SCREEN_BOLD - Default monospaced bold screen font.
- FL_ZAPF_DINGBATS - Zapf-dingbats font.

1.15.9 Colors

The Fl_Color enumeration type holds a FLTK color value. Colors are either 8-bit indexes into a `virtual colormap` or 24-bit RGB color values. Color indices occupy the lower 8 bits of the value, while RGB colors occupy the upper 24 bits, for a byte organization of RGBI.

1.15.9.1 Color Constants

Constants are defined for the user-defined foreground and background colors, as well as specific colors and the start of the grayscale ramp and color cube in the `virtual colormap`. Inline functions are provided to retrieve specific grayscale, color cube, or RGB color values.

The following color constants can be used to access the user-defined colors:

- FL_BACKGROUND_COLOR - the default background color
- FL_BACKGROUND2_COLOR - the default background color for text, list, and valuator widgets
- FL_FOREGROUND_COLOR - the default foreground color (0) used for labels and text
- FL_INACTIVE_COLOR - the inactive foreground color
- FL_SELECTION_COLOR - the default selection/highlight color

The following color constants can be used to access the colors from the FLTK standard color cube:

- FL_BLACK
- FL_BLUE
- FL_CYAN
- FL_DARK_BLUE
- FL_DARK_CYAN
- FL_DARK_GREEN
- FL_DARK_MAGENTA

- FL_DARK_RED
- FL_DARK_YELLOW
- FL_GREEN
- FL_MAGENTA
- FL_RED
- FL_WHITE
- FL_YELLOW

The following are named values within the standard grayscale:

- FL_GRAY0
- FL_DARK3
- FL_DARK2
- FL_DARK1
- FL_LIGHT1
- FL_LIGHT2
- FL_LIGHT3

The inline methods for getting a grayscale, color cube, or RGB color value are described in the Colors section of the Drawing Things in FLTK chapter.

1.15.10 Cursors

The following constants define the mouse cursors that are available in FLTK. The double-headed arrows are bitmaps provided by FLTK on X, the others are provided by system-defined cursors.

- FL_CURSOR_DEFAULT - the default cursor, usually an arrow (0)
- FL_CURSOR_ARROW - an arrow pointer
- FL_CURSOR_CROSS - crosshair
- FL_CURSOR_WAIT - watch or hourglass
- FL_CURSOR_INSERT - I-beam
- FL_CURSOR_HAND - hand (uparrow on Windows)
- FL_CURSOR_HELP - question mark
- FL_CURSOR_MOVE - 4-pointed arrow
- FL_CURSOR_NS - up/down arrow
- FL_CURSOR_WE - left/right arrow
- FL_CURSOR_NWSE - diagonal arrow
- FL_CURSOR_NESW - diagonal arrow
- FL_CURSOR_NONE - invisible

1.15.11 FD "When" Conditions

- FL_READ - Call the callback when there is data to be read.
- FL_WRITE - Call the callback when data can be written without blocking.
- FL_EXCEPT - Call the callback if an exception occurs on the file.

1.15.12 Damage Masks

The following damage mask bits are used by the standard FLTK widgets:

- FL_DAMAGE_CHILD - A child needs to be redrawn.
- FL_DAMAGE_EXPOSE - The window was exposed.
- FL_DAMAGE_SCROLL - The Fl_Scroll widget was scrolled.
- FL_DAMAGE_OVERLAY - The overlay planes need to be redrawn.
- FL_DAMAGE_USER1 - First user-defined damage bit.
- FL_DAMAGE_USER2 - Second user-defined damage bit.
- FL_DAMAGE_ALL - Everything needs to be redrawn.

1.16 GLUT Compatibility

This appendix describes the GLUT compatibility header file supplied with FLTK.

FLTK's GLUT compatibility is based on the original GLUT 3.7 and the follow-on FreeGLUT 2.4.0 libraries.

1.16.1 Using the GLUT Compatibility Header File

You should be able to compile existing GLUT source code by including <FL/glut.H> instead of <GL/glut.↩h>. This can be done by editing the source, by changing the -I switches to the compiler, or by providing a symbolic link from GL/glut.h to FL/glut.H.

All files calling GLUT procedures must be compiled with C++. You may have to alter them slightly to get them to compile without warnings, and you may have to rename them to get make to use the C++ compiler.

You must link with the FLTK library. Most of FL/glut.H is inline functions. You should take a look at it (and maybe at test/glpuzzle.cxx in the FLTK source) if you are having trouble porting your GLUT program.

This has been tested with most of the demo programs that come with the GLUT and FreeGLUT distributions.

1.16.2 Known Problems

The following functions and/or arguments to functions are missing, and you will have to replace them or comment them out for your code to compile:

- `glutGet(GLUT_ELAPSED_TIME)`
- `glutGet(GLUT_SCREEN_HEIGHT_MM)`
- `glutGet(GLUT_SCREEN_WIDTH_MM)`
- `glutGet(GLUT_WINDOW_NUM_CHILDREN)`
- `glutInitDisplayMode(GLUT_LUMINANCE)`
- `glutKeyboardUpFunc`(void(*callback)(unsigned char key, int x, int y))
- `glutLayerGet(GLUT_HAS_OVERLAY)`
- `glutLayerGet(GLUT_LAYER_IN_USE)`
- `glutPushWindow()`
- `glutSetColor(),glutGetColor(),glutCopyColormap()`
- `glutVideoResize()` missing.
- `glutWarpPointer()`
- `glutWindowStatusFunc()`
- Spaceball, buttonbox, dials, and tablet functions

Most of the symbols/enumerations have different values than GLUT uses. This will break code that relies on the actual values. The only symbols guaranteed to have the same values are true/false pairs like GLUT_DOWN and GLUT_UP, mouse buttons GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, GLUT_RIGHT_BUTTON, and GLUT_KEY_F1 thru GLUT_KEY_F12.

The strings passed as menu labels are not copied.

`glutPostRedisplay()` does not work if called from inside a display function. You must use `glutIdle↩Func()` if you want your display to update continuously.

`glutSwapBuffers()` does not work from inside a display function. This is on purpose, because FLTK swaps the buffers for you.

`glutUseLayer()` does not work well, and should only be used to initialize transformations inside a resize callback. You should redraw overlays by using `glutOverlayDisplayFunc()`.

Overlays are cleared before the overlay display function is called. `glutLayerGet(GLUT_OVERLAY_↩DAMAGED)` always returns true for compatibility with some GLUT overlay programs. You must rewrite your code so that `gl_color()` is used to choose colors in an overlay, or you will get random overlay colors.

`glutSetCursor(GLUT_CURSOR_FULL_CROSSHAIR)` just results in a small crosshair.

The fonts used by `glutBitmapCharacter()` and `glutBitmapWidth()` may be different.

`glutInit(argc,argv)` will consume different switches than GLUT does. It accepts the switches recognized by Fl::args(), and will accept any abbreviation of these switches (such as "-di" for "-display").

1.16.3 Mixing GLUT and FLTK Code

You can make your GLUT window a child of a Fl_Window with the following scheme. The biggest trick is that GLUT insists on a call to `show()` the window at the point it is created, which means the Fl_Window parent window must already be shown.

- Don't call `glutInit()`.
- Create your Fl_Window, and any FLTK widgets. Leave a blank area in the window for your GLUT window.
- `show()` the Fl_Window. Perhaps call `show(argc,argv)`.
- Call `window->begin()` so that the GLUT window will be automatically added to it.
- Use `glutInitWindowSize()` and `glutInitWindowPosition()` to set the location in the parent window to put the GLUT window.
- Put your GLUT code next. It probably does not need many changes. Call `window->end()` immediately after the `glutCreateWindow()`!
- You can call either `glutMainLoop()`, Fl::run(), or loop calling Fl::wait() to run the program.

1.16.4 class Fl_Glut_Window

1.16.4.1 Class Hierarchy

```
Fl_Gl_Window
   |
   +----Fl_Glut_Window
```

1.16.4.2 Include Files

```
#include <FL/glut.H>
```

1.16.4.3 Description

Each GLUT window is an instance of this class. You may find it useful to manipulate instances directly rather than use GLUT window id's. These may be created without opening the display, and thus can fit better into FLTK's method of creating windows.

The current GLUT window is available in the global variable `glut_window`.

`new Fl_Glut_Window(...)` is the same as `glutCreateWindow()` except it does not `show()` the window or make the window current.

`window->make_current()` is the same as `glutSetWindow(number)`. If the window has not had `show()` called on it yet, some functions that assume an OpenGL context will not work. If you do `show()` the window, call `make_current()` again to set the context.

`~Fl_Glut_Window()` is the same as `glutDestroyWindow()`.

1.16.4.4 Members

The Fl_Glut_Window class contains several public members that can be altered directly:

member	description
display	A pointer to the function to call to draw the normal planes.
entry	A pointer to the function to call when the mouse moves into or out of the window.
keyboard	A pointer to the function to call when a regular key is pressed.
menu[3]	The menu to post when one of the mouse buttons is pressed.
mouse	A pointer to the function to call when a button is pressed or released.
motion	A pointer to the function to call when the mouse is moved with a button down.
overlaydisplay	A pointer to the function to call to draw the overlay planes.
passivemotion	A pointer to the function to call when the mouse is moved with no buttons down.
reshape	A pointer to the function to call when the window is resized.
special	A pointer to the function to call when a special key is pressed.
visibility	A pointer to the function to call when the window is iconified or restored (made visible.)

1.16.4.5 Methods

Fl_Glut_Window::Fl_Glut_Window(int x, int y, int w, int h, const char *title = 0)
Fl_Glut_Window::Fl_Glut_Window(int w, int h, const char *title = 0)

The first constructor takes 4 int arguments to create the window with a preset position and size. The second constructor with 2 arguments will create the window with a preset size, but the window manager will choose the position according to its own whims.

virtual Fl_Glut_Window::~Fl_Glut_Window()

Destroys the GLUT window.

void Fl_Glut_Window::make_current()

Switches all drawing functions to the GLUT window.

1.17 Forms Compatibility

This appendix describes the Forms compatibility included with FLTK.

Note

The Forms compatibility library is deprecated, no longer actively maintained since FLTK 1.3.0, and likely to be removed completely in FLTK 1.6. Building it was optional (ON) in FLTK 1.4.

Since FLTK 1.5 building the Forms compatibility library `fltk_forms` (aka `fltk::forms`) is disabled by default and can be enabled by one of the following commands:

```
- cmake -D FLTK_BUILD_FORMS:BOOL=ON ...
- cmake-gui ... (edit option FLTK_BUILD_FORMS)
```

Fluid can still **import** Forms and XForms designer (.fd) files but w/o any guarantees for working results. Manual fixes may be necessary.

In the next minor or major release (1.6 or higher) the Forms compatibility library may be removed entirely.

1.17.1 Importing Forms Layout Files

FLUID can read the `.fd` files put out by all versions of Forms and XForms fdesign. However, it will mangle them a bit, but it prints a warning message about anything it does not understand. FLUID cannot write fdesign files, so you should save to a new name so you don't write over the old one.

You will need to edit your main code considerably to get it to link with the output from FLUID. If you are not interested in this you may have more immediate luck with the forms compatibility header, <FL/forms.H>.

1.17.2 Using the Compatibility Header File

You should be able to compile existing Forms or XForms source code by changing the include directory switch to your compiler so that the `forms.h` file supplied with FLTK is included. The `forms.h` file simply pulls in <FL/forms.H> so you don't need to change your source code. Take a look at <FL/forms.H> to see how it works, but the basic trick is lots of inline functions. Most of the XForms demo programs work without changes.

You will also have to compile your Forms or XForms program using a C++ compiler. The FLTK library does not provide C bindings or header files.

Although FLTK was designed to be compatible with the GL Forms library (version 0.3 or so), XForms has bloated severely and its interface is X-specific. Therefore, XForms compatibility is no longer a goal of FLTK. Compatibility was limited to things that were free, or that would add code that would not be linked in if the feature is unused, or that was not X-specific.

To use any new features of FLTK, you should rewrite your code to not use the inline functions and instead use "pure" FLTK. This will make it a lot cleaner and make it easier to figure out how to call the FLTK functions. Unfortunately this conversion is harder than expected and even Digital Domain's inhouse code still uses `forms.H` a lot.

1.17.3 Problems You Will Encounter

Many parts of XForms use X-specific structures like `XEvent` in their interface. I did not emulate these! Unfortunately these features (such as the "canvas" widget) are needed by most large programs. You will need to rewrite these to use FLTK subclasses.

Fl_Free widgets emulate the *old* Forms "free" widget. It may be useful for porting programs that change the `handle()` function on widgets, but you will still need to rewrite things.

Fl_Timer widgets are provided to emulate the XForms timer. These work, but are quite inefficient and inaccurate compared to using Fl::add_timeout().

All instance variables are hidden. If you directly refer to the `x`, `y`, `w`, `h`, `label`, or other fields of your Forms widgets you will have to add empty parenthesis after each reference. The easiest way to do this is to globally replace `"->x"` with `"->x()"`, etc. Replace `"boxtype"` with `"box()"`.

`const char *` arguments to most FLTK methods are simply stored, while Forms would `strdup()` the passed string. This is most noticeable with the label of widgets. Your program must always pass static data such as a string constant or malloc'd buffer to `label()`. If you are using labels to display program output you may want to try the Fl_Output widget.

The default fonts and sizes are matched to the older GL version of Forms, so all labels will draw somewhat larger than an XForms program does.

fdesign outputs a setting of a "fdui" instance variable to the main window. I did not emulate this because I wanted all instance variables to be hidden. You can store the same information in the `user_data()` field of a window. To do this, search through the fdesign output for all occurrences of `"->fdui"` and edit to use `"->user_data()"` instead. This will require casts and is not trivial.

The prototype for the functions passed to `fl_add_timeout()` and `fl_set_idle_callback()` callback are different.

All the following XForms calls are missing:

- `FL_REVISION`, `fl_library_version()`
- `FL_RETURN_DBLCLICK` (use Fl::event_clicks())
- `fl_add_signal_callback()`
- `fl_set_form_atactivate()` `fl_set_form_atdeactivate()`
- `fl_set_form_property()`
- `fl_set_app_mainform()`, `fl_get_app_mainform()`
- `fl_set_form_minsize()`, `fl_set_form_maxsize()`
- `fl_set_form_event_cmask()`, `fl_get_form_event_cmask()`
- `fl_set_form_dblbuffer()`, `fl_set_object_dblbuffer()` (use an Fl_Double_Window instead)
- `fl_adjust_form_size()`
- `fl_register_raw_callback()`
- `fl_set_object_bw()`, `fl_set_border_width()`
- `fl_set_object_resize()`, `fl_set_object_gravity()`
- `fl_set_object_shortcutkey()`
- `fl_set_object_automatic()`
- `fl_get_object_bbox()` (maybe FLTK should do this)
- `fl_set_object_prehandler()`, `fl_set_object_posthandler()`
- `fl_enumerate_fonts()`
- Most drawing functions
- `fl_set_coordunit()` (FLTK uses pixels all the time)
- `fl_ringbell()`
- `fl_gettime()`
- `fl_win*()` (all these functions)
- `fl_initialize(argc,argv,x,y,z)` ignores last 3 arguments
- `fl_read_bitmapfile()`, `fl_read_pixmapfile()`
- `fl_addto_browser_chars()`
- `FL_MENU_BUTTON` just draws normally
- `fl_set_bitmapbutton_file()`, `fl_set_pixmapbutton_file()`
- `FL_CANVAS` objects
- `FL_DIGITAL_CLOCK` (comes out analog)
- `fl_create_bitmap_cursor()`, `fl_set_cursor_color()`
- `fl_set_dial_angles()`
- `fl_show_oneliner()`
- `fl_set_choice_shortcut(a,b,c)`
- command log

- Only some of file selector is emulated
- `FL_DATE_INPUT`
- `fl_pup*`() (all these functions)
- textbox object (should be easy but I had no sample programs)
- xyplot object

1.17.4 Additional Notes

These notes were written for porting programs written with the older IRISGL version of Forms. Most of these problems are the same ones encountered when going from old Forms to XForms:

Does Not Run In Background

The IRISGL library always forked when you created the first window, unless "foreground()" was called. FLTK acts like "foreground()" is called all the time. If you really want the fork behavior do "if (fork()) exit(0)" right at the start of your program.

You Cannot Use IRISGL Windows or fl_queue

If a Forms (not XForms) program if you wanted your own window for displaying things you would create a IRISGL window and draw in it, periodically calling Forms to check if the user hit buttons on the panels. If the user did things to the IRISGL window, you would find this out by having the value FL_EVENT returned from the call to Forms.

None of this works with FLTK. Nor will it compile, the necessary calls are not in the interface.

You have to make a subclass of Fl_Gl_Window and write a `draw()` method and `handle()` method. This may require anywhere from a trivial to a major rewrite.

If you draw into the overlay planes you will have to also write a `draw_overlay()` method and call `redraw_overlay()` on the OpenGL window.

One easy way to hack your program so it works is to make the `draw()` and `handle()` methods on your window set some static variables, storing what event happened. Then in the main loop of your program, call Fl::wait() and then check these variables, acting on them as though they are events read from `fl_queue`.

You Must Use OpenGL to Draw Everything

The file <FL/gl.h> defines replacements for a lot of IRISGL calls, translating them to OpenGL. There are much better translators available that you might want to investigate.

You Cannot Make Forms Subclasses

Programs that call `fl_make_object` or directly setting the handle routine will not compile. You have to rewrite them to use a subclass of Fl_Widget. It is important to note that the `handle()` method is not exactly the same as the `handle()` function of Forms. Where a Forms `handle()` returned non-zero, your `handle()` must call `do_callback()`. And your `handle()` must return non-zero if it "understood" the event.

An attempt has been made to emulate the "free" widget. This appears to work quite well. It may be quicker to modify your subclass into a "free" widget, since the "handle" functions match.

If your subclass draws into the overlay you are in trouble and will have to rewrite things a lot.

You Cannot Use <device.h>

If you have written your own "free" widgets you will probably get a lot of errors about "getvaluator". You should substitute:

Forms	FLTK
MOUSE_X	Fl::event_x_root()
MOUSE_Y	Fl::event_y_root()
LEFTSHIFTKEY,RIGHTSHIFTKEY	Fl::event_shift()
CAPSLOCKKEY	Fl::event_capslock()
LEFTCTRLKEY,RIGHTCTRLKEY	Fl::event_ctrl()
LEFTALTKEY,RIGHTALTKEY	Fl::event_alt()
MOUSE1,RIGHTMOUSE	Fl::event_state()
MOUSE2,MIDDLEMOUSE	Fl::event_state()
MOUSE3,LEFTMOUSE	Fl::event_state()

Anything else in `getvaluator` and you are on your own...

Font Numbers Are Different

The "style" numbers have been changed because I wanted to insert bold-italic versions of the normal fonts. If you use Times, Courier, or Bookman to display any text you will get a different font out of FLTK. If you are really desperate to fix this use the following code:

```
fl_font_name(3,"*courier-medium-r-no*");
fl_font_name(4,"*courier-bold-r-no*");
fl_font_name(5,"*courier-medium-o-no*");
fl_font_name(6,"*times-medium-r-no*");
fl_font_name(7,"*times-bold-r-no*");
fl_font_name(8,"*times-medium-i-no*");
fl_font_name(9,"*bookman-light-r-no*");
fl_font_name(10,"*bookman-demi-r-no*");
fl_font_name(11,"*bookman-light-i-no*");
```

1.18 Operating System Issues

This appendix describes the operating system specific interfaces in FLTK:

- Accessing the OS Interfaces
- The Wayland/X11 hybrid library
- The UNIX (X11) Interface
- The Windows Interface
- The Apple OS X Interface
- The Wayland Interface

1.18.1 Accessing the OS Interfaces

All programs that need to access the operating system specific interfaces must include the following header file:
`#include <FL/platform.H>`

This header file will define the appropriate interface for your environment. The pages that follow describe the functionality that is provided for each operating system.

Note

These definitions used to be in FL/x.H up to FLTK 1.3.x. Usage of FL/x.H is deprecated since FLTK 1.4.0. You should replace all references of FL/x.H with FL/platform.H if your target is FLTK 1.4 or later. FL/x.H will be retained for backwards compatibility for some releases but will be removed in a later (not yet specified) FLTK release.

WARNING:
The interfaces provided by this header file may change radically in new FLTK releases. Use them only when an existing generic FLTK interface is not sufficient.

1.18.2 The Wayland/X11 hybrid library

By default, the FLTK library is, under Linux and Unix, a Wayland/X11 hybrid which can run FLTK-based apps as Wayland clients or as X11 clients. The choice between running an app as a Wayland or an X11 client is done as follows, when the app runs function `fl_open_display()` (that function can be called explicitly by the app or implicitly by FLTK, for example the first time an Fl_Window is show()'n) :

- if the app contains a global boolean variable named fl_disable_wayland and this variable is true, X11 is used;
- if environment variable FLTK_BACKEND is not defined, Wayland is used if a Wayland compositor is available, otherwise X11 is used;
- if $FLTK_BACKEND equals "wayland", the library makes the app a Wayland client, and stops with error if no Wayland compositor is available;

- if $FLTK_BACKEND equals "x11", the library makes the app an X11 client even if a Wayland compositor is available.

The first condition listed above is meant to facilitate conversion of code written for FLTK 1.3.x and containing X11-specific code; add this single statement anywhere in the app's source code:

```
FL_EXPORT bool fl_disable_wayland = true;
```

and the app will always run as an X11 client.

After function `fl_open_display()` has been called, exactly one of the functions `fl_wl_display()` and `fl_x11_display()` returns a non-NULL value. When the former function does, the app runs as a Wayland client, and Wayland-specific functions and symbols described below (The Wayland Interface) can be used, whereas X11-specific functions and symbols cannot. Otherwise, the app runs as an X11 client, and only X11-specific functions and symbols below (The UNIX (X11) Interface) can be used.

Because a single app can be expected to run either Wayland or X11, it's necessary to use distinct names for global variables and functions in the X11- and the Wayland-specific source code.

Non-default configurations of the FLTK library under Linux/Unix are described in file README.Wayland.txt.

1.18.3 The UNIX (X11) Interface

Cross-platform applications should bracket X11-specific source code between `#if defined(FLTK_USE_X11)` / `#endif` and should ensure function `fl_x11_display()` returns non-NULL before calling X11-specific functions and using X11-specific symbols.

The UNIX interface provides access to the X Window System state information and data structures.

1.18.3.1 Handling Other X Events

void Fl::add_handler(int (*f)(int))

Installs a function to parse unrecognized events. If FLTK cannot figure out what to do with an event, it calls each of these functions (most recent first) until one of them returns non-zero. If none of them returns non-zero then the event is ignored.

FLTK calls this for any X events it does not recognize, or X events with a window ID that FLTK does not recognize. You can look at the X event in the `fl_xevent` variable.

The argument is the FLTK event type that was not handled, or zero for unrecognized X events. These handlers are also called for global shortcuts and some other events that the widget they were passed to did not handle, for example `FL_SHORTCUT`.

extern XEvent *fl_xevent

This variable contains the most recent X event.

extern ulong fl_event_time

This variable contains the time stamp from the most recent X event that reported it; not all events do. Many X calls like cut and paste need this value.

Window fl_xid(const Fl_Window *)

Returns the XID for a window, or zero if not `shown()`.

Deprecated Kept for compatibility with FLTK versions before 1.4. Use preferentially fl_x11_xid(const Fl_Window *) with versions 1.4 and above.

Fl_Window *fl_find(ulong xid)

Returns the Fl_Window that corresponds to the given XID, or `NULL` if not found. This function uses a cache so it is slightly faster than iterating through the windows yourself.

Deprecated Kept for compatibility with FLTK versions before 1.4. Use preferentially fl_x11_find(Window) with versions 1.4 and above.

int fl_handle(const XEvent &)

This call allows you to supply the X events to FLTK, which may allow FLTK to cooperate with another toolkit or library. The return value is non-zero if FLTK understood the event. If the window does not belong to FLTK and the `add_handler()` functions all return 0, this function will return false.

Besides feeding events your code should call Fl::flush() periodically so that FLTK redraws its windows.

This function will call the callback functions. It will not return until they complete. In particular, if a callback pops up a modal window by calling fl_choice(), for instance, it will not return until the modal function returns.

1.18.3.2 Drawing using Xlib

The following global variables are set before Fl_Widget::draw() is called, or by Fl_Window::make_current():

```
extern Display *fl_display; // for compatibility with previous FLTK versions
extern Display *fl_x11_display(); // preferred access starting with FLTK 1.4
extern Window fl_window;
extern GC fl_gc; // for compatibility with previous FLTK versions
extern GC fl_x11_gc(); // preferred access starting with FLTK 1.4
extern int fl_screen;
extern XVisualInfo *fl_visual;
extern Colormap fl_colormap;
```

You must use them to produce Xlib calls. Don't attempt to change them. A typical X drawing call is written like this:

```
XDrawSomething(fl_display, fl_window, fl_gc, ...);
```

Other information such as the position or size of the X window can be found by looking at Fl_Window::current(), which returns a pointer to the Fl_Window being drawn.

unsigned long fl_xpixel(Fl_Color i)
unsigned long fl_xpixel(uchar r, uchar g, uchar b)

Returns the X pixel number used to draw the given FLTK color index or RGB color. This is the X pixel that fl_color() would use.

int fl_parse_color(const char* p, uchar& r, uchar& g, uchar& b)

Convert a name into the red, green, and blue values of a color by parsing the X11 color names. On other systems, `fl_parse_color()` can only convert names in hexadecimal encoding, for example `#ff8083`.

extern XFontStruct *fl_xfont

Points to the font selected by the most recent fl_font(). This is not necessarily the current font of `fl_gc`, which is not set until fl_draw() is called. If FLTK was compiled with Xft support, `fl_xfont` will usually be 0 and `fl_xftfont` will contain a pointer to the `XftFont` structure instead.

extern void *fl_xftfont

If FLTK was compiled with Xft support enabled, `fl_xftfont` points to the xft font selected by the most recent fl_font(). Otherwise it will be 0. `fl_xftfont` should be cast to `XftFont*`.

1.18.3.3 Changing the Display, Screen, or X Visual

FLTK uses only a single display, screen, X visual, and X colormap. This greatly simplifies its internal structure and makes it much smaller and faster. You can change which it uses by setting global variables *before the first Fl_Window::show() is called.* You may also want to call Fl::visual(), which is a portable interface to get a full color and/or double buffered visual.

int Fl::display(const char *)

Set which X display to use. This actually does `putenv("DISPLAY=...")` so that child programs will display on the same screen if called with `exec()`. This must be done before the display is opened. This call is provided under MacOS and Windows but it has no effect.

extern Display *fl_display

The open X display. This is needed as an argument to most Xlib calls. Don't attempt to change it! This is `NULL` before the display is opened.

void fl_open_display()

Opens the display. Does nothing if it is already open. This will make sure `fl_display` is non-zero. You should call this if you wish to do X calls and there is a chance that your code will be called before the first `show()` of a window.

This may call Fl::abort() if there is an error opening the display.

void fl_x11_use_display(Display *d)

Directs FLTK to use a pre-established X11 connection.

void fl_close_display()

This closes the X connection. You do *not* need to call this to exit, and in fact it is faster to not do so! It may be useful to call this if you want your program to continue without the X connection. You cannot open the display again, and probably cannot call any FLTK functions.

extern int fl_screen

Which screen number to use. This is set by `fl_open_display()` to the default screen. You can change it by setting this to a different value immediately afterwards. It can also be set by changing the last number in the Fl::display() string to "host:0.#".

extern XVisualInfo *fl_visual
extern Colormap fl_colormap

The visual and colormap that FLTK will use for all windows. These are set by `fl_open_display()` to the default visual and colormap. You can change them before calling `show()` on the first window. Typical code for changing the default visual is:

```
Fl::args(argc, argv); // do this first so $DISPLAY is set
fl_open_display();
fl_visual = find_a_good_visual(fl_display, fl_screen);
if (!fl_visual) Fl::abort("No good visual");
fl_colormap = make_a_colormap(fl_display, fl_visual->visual, fl_visual->depth);
// it is now ok to show() windows:
window->show(argc, argv);
```

1.18.3.4 Using a Subclass of Fl_Window for Special X Stuff

FLTK can manage an X window on a different screen, visual and/or colormap, you just can't use FLTK's drawing routines to draw into it. But you can write your own `draw()` method that uses Xlib (and/or OpenGL) calls only.

FLTK can also manage XID's provided by other libraries or programs, and call those libraries when the window needs to be redrawn.

To do this, you need to make a subclass of Fl_Window and override some of these virtual functions:

virtual void Fl_Window::show()

If the window is already `shown()` this must cause it to be raised, this can usually be done by calling Fl_Window::show(). If not `shown()` your implementation must call either Fl_X::set_xid() or Fl_X::make↩_xid().

An example:

```
void MyWindow::show() {
  if (shown()) {Fl_Window::show(); return;}  // you must do this!
  fl_open_display();    // necessary if this is first window
  // we only calculate the necessary visual colormap once:
  static XVisualInfo *visual;
  static Colormap colormap;
  if (!visual) {
    visual = figure_out_visual();
    colormap = XCreateColormap(fl_display, RootWindow(fl_display,fl_screen),
                               vis->visual, AllocNone);
  }
  Fl_X::make_xid(this, visual, colormap);
}
```

`Fl_X *Fl_X::set_xid(Fl_Window*, Window xid)`

Allocate a hidden class called an Fl_X, put the XID into it, and set a pointer to it from the Fl_Window. This causes Fl_Window::shown() to return true.

void Fl_X::make_xid(Fl_Window∗, XVisualInfo∗ = fl_visual, Colormap = fl_colormap)

This static method does the most onerous parts of creating an X window, including setting the label, resize limitations, etc. It then does Fl_X::set_xid() with this new window and maps the window.

virtual void Fl_Window::flush()

This virtual function is called by Fl::flush() to update the window. For FLTK's own windows it does this by setting the global variables `fl_window` and `fl_gc` and then calling the `draw()` method. For your own windows you might just want to put all the drawing code in here.

The X region that is a combination of all `damage()` calls done so far is in `Fl_X::flx(this)->region`. If `NULL` then you should redraw the entire window. The undocumented function `fl_clip_region(XRegion)` will initialize the FLTK clip stack with a region or `NULL` for no clipping. You must set region to `NULL` afterwards as `fl_clip_region()` will own and delete it when done.

If `damage() & FL_DAMAGE_EXPOSE` then only X expose events have happened. This may be useful if you have an undamaged image (such as a backing buffer) around.

Here is a sample where an undamaged image is kept somewhere:

```
void MyWindow::flush() {
  fl_clip_region(Fl_X::flx(this)->region);
  Fl_X::flx(this)->region = 0;
  if (damage() != 2) {... draw things into backing store ...}
  ... copy backing store to window ...
}
```

Note

For compatibility with FLTK versions before 1.4, member function `Fl_X::flx(Fl_Window*)` can also be written `Fl_X::i(Fl_Window*)`.

virtual void Fl_Window::hide()

Destroy the window server copy of the window. Usually you will destroy contexts, pixmaps, or other resources used by the window, and then call Fl_Window::hide() to get rid of the main window identified by `xid()`. If you override this, you must also override the destructor as shown:

```
void MyWindow::hide() {
  if (mypixmap) {
    XFreePixmap(fl_display,mypixmap);
    mypixmap = 0;
  }
  Fl_Window::hide(); // you must call this
}
```

virtual void Fl_Window::~Fl_Window()

Because of the way C++ works, if you override `hide()` you *must* override the destructor as well (otherwise only the base class `hide()` is called):

```
MyWindow::~MyWindow() {
  hide();
}
```

Note

Access to the Fl_X hidden class requires to #define FL_INTERNALS before compilation.

1.18.3.5 Setting the Icon of a Window

FLTK recommends to set window icons using these platform-independent methods: Fl_Window::icon(const Fl_RGB_Image *) and Fl_Window::icons(const Fl_RGB_Image *[], int). See also methods setting default window icons Fl_Window::default_icon(const Fl_RGB_Image *) and Fl_Window::default_icons(const Fl_RGB_Image *[], int).

FLTK on X11 also supports, for backward compatibility, use of the deprecated method Fl_Window::icon(const void *) as follows :
Sets the icon for the window to the passed pointer. You will need to cast the icon `Pixmap` to a `char*` when calling this method. To set a monochrome icon using a bitmap compiled with your application use:

```
#include "icon.xbm"

fl_open_display(); // needed if display has not been previously opened

Pixmap p = XCreateBitmapFromData(fl_display, DefaultRootWindow(fl_display),
                                 icon_bits, icon_width, icon_height);

window->icon((const void*)p);
```

To use a multi-colored icon, the XPM format and library should be used as follows:

```
#include <X11/xpm.h>
#include "icon.xpm"

fl_open_display();                // needed if display has not been previously opened

Pixmap p, mask;

XpmCreatePixmapFromData(fl_display, DefaultRootWindow(fl_display),
                                 icon_xpm, &p, &mask, NULL);

window->icon((const void *)p);
```

When using the Xpm library, be sure to include it in the list of libraries that are used to link the application (usually "-lXpm").

> **NOTE:**
> You must call Fl_Window::show(int argc, char** argv) for the icon to be used. The Fl_Window::show() method does not bind the icon to the window.

Any window icon must be set with the above methods before the window is shown.

1.18.3.6 X Resources

When the Fl_Window::show(int argc, char** argv) method is called, FLTK looks for the following X resources:

- `background` - The default background color for widgets (color).
- `dndTextOps` - The default setting for drag and drop text operations (boolean).
- `foreground` - The default foreground (label) color for widgets (color).
- `scheme` - The default scheme to use (string).

- `selectBackground` - The default selection color for menus, etc. (color).
- `Text.background` - The default background color for text fields (color).
- `tooltips` - The default setting for tooltips (boolean).
- `visibleFocus` - The default setting for visible keyboard focus on non-text widgets (boolean).

Resources associated with the first window's Fl_Window::xclass() string are queried first, or if no class has been specified then the class "fltk" is used (e.g. `fltk.background`). If no match is found, a global search is done (e.g. `*background`).

1.18.3.7 Display Scaling Factor

FLTK uses the value of the Xft.dpi resource divided by 96. to initialize the display scaling factor. That is also what is done by the gnome and KDE desktops.

1.18.4 The Windows Interface

Cross-platform applications should bracket Windows-specific source code between `#ifdef _WIN32` / `#endif`.

The Windows interface provides access to the Windows GDI state information and data structures.

1.18.4.1 Using filenames with non-ASCII characters

In FLTK, all strings, including filenames, are UTF-8 encoded. The utility functions fl_fopen() and fl_open() allow to open files potentially having non-ASCII names in a cross-platform fashion, whereas the standard fopen()/open() functions fail to do so.

1.18.4.2 Responding to WM_QUIT

FLTK will intercept WM_QUIT messages that are directed towards the thread that runs the main loop. These are converted to SIGTERM signals via `raise()`. This allows you to deal with outside termination requests with the same code on both Windows and UNIX systems. Other processes can send this message via `PostThread↩Message()` in order to request, rather than force your application to terminate.

1.18.4.3 Handling Other Windows API Messages

By default a single WNDCLASSEX called "FLTK" is created. All Fl_Window's are of this class unless you use Fl_Window::xclass(). The window class is created the first time Fl_Window::show() is called.

You can probably combine FLTK with other libraries that make their own window classes. The easiest way is to call Fl::wait(), as it will call `DispatchMessage()` for all messages to the other windows. If necessary you can let the other library take over as long as it calls `DispatchMessage()`, but you will have to arrange for the function Fl::flush() to be called regularly so that widgets are updated, timeouts are handled, and the idle functions are called.

extern MSG fl_msg

This variable contains the most recent message read by `GetMessage()`, which is called by Fl::wait(). This may not be the most recent message sent to an FLTK window, because silly Windows calls the handle procedures directly for some events (sigh).

void Fl::add_handler(int (*f)(int))

Installs a function to parse unrecognized messages sent to FLTK windows. If FLTK cannot figure out what to do with a message, it calls each of these functions (most recent first) until one of them returns non-zero. The argument passed to the functions is the FLTK event that was not handled or zero for unknown messages. If all the handlers return zero then FLTK calls `DefWindowProc()`.

HWND fl_xid(const Fl_Window *)

Returns the window handle for a Fl_Window, or zero if not `shown()`.

Fl_Window *fl_find(HWND xid)

Returns the Fl_Window that corresponds to the given window handle, or `NULL` if not found. This function uses a cache so it is slightly faster than iterating through the windows yourself.

1.18.4.4 Drawing Things Using the Windows GDI

When the virtual function Fl_Widget::draw() is called, FLTK stores all the extra arguments you need to make a proper GDI call in some global variables:

```
extern HINSTANCE fl_display; // for compatibility with previous FLTK versions
extern HINSTANCE fl_win32_display(); // preferred access starting with FLTK 1.4
extern HWND fl_window;
extern HDC fl_gc;  // for compatibility with previous FLTK versions
extern HDC fl_win32_gc(); // preferred access starting with FLTK 1.4
COLORREF fl_RGB();
HPEN fl_pen();
HBRUSH fl_brush();
```

These global variables are set before Fl_Widget::draw() is called, or by Fl_Window::make_current(). You can refer to them when needed to produce GDI calls, but don't attempt to change them. The functions return GDI objects for the current color set by fl_color() and are created as needed and cached. A typical GDI drawing call is written like this:

```
DrawSomething(fl_gc, ..., fl_brush());
```

It may also be useful to refer to Fl_Window::current() to get the window's size or position.

1.18.4.5 HighDPI support

FLTK apps for the Windows platform are by default "Per-monitor DPI-aware V2". This means that any window automatically adjusts its physical size in relation to the scaling factor of the display where it maps. This also means that all drawings (e.g., text, lines, images) take advantage of the full resolution of the display in use. FLTK apps may also use the manifest mechanism to declare their level of DPI awareness. The FLTK library adapts to the DPI awareness level set in the app's manifest, which can be lower than the default level if the manifest sets it so.

1.18.4.6 Display Scaling Factor

FLTK uses the value given by function GetDpiForMonitor() divided by 96. to initialize the scaling factor of each display in the system. This matches the value of "Change the size of text, apps and other items" found in section "System" subsection "Display" of Windows settings.

1.18.4.7 Setting the Icon of a Window

FLTK recommends to set window icons using these platform-independent methods: Fl_Window::icon(const Fl_RGB_Image ∗) and Fl_Window::icons(const Fl_RGB_Image ∗[], int). See also methods setting default window icons Fl_Window::default_icon(const Fl_RGB_Image ∗) and Fl_Window::default_icons(const Fl_RGB_Image ∗[], int).

FLTK on Windows also supports, for backward compatibility, use of the deprecated method Fl_Window::icon(const void ∗) as follows :
Set the icon for the window to the passed pointer. You will need to cast the `HICON` handle to a `char*` when calling this method. To set the icon using an icon resource compiled with your application use:

```
window->icon((const void *)LoadIcon(fl_display, MAKEINTRESOURCE(IDI_ICON)));
```

You can also use the `LoadImage()` and related functions to load specific resolutions or create the icon from bitmap data.

> **NOTE:**
> You must call Fl_Window::show(int argc, char∗∗ argv) for the icon to be used. The Fl_Window::show() method does not bind the icon to the window.

Any window icon must be set with the above methods before the window is shown.

1.18.4.8 How to Not Get a MSDOS Console Window

Windows has a really stupid mode switch stored in the executables that controls whether or not to make a console window.

To always get a console window you simply create a console application (the "/SUBSYSTEM:CONSOLE" option for the linker). For a GUI-only application create a Windows application (the "/SUBSYSTEM:WINDOWS" option for the linker).

FLTK includes a `WinMain()` function that calls the ANSI standard `main()` entry point for you. *This function creates a console window when you use the debug version of the library.*

Windows applications without a console cannot write to `stdout` or `stderr`, even if they are run from a console window. Any output is silently thrown away. Additionally, Windows applications are run in the background by the console, although you can use "start /wait program" to run them in the foreground.

1.18.4.9 Known Windows Bugs and Problems

The following is a list of known bugs and problems in the Windows version of FLTK:

- If a program is deactivated, `Fl::wait()` does not return until it is activated again, even though many events are delivered to the program. This can cause idle background processes to stop unexpectedly. This also happens while the user is dragging or resizing windows or otherwise holding the mouse down. We were forced to remove most of the efficiency FLTK uses for redrawing in order to get windows to update while being moved. This is a design error in Windows and probably impossible to get around.

- `Fl_Gl_Window::can_do_overlay()` returns true until the first time it attempts to draw an overlay, and then correctly returns whether or not there is overlay hardware.

- `SetCapture` (used by `Fl::grab()`) doesn't work, and the main window title bar turns gray while menus are popped up.

- Compilation with `gcc 3.4.4` and `-Os` exposes an optimisation bug in gcc. The symptom is that when drawing filled circles only the perimeter is drawn. This can for instance be seen in the symbols demo. Other optimisation options such as -O2 and -O3 seem to work OK. More details can be found in STR#1656

1.18.5 The Apple OS X Interface

Cross-platform applications should bracket macOS-specific source code between `#if defined(__APPLE_↩_) && !defined(FLTK_USE_X11) / #endif`.

FLTK supports Apple OS X using the Apple Cocoa library. Older versions of MacOS are no longer supported.

Control, Option, and Command Modifier Keys

FLTK maps the Mac 'control' key to `FL_CTRL`, the 'option' key to `FL_ALT` and the 'Apple' key to `FL_META`. Furthermore, `FL_COMMAND` designates the 'Apple' key on Mac OS X and the 'control' key on other platforms. Keyboard events return the key name in Fl::event_key() and the keystroke translation in Fl::event_text(). For example, typing Option-Y on a Mac US keyboard will set `FL_ALT` in Fl::event_state(), set Fl::event_key() to 'y' and return the Yen symbol in Fl::event_text().

Right Click simulation with Ctrl Click

The Apple HIG guidelines indicate applications should support 'Ctrl Click' to simulate 'Right Click' for e.g. context menus, so users with one-button mice and one-click trackpads can still access right-click features. However, paraphrasing `Manolo's comment on the fltk.coredev newsgroup`:

- *FLTK does /not/ support Ctrl-Click == Right Click itself because Mac OS X event processing doesn't support this at the system level: the system reports left-clicks with the ctrl modifier when the user ctrl-clicks, and OS X system preferences don't allow changing this behavior. Therefore, applications must handle simulation of Right Click with Ctrl Click in the application code.*

Ian MacArthur provided the following handle() method code snippet showing an example of how to do this:

```
    case FL_PUSH:
    {
        int btn = Fl::event_button();
#ifdef __APPLE__
        int ev_state = Fl::event_state();
#endif
        //
        // Context menu can be called up in one of two ways: -
        //     1 - right click, as normally used on Windows and Linux
        //     2 - Ctrl + left click, as sometimes used on Mac
        //
#ifdef __APPLE__
        // On apple, check right click, and ctrl+left click
        if ((btn == FL_RIGHT_MOUSE) || (ev_state == (FL_CTRL | FL_BUTTON1)))
#else
        // On other platforms, only check right click as ctrl+left is used for selections
        if (btn == FL_RIGHT_MOUSE)
#endif
        {
            // Did we right click on the object?..
```

There is a thread about this subject on fltk.coredev (Aug 1-14, 2014) entitled "[RFC] Right click emulation for one button mouse on Mac".

Apple "Quit" Event

When the user presses Cmd-Q or requests a termination of the application, FLTK sends an `FL_CLOSE` event to all open windows. If any window remains open, the termination request aborts. If all windows close, the application's event loop terminates, that is, Fl::run() returns. The application can then follow FLTK's normal termination path executing cleanup code that may be programmed after termination of the event loop, and returning from main(). Function Fl::program_should_quit() allows to detect whether the event loop terminated because of a program termination request.

Apple "Open" Event

Whenever the user drops a file onto an application icon, OS X generates an Apple Event of the type "Open". You can have FLTK notify you of an Open event by calling the fl_open_callback() function.

void fl_open_display()

Opens the display. Does nothing if it is already open. You should call this if you wish to do Cocoa or Quartz calls and there is a chance that your code will be called before the first `show()` of a window.

Window fl_xid(const Fl_Window *)

Returns the window reference for an Fl_Window, or `NULL` if the window has not been shown. This reference is a pointer to an instance of the subclass FLWindow of Cocoa's NSWindow class.

Fl_Window *fl_find(Window xid)

Returns the Fl_Window that corresponds to the given window reference, or NULL if not found.

void fl_mac_set_about(Fl_Callback *cb, void *user_data, int shortcut)

Attaches the callback cb to the "About myprog" item of the system application menu. cb will be called with NULL first argument and user_data second argument. This MacOS-specific function is deprecated in FLTK 1.4 and replaced by Fl_Sys_Menu_Bar::about(Fl_Callback *cb, void *data) which is cross-platform.

Fl_Sys_Menu_Bar class

The Fl_Sys_Menu_Bar class allows to build menu bars that, on Mac OS X, are placed in the system menu bar (at top-left of display), and, on other platforms, at a user-chosen location of a user-chosen window.

1.18.5.1 Setting the icon of an application

- First, create a .icns file containing several copies of your icon of decreasing sizes. This can be done using the Preview application or the Icon Composer application available in "Graphics Tools for Xcode". To create a high resolution icon file, it is necessary to use the iconutil command-line utility.
- Put your .icns file in the Resources subdirectory of your application bundle.
- Add these two lines to the Info.plist file of your application bundle

  ```
  <key>CFBundleIconFile</key>
  <string>foo.icns</string>
  ```

  replacing foo by your application name. If you use Xcode, just add your .icns file to your application target.

1.18.5.2 Drawing Things Using Quartz

All code inside Fl_Widget::draw() is expected to call Quartz drawing functions. The Quartz coordinate system is flipped to match FLTK's coordinate system. The origin for all drawing is in the top left corner of the enclosing Fl_Window. The function fl_mac_gc() returns the appropriate Quartz 2D drawing environment (of type CGContextRef). For compatibility with previous FLTK versions, deprecated global variable fl_gc gives the same value.

Include FL/platform.H to declare the fl_mac_gc() function (or the fl_gc variable).

1.18.5.3 Internationalization

All FLTK programs contain an application menu with, e.g., the About xxx, Hide xxx, and Quit xxx items. This menu can be internationalized/localized by any of two means.

- using the Fl_Mac_App_Menu class.
- using the standard Mac OS X localization procedure. Create a language-specific .lproj directory (e.g., German.lproj) in the Resources subdirectory of the application bundle. Create therein a Localizable.strings file that translates all menu items to this language. The German Localizable.strings file, for example, contains:

  ```
  "About %@" = "Über %@";
  "Print Front Window"="Frontfenster drucken";
  "Services" = "Dienste";
  "Hide %@"="%@ ausblenden";
  "Hide Others"="Andere ausblenden";
  "Show All"="Alle einblenden";
  "Quit %@"="%@ beenden";
  ```

  Set "Print Front Window" = ""; therein so the application menu doesn't show a "Print Front Window" item. To localize the application name itself, create a file InfoPlist.strings in each .lproj directory and put CFBundleName = "localized name"; in each such file.

1.18.5.4 Fl_Double_Window

OS X double-buffers all windows automatically. On OS X, Fl_Window and Fl_Double_Window are handled internally in the same way.

1.18.5.5 Mac File System Specifics

FLTK uses UTF-8-encoded UNIX-style filenames and paths.

See also

Mac OS X-specific symbols

1.18.6 The Wayland Interface

Cross-platform applications should bracket Wayland-specific source code between `#ifdef FLTK_USE_WAYLAND` / `#endif` and should ensure function `fl_wl_display()` returns non-NULL before calling Wayland-specific functions and using Wayland-specific symbols.

extern struct wl_display *fl_wl_display();

After fl_open_display() has run, function `fl_wl_display()` returns a pointer to the struct wl_display representing the connection between the application and Wayland. For example, `wl_display_get_fd(fl_wl_display())` gives the file descriptor one can use to communicate with the Wayland compositor according to the Wayland protocol.

struct wld_window *fl_wl_xid(const Fl_Window *)

Returns a pointer to an FLTK-defined structure holding Wayland-related data created when a window gets show()'n, or NULL if not show()'n.

Fl_Window *fl_wl_find(struct wld_window * wld_win)

Returns the Fl_Window that corresponds to the given Window, or NULL if not found.

struct wl_surface *fl_wl_surface(struct wld_window *wld_win)

Returns a pointer to the struct wl_surface corresponding to a show()'n top-level window or subwindow.

cairo_t *fl_wl_gc()

Drawing natively to a Wayland window : Within an overridden Fl_Widget::draw() method, or after a call to Fl_Window::make_current(), it's possible to draw using the Cairo library. Function `fl_wl_gc()` returns the adequate `cairo_t*` value. Regular FLTK coordinates, with top-left origin, are to be used. All FLTK-defined drawing functions (e.g., fl_rect(), fl_draw()) can be used too.

void fl_close_display()

This closes the Wayland connection. You do not need to call this to exit. It may be useful to call this if you want your program to continue without the Wayland connection. You cannot open the display again, and cannot call any FLTK functions.

See also

wayland.H for all functions specific of the Wayland platform.

1.18.6.1 HiDPI display support

FLTK Wayland apps automatically scale according to the Wayland-defined, integer-valued scale factor. On a HiDPI display, it's enough to set this factor to 2 for any FLTK app to be drawn using twice as many pixels and thus to be as readable as it is on a regular display. With the gnome and KDE-plasma desktops, that is achieved in the "Displays" section of the "Settings" application, selecting 200 % for the "Scale" parameter. In addition to this, FLTK apps can also be scaled up or down typing ctrl/+/-/0/ and with the `FLTK_SCALING_FACTOR` environment variable.

1.18.6.2 Window icons

Standard FLTK functions Fl_Window::icon(const Fl_RGB_Image∗), Fl_Window::icons(const Fl_RGB_Image∗[], int), Fl_Window::default_icon(const Fl_RGB_Image∗) and Fl_Window::default_icons(const Fl_RGB_Image∗[], int) have no effect on the Wayland platform. The equivalent of a call to Fl_Window::default_icon(const Fl_RGB_Image∗) to set the application-specific window icon can be obtained as follows, using FLTK's editor app as an example:

- create a text file named `editor.desktop` containing :

```
[Desktop Entry]
Version=1.0
Type=Application
Name=Editor
Name[fr]=Editeur
Comment=FLTK editor
Exec=editor %F
Icon=/path/to/icon/file/editor.svg
MimeType=text/plain
```

- The `Name=` line therein determines the string displayed when the app runs.
- Optionally, one or more `Name[locale]=` lines can be used to set locale-specific app names.
- The `Icon=` line accepts also .png files.
- Put this file in `/usr/local/share/applications/` so it's available to all system users or in `$HOME/.local/share/applications/` so it's available to a single user.

1.18.6.3 Window titlebars

Wayland supports both client-side window decoration (CSD), where client applications are responsible for drawing window titlebars, and server-side window decoration (SSD), where the Wayland compositor itself draws window titlebars. Among 4 tested Wayland compositors, Mutter (gnome's compositor) and Weston use CSD mode whereas the KWin and Sway compositors use SSD mode. When running in CSD mode, FLTK uses a library called `libdecor` to draw titlebars. The libdecor library has been conceived to use various plug-in's to draw titlebars in various fashions intended to match any desktop's preferred titlebar style. FLTK supports drawing titlebars with any libdecor plug-in via an environment variable called `LIBDECOR_PLUGIN_DIR` which can be given the name of a directory containing the desired plug-in. When `LIBDECOR_PLUGIN_DIR` is not defined, or points to a directory that doesn't contain a libdecor plug-in, FLTK uses its built-in plug-in to draw titlebars. That is the most common situation, until libdecor plug-in's become available for popular UNIX desktops.

1.18.6.4 Fl_Double_Window

Under Wayland, windows are natively double-buffered. Therefore Wayland-FLTK handles Fl_Window and Fl_Double_Window internally the same way.

1.19 Migrating Code from FLTK 1.3 to 1.4

This appendix describes the differences between FLTK 1.3.x and FLTK 1.4.x functions and classes and potential requirements to change source code.

We also explain how code can be made compatible so it can be compiled by both FLTK 1.3.x and 1.4.x.

If you need to migrate your code from prior FLTK versions to FLTK 1.4, please consult the relevant appendices in the FLTK 1.3 online documentation or by downloading the FLTK 1.3 documentation. See https://www.fltk.org/doc-1.3/migration_1_3.html and/or https://www.fltk.org/software.php, respectively.

1.19.1 Changes in Header Files

We strive to include only necessary header files in the public headers of the FLTK library to reduce dependencies and hence compile times.

We try to avoid including system header files as far as possible. Known exceptions are `<stdio.h>` where file system structures and functions are visible in the public API, for instance `FILE*`, and sometimes essential header files like `<stdlib.h>` and/or `<stddef.h>`. Some required system headers **may** be included in platform specific header files like `<FL/platform.H>` or `<FL/platform_types.h>`.

In earlier versions (1.3.x) some of the public FLTK headers included some not strictly required system headers by accident.

The consequence for building user programs with FLTK 1.4 is: if you require a system or FLTK header in your user program that you don't `#include` explicitly but which has been included by FLTK 1.3.x your FLTK 1.3 program may issue compiler errors or warnings about missing header files or missing declarations when compiled with FLTK 1.4.

This is not a fault of FLTK 1.4 but a fault of the source code that did not include all required headers.

In FLTK 1.4 inclusion of `<FL/Fl.H>` is no longer a strict requirement as it was required and documented in FLTK 1.3.x. In FLTK 1.4 you may still need to `'#include <FL/Fl.H>'` if you are using enumerations or methods of class `Fl` like Fl::run() but there are exceptions where this header is included by other FLTK headers, like `Fl_Window.H` and other subclasses.

Suggested solution: include all FLTK and system header files your source code requires explicitly and don't rely on FLTK headers to include a particular header file. If you want your code to be as much as possible compatible with FLTK 1.3.x, then you should `'#include <FL/Fl.H>'` as required by 1.3.x.

You don't need to include headers of base classes - this is done by all FLTK headers as required. Besides that you need to include some support headers if you use FLTK functions like `fl_choice()` and others. This is described in the function's documentation (if a required header is missing in the docs this is a bug).

If you follow these rules your program will be compatible with both FLTK 1.3.x and FLTK 1.4.x as long as you use only functions and classes defined in FLTK 1.3.

1.19.2 Fl_Preferences

Starting with FLTK 1.3, preference databases are expected to be in UTF-8 encoding. Previous databases were stored in the current character set or code page which renders them incompatible for text entries using international characters.

Starting with FLTK 1.4, searching a valid path to store the preference files has changed slightly. Please see Fl_Preferences::Fl_Preferences(Root, const char*, const char*) for details.

On Unix/Linux platforms new FLTK preference files are stored using the XDG Base Directory Specification which means in essence that user preference files are stored in the user's home directory under the subdirectory .config, i.e. in $HOME/.config/fltk.org/ rather than $HOME/.fltk/fltk.org/. Existing preference files are still found and used, hence this new location is optional.

You may want to move the preference files from their old locations to their new locations as documented in Fl_Preferences::Fl_Preferences(Root, const char*, const char*) .

New Fl_Preferences types Fl_Preferences::USER_L, Fl_Preferences::SYSTEM_L and some more combinations with "_L" suffix have been defined to make preference files independent of the current locale. This is particularly important for floating point data which is stored in text form with varying decimal separator depending on the locale (either '.' or ','). You may want to change your program to use these new constants instead of those without the "_L" suffix. For more information see the documentation of Fl_Preferences.

1.19.3 Fl::add_timeout and friends

Since FLTK 1.4.0 internal timeout handling has been unified across platforms. This ensures equal timeout handling, improved accuracy of Fl::repeat_timeout(), and easier maintenance (less potential for errors).

This will very likely not affect user code, however there is one subtle exception on macOS and Windows: in FLTK 1.3.x these platforms used system timers to schedule timeouts. Since FLTK 1.4.0 all platforms use the same internal timer management that was previously only used on Unix/Linux/X11. The consequence of this change is that the FLTK event loop needs to be executed to trigger timeout events, i.e. you must either call Fl::wait() repeatedly or start the event loop with Fl::run().

Code that did not execute the event loop and relied on the system timers has never been cross platform compatible, i.e. it wouldn't work on Unix/Linux. An example would be code that opened a splash window, scheduled a timeout with Fl::add_timeout(), and waited for the timer event w/o running the FLTK event loop. Such code must be modified to execute Fl::run() and/or use Fl::wait().

1.19.4 New FL_OVERRIDE Macro

FLTK 1.4 defines a new macro FL_OVERRIDE as "override" if a recent C++ standard (C++11 or higher) is used to compile your code.

This macro is currently defined in FL/fl_attr.h but this may change in a future release. It is enough to '#include <FL/Fl.H>' to enable this macro.

Unfortunately Visual Studio does not define a meaningful value of __cplusplus to detect the C++ standard. Hence we use the Visual Studio version (2015 or higher) to decide whether we can define FL_OVERRIDE or not.

The FL_OVERRIDE macro is used to decorate declarations of overridden virtual methods in subclasses. Example code from FL/Fl_Window.H:

```
int handle(int) FL_OVERRIDE;
void resize(int X, int Y, int W, int H) FL_OVERRIDE;
Fl_Window * as_window() FL_OVERRIDE { return this; }
```

The FL_OVERRIDE macro translates to 'override' on newer compilers and to an empty string for older compilers.

We recommend to add this to your overridden virtual methods in subclasses derived from FLTK base classes (widgets) and to compile with C++ standard C++11 or higher to enable the compiler to detect some errors if methods are not overridden correctly.

You don't need to declare the overridden methods 'virtual' if you use FL_OVERRIDE or the keyword override.

Hint: For the GCC and clang compilers you can enable the warning '-Wsuggest-override' to detect where you may (want to) add the FL_OVERRIDE macro.

1.19.5 Fl_Image::copy() 'const'

Since FLTK 1.4.0 the virtual method Fl_Image::copy() has been declared 'const' so read-only ('const') images can be copied w/o casts.

This will very likely not affect user code. However, if you derived your own class from any of the Fl_*_Image variants **and** you overrode Your_Image::copy() then you **must** declare this 'const' as well, i.e. you must add the keyword 'const' to the declaration of copy() in your header file and in the implementation.

We suggest to add the new FL_OVERRIDE macro or the keyword 'override' (see above) to your own overridden method declarations to enable the compiler to detect such incompatibilities.

Code example in header file:

```
class Your_Image {
  // ...
  Fl_Image *copy() const FL_OVERRIDE;
  Fl_Image *copy(int w, int h) const FL_OVERRIDE;
};
```

Note the 'const' attribute **and** the FL_OVERRIDE macro.

1.19.6 Using X11 specific code with a "hybrid" FLTK library

Read this section if your FLTK 1.3 program uses X11 specific code with platform specific guards like

```
#if defined(WIN32)
  // Windows specific code here
#elif defined(__APPLE__)
  // macOS specific code here
#else
  // X11 specific code here
#endif
```

or similar.

You may skip this section if you don't use platform (X11) specific code.

FLTK 1.4 introduced Wayland support on Unix-like systems that support Wayland at runtime. The default build selects the Wayland runtime if it exists at program startup and falls back to using X11 if Wayland is not supported. On all currently known Wayland-enabled systems X11 programs are still supported by using "XWayland" which must be installed and enabled on the system. This is usually the case (as of this writing in November 2024).

Since using the Wayland runtime is the default you may get runtime errors if your program uses X11 specific code with the Wayland runtime.

There are two solutions:

1. Change the conditional code shown above everywhere in your program. This is the best and recommended long-term solution but may require some work.
2. Disable the Wayland backend (runtime) for your program. This can be a short-term solution for quick "porting" success.

Details about solution 1 are beyond the scope of this documentation but here's an example how such code can be rewritten in FLTK 1.4 for all platforms. This code is now in test/fltk-versions.cxx but may be changed in the future. Change this as you need.

```
static const char *get_platform() {
#if defined(_WIN32)
  return "Windows";
#elif defined(FLTK_USE_X11) || defined(FLTK_USE_WAYLAND)
# if defined(FLTK_USE_X11)
  if (fl_x11_display())
    return "Unix/Linux (X11)";
# endif
# if defined(FLTK_USE_WAYLAND)
  if (fl_wl_display())
    return "Unix/Linux (Wayland)";
# endif
  return "X11 or Wayland (backend unknown or display not opened)";
#elif defined(__APPLE__)
  return "macOS (native)";
#endif
  return "platform unknown, unsupported, or display not opened";
}
```

Solution 2 is described in chapter 2.1 of README.Wayland.txt. Please read this chapter if you need to support X11 specific code w/o the Wayland runtime.

Excerpt from README.Wayland.txt:

It is possible to force a program linked to a Wayland-enabled FLTK library to use X11 in all situations by putting this declaration somewhere in the source code:

```
FL_EXPORT bool fl_disable_wayland = true;
```

Please note that you may need to do more to link and run your program successfully, depending on the build system.

Additional info can be found in chapter The Wayland/X11 hybrid library.

1.19.7 Modern CMake

FLTK 1.4.0 supports "modern" CMake rather than old or "classic" CMake which was used in FLTK 1.3.x. Modern CMake was introduced in CMake 3.0 ($\sim$ 2014) and further developed in later CMake versions. FLTK 1.4.0 requires at least CMake 3.15 ($\sim$ 2019) as of Febrary 2024.

There are a lot of advantages that motivated this transition (mentioning only some):

- easier to use for projects using FLTK
- better structure
- uses CMake targets rather than variables
- embeddable in user projects via FetchContent() etc.
- embeddable in user projects via add_subdirectory()
- better coexistence with main projects if built as a subproject

Note that CMake targets can provide all required build flags and build dependencies which is the main advantage for user projects. For instance, instead of linking both fltk and fltk_images you need only fltk_images and fltk is linked in automatically.

Unfortunately there is one drawback you may encounter: Several CMake build option names have been changed, compared to FLTK 1.3.x. This is due to the fact that CMake cache variables are shared between the main (aka superbuild) project and all subprojects. Therefore all FLTK options are now prefixed with FLTK_.

This feature is now CMake standard and very common in newer projects. The CMake developers recommend strongly to use modern CMake.

We took the opportunity to redesign all CMake related options and target names for FLTK 1.4.0 to avoid changing these names later. Note that CMake support in 1.3.x was only experimental and the one in FLTK 1.4 (Git) up to the official release was beta state by definition. We apologize for all inconveniencies, hope that this is one of the rare exceptions in FLTK development, and that the new names are now stable as usual.

Changes in Detail:

Since FLTK 1.4.0 CMake target names are "namespaced", i.e. they are created with the prefix 'fltk::' and the old prefix 'fltk_' has been stripped off as far as the CMakeLists.txt file of user projects is concerned. The known filenames on disk did not change though.

The shared library target names use the common suffix "-shared" rather than "_SHARED".

The library 'fltk_cairo' is no longer used. Its functionality has been included in libfltk. FLTK 1.4.0 creates a dummy (empty) libfltk_cairo for backwards compatibility only. Please remove fltk_cairo from your projects and use only 'fltk::fltk' and/or the other libraries instead.

For more information and documentation of all options please refer to the file README.CMake.txt in the FLTK root directory.

Old and New Library Targets:

Library	Old Target	New Target	Shared Library Target
fltk	fltk	fltk::fltk	fltk::fltk-shared
fltk_forms	fltk_forms	fltk::forms	fltk::forms-shared
fltk_gl	fltk_gl	fltk::gl	fltk::gl-shared
fltk_images	fltk_images	fltk::images	fltk::images-shared
fltk_jpeg	fltk_jpeg	fltk::jpeg	fltk::jpeg-shared
fltk_png	fltk_png	fltk::png	fltk::png-shared
fltk_z	fltk_z	fltk::z	fltk::z-shared
fluid	fluid	fltk::fluid	n/a

For project developers used to the old (1.3.x) names the following table can assist to find the new option names. This table is ordered alphabetically by the old option name. Note that some option names did not change and some of the "old" names have been introduced in early 1.4.0 development.

Old Option Name (FLTK 1.3.x)	New Option Name (FLTK 1.4.x)
none (built always)	FLTK_BUILD_FORMS
FLTK_BUILD_EXAMPLES	FLTK_BUILD_EXAMPLES
FLTK_BUILD_FLTK_OPTIONS	FLTK_BUILD_FLTK_OPTIONS
FLTK_BUILD_FLUID	FLTK_BUILD_FLUID
FLTK_BUILD_TEST	FLTK_BUILD_TEST
FLTK_MSVC_RUNTIME_DLL	FLTK_MSVC_RUNTIME_DLL

Old Option Name (FLTK 1.3.x)	New Option Name (FLTK 1.4.x)
OPTION_ABI_VERSION	FLTK_ABI_VERSION
OPTION_ALLOW_GTK_PLUGIN	FLTK_USE_LIBDECOR_GTK
OPTION_APPLE_X11	FLTK_BACKEND_X11
OPTION_ARCHFLAGS	FLTK_ARCHFLAGS
OPTION_BUILD_HTML_DOCUMENTATION	FLTK_BUILD_HTML_DOCS
OPTION_BUILD_PDF_DOCUMENTATION	FLTK_BUILD_PDF_DOCS
OPTION_BUILD_SHARED_LIBS	FLTK_BUILD_SHARED_LIBS
OPTION_CAIRO	FLTK_OPTION_CAIRO_WINDOW
OPTION_CAIROEXT	FLTK_OPTION_CAIRO_EXT
OPTION_CREATE_LINKS	FLTK_INSTALL_LINKS
OPTION_FILESYSTEM_SUPPORT	FLTK_OPTION_FILESYSTEM_SUPPORT
OPTION_INCLUDE_DRIVER_DOCUMENTATION	FLTK_INCLUDE_DRIVER_DOCS
OPTION_INSTALL_HTML_DOCUMENTATION	FLTK_INSTALL_HTML_DOCS
OPTION_INSTALL_PDF_DOCUMENTATION	FLTK_INSTALL_PDF_DOCS
OPTION_LARGE_FILE	FLTK_OPTION_LARGE_FILE
OPTION_OPTIM	FLTK_OPTION_OPTIM
OPTION_PRINT_SUPPORT	FLTK_OPTION_PRINT_SUPPORT
OPTION_USE_CAIRO	FLTK_GRAPHICS_CAIRO
OPTION_USE_GDIPLUS	FLTK_GRAPHICS_GDIPLUS
OPTION_USE_GL	FLTK_BUILD_GL
OPTION_USE_KDIALOG	FLTK_USE_KDIALOG
OPTION_USE_PANGO	FLTK_USE_PANGO
OPTION_USE_POLL	FLTK_USE_POLL
OPTION_USE_STD	FLTK_OPTION_STD [1]
OPTION_USE_SVG	FLTK_OPTION_SVG
OPTION_USE_SYSTEM_LIBDECOR	FLTK_USE_SYSTEM_LIBDECOR
OPTION_USE_SYSTEM_LIBJPEG	FLTK_USE_SYSTEM_LIBJPEG
OPTION_USE_SYSTEM_LIBPNG	FLTK_USE_SYSTEM_LIBPNG
OPTION_USE_SYSTEM_ZLIB	FLTK_USE_SYSTEM_ZLIB
OPTION_USE_THREADS	FLTK_USE_PTHREADS
OPTION_USE_WAYLAND	FLTK_BACKEND_WAYLAND
OPTION_USE_XCURSOR	FLTK_USE_XCURSOR
OPTION_USE_XFIXES	FLTK_USE_XFIXES
OPTION_USE_XFT	FLTK_USE_XFT
OPTION_USE_XINERAMA	FLTK_USE_XINERAMA
OPTION_USE_XRENDER	FLTK_USE_XRENDER
OPTION_WAYLAND_ONLY	FLTK_BACKEND_X11=OFF

[1] Note: obsolete and therefore removed in 1.5

1.19.8 New FL_HELVETICA Font on Windows

In FLTK releases before 1.4.2 the Windows font name used for the FLTK font 'FL_HELVETICA' and its variants was named 'Arial'. It turned out that this was a very old font used in early Windows versions and had been replaced with the newer font name 'Microsoft Sans Serif'. This font has many more UTF-8 capable glyphs (characters) and is almost 100% compatible with the previously used font 'Arial'. It is supported by Microsoft since Windows XP, our oldest currently (partially) supported Windows platform.
This new font is used since FLTK 1.4.2.
We don't expect any issues with this much better font but if you really want strict backwards compatibility with FLTK 1.3.x or early 1.4.0/1.4.1 versions you can apply a tiny patch at the beginning of your program.
This code will load the old font 'Arial' on Windows:

```
#ifdef _WIN32
  // reset Windows fonts to pre-1.4.2 state
  Fl::set_font(FL_HELVETICA,     " Arial");
```

```
  Fl::set_font(FL_HELVETICA + 1, "BArial");
  Fl::set_font(FL_HELVETICA + 2, "IArial");
  Fl::set_font(FL_HELVETICA + 3, "PArial");
#endif
```

1.20 Migrating Code from FLTK 1.4 to 1.5

This appendix describes the differences between FLTK 1.4.x and FLTK 1.5.x functions and classes and potential requirements to change source code.
We also explain how code can be made compatible so it can be compiled by FLTK 1.3.x, 1.4.x, and 1.5.x.
If you need to migrate your code from prior FLTK versions to FLTK 1.4, please consult the relevant appendices in the FLTK 1.3 online documentation or by downloading the FLTK 1.3 documentation. See https://www.fltk.org/doc-1.3/migration_1_3.html and/or https://www.fltk.org/software.php, respectively.

1.20.1 CMake is required to build FLTK

Building with configure/make and included Makefile's has been discontinued. Since FLTK 1.5.0 CMake is required to build FLTK. Please see README.CMake.txt for details.

1.20.2 FLTK requires C++11 or higher

You need a compiler or build system capable of at least C++ standard C++11. The compiler must be configured to run in C++11 mode or higher.
FLTK sets the required compiler options automatically (via CMake) for its own build.
If you use "Modern CMake" (basically CMake in CONFIGURE mode) to build your own projects the appropriate options are inherited from the properties of the FLTK libraries. Otherwise you may need to set the compiler flags to use at least C++11 yourself.

1.20.3 Default for FORMS compatibility is now OFF

If you need (X)Forms compatibility you must set the CMake configure option

```
FLTK_BUILD_FORMS:BOOL=ON
```

manually. In previous versions (up to 1.4.x) the default was 'ON', but since Forms compatibility is not a primary goal of FLTK this is now optional.
(X)Forms support **may** be removed entirely in a future FLTK release. We recommend all users to rewrite their code to use "native" FLTK code anyway.

1.20.4 Simplified FL_OVERRIDE Macro

FLTK 1.4 defined a new macro `FL_OVERRIDE` as "override" if (and only if) a recent C++ standard (C++11 or higher) was used to compile its code.
Since FLTK 1.5 this macro is unconditionally defined as 'override' because FLTK requires C++11 anyway.
For more details please see New FL_OVERRIDE Macro.

1.20.5 Removed Methods and Functions

Methods, functions, and global symbols that have been deprecated in FLTK 1.3 or earlier have been removed in FLTK 1.5.0.
If you need help with these functions because old code still uses them, please consult the table below for a quick solution. If this doesn't help please read the FLTK 1.3 or 1.4 documentation to find out how to replace them with newer functions or methods, for instance https://www.fltk.org/doc-1.4/deprecated.html.
The following (possibly incomplete) list can help to identify and replace these removed methods and functions.

Removed Symbol	Please Replace With ...
Fl::release()	Fl::grab(0)
Fl::set_idle(Fl_Old_Idle_Handler cb)	Fl::add_idle(cb [, data])

Removed Symbol	Please Replace With ...
fl_ask(const char *fmt,...)	fl_choice()
fl_clip(int, int, int, int)	fl_push_clip(int x, int y, int w, int h)
Fl_Group::focus(Fl_Widget *W)	(Fl_Widget*) W->take_focus()
Fl_Menu_Item::check()	Fl_Menu_Item::set()
Fl_Menu_Item::checked() const	int Fl_Menu_Item::value() const
Fl_Menu_Item::uncheck()	Fl_Menu_Item::clear()
Fl_Widget::color2() const	Fl_Widget::selection_color() const
Fl_Widget::color2(unsigned)	Fl_Widget::selection_color(unsigned)
void Fl_File_Input::errorcolor(Fl_Color)	no replacement (not unused)
Fl_Color Fl_File_Input::errorcolor()	no replacement (was FL_RED)
static int Fl_Browser_::scrollbar_width()	static int Fl::scrollbar_size()
void Fl_Browser_::scrollbar_width(int)	Fl_Browser_::scrollbar_size(int)

1.20.6 More Deprecated Methods and Functions

All methods deprecated in **FLTK 1.4.x or earlier** are likely to be removed in the next minor FLTK version, presumably 1.6.0.
Please update your code!

Todo This table of deprecated functions since FLTK 1.5 may be incomplete and should be completed before the release of FLTK 1.5.0.

Symbol To Be Removed In The Next Minor Release	Please Replace With ...
void Fl_Window::icon(const void *ic)	void Fl_Window::icon(const Fl_RGB_Image *icon)
const void * Fl_Window::icon() const	no replacement (Windows platform specific)
Fl_Tree::first_visible()	in 1.3.3 ABI
Fl_Tree::first_visible_item()	?
Fl_Tree::item_clicked(Fl_Tree_Item *val)	callback_item()
Fl_Tree::item_clicked()	callback_item()
Fl_Tree::last_visible()	last_visible_item()
Fl_Tree_Item::Fl_Tree_Item(const Fl_Tree_Prefs &prefs)	Fl_Tree_Item(Fl_Tree*)
Fl_Tree_Item::next_displayed(Fl_Tree_Prefs &prefs)	next_visible()
Fl_Tree_Item::prev_displayed(Fl_Tree_Prefs &prefs)	prev_visible()

For details please refer to the documentation of FLTK 1.4 at https://www.fltk.org/doc-1.4/deprecated.html.

1.21 Software License

December 11, 2001

The FLTK library and included programs are provided under the terms of the GNU Library General Public License (LGPL) with the following exceptions:

1. Modifications to the FLTK configure script, config header file, and makefiles by themselves to support a specific platform do not constitute a modified or derivative work.

   The authors do request that such modifications be contributed to the FLTK project - send all contributions

through the "Software Trouble Report" on the following page: https://www.fltk.org/bugs.php

2. Widgets that are subclassed from FLTK widgets do not constitute a derivative work.

3. Static linking of applications and widgets to the FLTK library does not constitute a derivative work and does not require the author to provide source code for the application or widget, use the shared FLTK libraries, or link their applications or widgets against a user-supplied version of FLTK.

   If you link the application or widget to a modified version of FLTK, then the changes to FLTK must be provided under the terms of the LGPL in sections 1, 2, and 4.

4. You do not have to provide a copy of the FLTK license with programs that are linked to the FLTK library, nor do you have to identify the FLTK license in your program or documentation as required by section 6 of the LGPL.

   However, programs must still identify their use of FLTK. The following example statement can be included in user documentation to satisfy this requirement:

   [program/widget] is based in part on the work of the FLTK project (https://www.fltk.org).

GNU LIBRARY GENERAL PUBLIC LICENSE

Version 2, June 1991
Copyright (C) 1991 Free Software Foundation, Inc.
59 Temple Place - Suite 330, Boston, MA 02111-1307, USA
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.
[This is the first released version of the library GPL. It is numbered 2 because it goes with version 2 of the ordinary GPL.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users.
This license, the Library General Public License, applies to some specially designated Free Software Foundation software, and to any other libraries whose authors decide to use it. You can use it for your libraries, too.
When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.
To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library, or if you modify it.
For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link a program with the library, you must provide complete object files to the recipients so that they can relink them with the library, after making changes to the library and recompiling it. And you must show them these terms so they know their rights.
Our method of protecting your rights has two steps: (1) copyright the library, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the library.
Also, for each distributor's protection, we want to make certain that everyone understands that there is no warranty for this free library. If the library is modified by someone else and passed on, we want its recipients to know that

what they have is not the original version, so that any problems introduced by others will not reflect on the original authors' reputations.
Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that companies distributing free software will individually obtain patent licenses, thus in effect transforming the program into proprietary software. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.
Most GNU software, including some libraries, is covered by the ordinary GNU General Public License, which was designed for utility programs. This license, the GNU Library General Public License, applies to certain designated libraries. This license is quite different from the ordinary one; be sure to read it in full, and don't assume that anything in it is the same as in the ordinary license.
The reason we have a separate public license for some libraries is that they blur the distinction we usually make between modifying or adding to a program and simply using it. Linking a program with a library, without changing the library, is in some sense simply using the library, and is analogous to running a utility program or application program. However, in a textual and legal sense, the linked executable is a combined work, a derivative of the original library, and the ordinary General Public License treats it as such.
Because of this blurred distinction, using the ordinary General Public License for libraries did not effectively promote software sharing, because most developers did not use the libraries. We concluded that weaker conditions might promote sharing better.
However, unrestricted linking of non-free programs would deprive the users of those programs of all benefit from the free status of the libraries themselves. This Library General Public License is intended to permit developers of non-free programs to use free libraries, while preserving your freedom as a user of such programs to change the free libraries that are incorporated in them. (We have not seen how to achieve this as regards changes in header files, but we have achieved it as regards changes in the actual functions of the Library.) The hope is that this will lead to faster development of free libraries.
The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the libary" and a "work that uses the library". The former contains code derived from the library, while the latter only works together with the library.
Note that it is possible for a library to be covered by the ordinary General Public License rather than by this special one.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Library General Public License (also called "this License"). Each licensee is addressed as "you".
A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.
The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)
"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.
Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.
1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.
You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.
2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
a) The modified work must itself be a software library.

b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.
c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.
d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.
(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)
These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.
Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.
In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.
3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.
Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.
This option is useful when you wish to copy part of the code of the Library into a program that is not a library.
4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.
If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.
5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.
However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.
When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.
If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)
Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.
6. As an exception to the Sections above, you may also compile or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging

such modifications.
You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:
a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)
b) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.
c) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.
d) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.
For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.
It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.
7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:
a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.
b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.
8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.
10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.
If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.
It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is

up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.
This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.
12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
13. The Free Software Foundation may publish revised and/or new versions of the Library General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.
Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.
14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

1.22 Example Source Code

The FLTK distribution contains over 60 sample applications written in, or ported to, FLTK.
If the FLTK archive you received does not contain either an 'examples' or 'test' directory, you can download the complete FLTK distribution from `https://www.fltk.org/software.php`.
Most of the example programs were created while testing a group of widgets. They are not meant to be great achievements in clean C++ programming, but merely a test platform to verify the functionality of the FLTK library.
Note that extra example programs are also available in an additional 'examples' directory, but these are **NOT** built automatically when you build FLTK, unlike those in the 'test' directory shown below.

1.22.1 Example Applications: Overview

adjuster	animated	arc	ask	bitmap	blocks
boxtype	browser	button	buttons	cairo_test	checkers
clipboard	clock	colbrowser	color_chooser	cube	CubeView

cursor	curve	demo	device	doublebuffer	editor
fast_slow	file_chooser	FLUID	fonts	forms	fractals
fullscreen	gl_overlay	glpuzzle	hello	help_dialog	icon
iconize	image	inactive	input	input_choice	keyboard
label	line_style	list_visuals	mandelbrot	menubar	message
minimum	native-filechooser	navigation	offscreen	output	overlay
pack	pixmap	pixmap_browser	preferences	radio	resize
resizebox	rotated_text	scroll	shape	subwindow	sudoku
symbols	table	tabs	threads	tile	tiled_image
tree	twowin	unittests	utf8	valuators	windowfocus
New in FLTK 1.5					
gl_image					

1.22.1.1 adjuster

`adjuster` shows a nifty little widget for quickly setting values in a great range.

1.22.1.2 animated

`animated` shows a window with an animated square that shows drawing with transparency (alpha channel).

1.22.1.3 arc

The `arc` demo explains how to derive your own widget to generate some custom drawings. The sample drawings use the matrix based arc drawing for some fun effects.

1.22.1.4 ask

`ask` shows some of FLTK's standard dialog boxes. Click the correct answers or you may end up in a loop, or you may end up in a loop, or you... .

1.22.1.5 bitmap

This simple test shows the use of a single color bitmap as a label for a box widget. Bitmaps are stored in the X11 '.bmp' file format and can be part of the source code.

1.22.1.6 blocks

A wonderful and addictive game that shows the usage of FLTK timers, graphics, and how to implement sound on all platforms. `blocks` is also a good example for the Mac OS X specific bundle format.

1.22.1.7 boxtype

`boxtype` gives an overview of readily available boxes and frames in FLTK. More types can be added by the application programmer. When using themes, FLTK shuffles boxtypes around to give your program a new look.

1.22.1.8 browser

`browser` shows the capabilities of the Fl_Browser widget. Important features tested are loading of files, line formatting, and correct positioning of the browser data window.

1.22.1.9 button

The `button` test is a simple demo of push-buttons and callbacks.

1.22.1.10 buttons

`buttons` shows a sample of FLTK button types.

1.22.1.11 cairo_test

`cairo_test` shows a sample of drawing with Cairo in an Fl_Cairo_Window. This program can only be built completely if FLTK was configured with Cairo support. It displays a message box if Cairo support was not included.

1.22.1.12 checkers

Written by Steve Poulsen in early 1979, `checkers` shows how to convert a VT100 text-terminal based program into a neat application with a graphical UI. Check out the code that drags the pieces, and how the pieces are drawn by layering. Then tell me how to beat the computer at Checkers.

1.22.1.13 clipboard

The `clipboard` demo can be used to view the contents of the system clipboard, either text or image contents. Currently an image is preferred if the clipboard contains both formats. Images can be stored as PNG files so screenshots can be stored on disk with this little FLTK demo program.

1.22.1.14 clock

The `clock` demo shows two analog clocks. The innards of the Fl_Clock widget are pretty interesting, explaining the use of timeouts and matrix based drawing.

1.22.1.15 colbrowser

`colbrowser` runs only on X11 systems. It reads */usr/lib/X11/rgb.txt* to show the color representation of every text entry in the file. This is beautiful, but only moderately useful unless your UI is written in *Motif*.

1.22.1.16 color_chooser

The `color_chooser` gives a short demo of FLTK's palette based color chooser and of the RGB based color wheel.

1.22.1.17 cube

The `cube` demo shows the speed of OpenGL. It also tests the ability to render two OpenGL buffers into a single window, and shows OpenGL text.

1.22.1.18 CubeView

`CubeView` shows how to create a UI containing OpenGL with FLUID.

1.22.1.19 cursor

The `cursor` demo shows all mouse cursor shapes that come standard with FLTK. The *fgcolor* and *bgcolor* sliders work only on few systems (some version of Irix for example).

1.22.1.20 curve

`curve` draws a nice Bézier curve into a custom widget. The *points* option for splines is not supported on all platforms.

1.22.1.21 demo

This tool allows quick access to most programs in the `test` directory. The menu tree can be changed by editing `test/demo.menu`.

1.22.1.22 device

Exercises the Fl_Image_Surface, Fl_Copy_Surface, and Fl_Printer classes to draw to an Fl_Image object, copy graphical data to the clipboard, and for print support.

Note

The clipboard.cxx program of the 'test' directory is a clipboard watching application that continuously displays the textual or graphical content of the system clipboard (a.k.a pasteboard on macOS) exercising Fl::paste().

1.22.1.23 doublebuffer

The `doublebuffer` demo shows the difference between a single buffered window, which may flicker during a slow redraw, and a double buffered window, which never flickers, but uses twice the amount of RAM. Some modern OS's double buffer all windows automatically to allow transparency and shadows on the desktop. FLTK is smart enough to not tripple buffer a window in that case.

1.22.1.24 editor

FLTK has two very different text input widgets. Fl_Input and derived classes are rather light weight, however Fl_Text_Editor is a complete port of *nedit* (with permission). The `editor` test is almost a full application, showing custom syntax highlighting and dialog creation. See chapter Designing a Simple Text Editor for a tutorial about creating this program.

1.22.1.25 fast_slow

`fast_slow` shows how an application can use the Fl_Widget::when() setting to receive different kinds of callbacks.

1.22.1.26 file_chooser

The standard FLTK `file_chooser` is the result of many iterations, trying to find a middle ground between a complex browser and a fast light implementation.

1.22.1.27 fonts

`fonts` shows all available text fonts on the host system. If your machine still has some pixmap based fonts, the supported sizes will be shown in bold face. Only the first 256 fonts will be listed.

1.22.1.28 forms

`forms` is an XForms program with very few changes. Search for "fltk" to find all changes necessary to port to fltk. This demo shows the different boxtypes. Note that some boxtypes are not appropriate for some objects.

1.22.1.29 fractals

`fractals` shows how to mix OpenGL, Glut and FLTK code. FLTK supports a rather large subset of Glut, so that many Glut applications compile just fine.

1.22.1.30 fullscreen

This demo shows how to do many of the window manipulations that are popular for games. You can toggle the border on/off, switch between single- and double-buffered rendering, and take over the entire screen. More information in the source code.

1.22.1.31 gl_image

`gl_image` shows how to draw any image supported by FLTK in an Fl_Gl_Window.

1.22.1.32 gl_overlay

`gl_overlay` shows OpenGL overlay plane rendering. If no hardware overlay plane is available, FLTK will simulate it for you.

1.22.1.33 glpuzzle

The `glpuzzle` test shows how most Glut source code compiles easily under FLTK.

1.22.1.34 hello

`hello`: Hello, World. Need I say more? Well, maybe. This tiny demo shows how little is needed to get a functioning application running with FLTK. Quite impressive, I'd say.

1.22.1.35 help_dialog

`help_dialog` displays the built-in FLTK help browser. The Fl_Help_Dialog understands a subset of html and renders various image formats. This widget makes it easy to provide help pages to the user without depending on the operating system's html browser.

1.22.1.36 icon

`icon` demonstrates how an application icon can be set from an image. This icon should be displayed in the window bar (label), in the task bar, and in the task switcher (Windows: Alt-Tab). This feature is platform specific, hence it is possible that you can't see the icon. On Unix/Linux (X11) this can even depend on the Window Manager (WM).

1.22.1.37 iconize

`iconize` demonstrates the effect of the window functions hide(), iconize(), and show().

1.22.1.38 image

The `image` demo shows how an image can be created on the fly. This generated image contains an alpha (transparency) channel which lets previous renderings 'shine through', either via true transparency or by using screen door transparency (pixelation).

1.22.1.39 inactive

`inactive` tests the correct rendering of inactive widgets. To see the inactive version of images, you can check out the pixmap or image test.

1.22.1.40 input

This tool shows and tests different types of text input fields based on Fl_Input_. The `input` program also tests various settings of Fl_Input::when().

1.22.1.41 input_choice

`input_choice` tests the latest addition to FLTK1, a text input field with an attached pulldown menu. Windows users will recognize similarities to the 'ComboBox'. `input_choice` starts up in 'plastic' scheme, but the traditional scheme is also supported.

1.22.1.42 keyboard

FLTK unifies keyboard events for all platforms. The `keyboard` test can be used to check the return values of Fl::event_key() and Fl::event_text(). It is also great to see the modifier buttons and the scroll wheel at work. Quit this application by closing the window. The ESC key will not work.

1.22.1.43 label

Every FLTK widget can have a label attached to it. The `label` demo shows alignment, clipping, and wrapping of text labels. Labels can contain symbols at the start and end of the text, like *@FLTK* or *@circle uh-huh @square*.

1.22.1.44 line_style

Advanced line drawing can be tested with `line_style`. Not all platforms support all line styles.

1.22.1.45 list_visuals

This little app finds all available pixel formats for the current X11 screen. But since you are now an FLTK user, you don't have to worry about any of this.

1.22.1.46 mandelbrot

`mandelbrot` shows two advanced topics in one test. It creates grayscale images on the fly, updating them via the *idle* callback system. This is one of the few occasions where the *idle* callback is very useful by giving all available processor time to the application without blocking the UI or other apps.

1.22.1.47 menubar

The `menubar` tests many aspects of FLTK's popup menu system. Among the features are radio buttons, menus taller than the screen, arbitrary sub menu depth, and global shortcuts.

1.22.1.48 message

`message` pops up a few of FLTK's standard message boxes.

1.22.1.49 minimum

The `minimum` test program verifies that the update regions are set correctly. In a real life application, the trail would be avoided by choosing a smaller label or by setting label clipping differently.

1.22.1.50 native-filechooser

The `native-filechooser` program invokes the platform specific file chooser, if available (see Fl_Native_File_Chooser widget).

1.22.1.51 navigation

`navigation` demonstrates how the text cursor moves from text field to text field when using the arrow keys, tab, and shift-tab.

1.22.1.52 offscreen

`offscreen` shows how to draw into an offscreen image and display the offscreen image in the program window.

1.22.1.53 output

`output` shows the difference between the single line and multi line mode of the Fl_Output widget. Fonts can be selected from the FLTK standard list of fonts.

1.22.1.54 overlay

The `overlay` test app shows how easy an FLTK window can be layered to display cursor and manipulator style elements. This example derives a new class from Fl_Overlay_Window and provides a new function to draw custom overlays.

1.22.1.55 pack

The `pack` test program demonstrates the resizing and repositioning of children of the Fl_Pack group. Putting an Fl_Pack into an Fl_Scroll is a useful way to create a browser for large sets of data.

1.22.1.56 pixmap

This simple test shows the use of a LUT based pixmap as a label for a box widget. Pixmaps are stored in the X11 '.xpm' file format and can be part of the source code. Pixmaps support one transparent color.

1.22.1.57 pixmap_browser

`pixmap_browser` tests the shared-image interface. When using the same image multiple times, Fl_Shared_Image will keep it only once in memory.

1.22.1.58 preferences

I do have my `preferences` in the morning, but sometimes I just can't remember a thing. This is where the Fl_Preferences come in handy. They remember any kind of data between program launches.

1.22.1.59 radio

The `radio` tool was created entirely with *FLUID*. It shows some of the available button types and tests radio button behavior.

1.22.1.60 resizebox

`resizebox` shows some possible ways of FLTK's automatic resize behavior.

1.22.1.61 rotated_text

`rotated_text` shows how text can be rotated, i.e. drawn in any given angle. This demo is device specific, for instance it works under X11 only if configured with Xft.

1.22.1.62 resize

The `resize` demo tests size and position functions with the given window manager.

1.22.1.63 scroll

`scroll` shows how to scroll an area of widgets, one of them being a slow custom drawing. Fl_Scroll uses clipping and smart window area copying to improve redraw speed. The buttons at the bottom of the window control decoration rendering and updates.

1.22.1.64 shape

`shape` is a very minimal demo that shows how to create your own OpenGL rendering widget. Now that you know that, go ahead and write that flight simulator you always dreamt of.

1.22.1.65 subwindow

The `subwindow` demo tests messaging and drawing between the main window and 'true' sub windows. A sub window is different to a group by resetting the FLTK coordinate system to 0, 0 in the top left corner. On Win32 and X11, subwindows have their own operating system specific handle.

1.22.1.66 sudoku

Another highly addictive game - don't play it, I warned you. The implementation shows how to create application icons, how to deal with OS specifics, and how to generate sound.

1.22.1.67 symbols

`symbols` are a speciality of FLTK. These little vector drawings can be integrated into labels. They scale and rotate, and with a little patience, you can define your own. The rotation number refers to 45 degree rotations if you were looking at a numeric keypad (2 is down, 6 is right, etc.).

1.22.1.68 table

The `table` demo shows the features of the Fl_Table widget.

1.22.1.69 tabs

The `tabs` tool was created with *FLUID*. It tests correct hiding and redisplaying of tabs, navigation across tabs, resize behavior, and no unneeded redrawing of invisible widgets.

The `tabs` application shows the Fl_Tabs widget on the left and the Fl_Wizard widget on the right side for direct comparison of these two panel management widgets.

1.22.1.70 threads

FLTK can be used in a multithreading environment. There are some limitations, mostly due to the underlying operating system. `threads` shows how to use Fl::lock(), Fl::unlock(), and Fl::awake() in secondary threads to keep FLTK happy. Although locking works on all platforms, this demo is not available on every machine.

1.22.1.71 tile

The `tile` tool shows a nice way of using Fl_Tile. To test correct resizing of subwindows, the widget for region 1 is created from an Fl_Window class.

1.22.1.72 tiled_image

The `tiled_image` demo uses a small image as the background for a window by repeating it over the full size of the widget. The window is resizable and shows how the image gets repeated.

1.22.1.73 tree

The `tree` demo shows the features of the Fl_Tree widget.

1.22.1.74 twowin

The `twowin` program tests focus transfer from one window to another window.

1.22.1.75 unittests

`unittests` exercises all of FLTK's drawing features (e.g., text, lines, circles, images), as well as scrollbars and schemes.

1.22.1.76 utf8

`utf8` shows all fonts available to the platform that runs it, and how each font draws each of the Unicode code points ranging between U+0020 and U+FFFF.

1.22.1.77 valuators

`valuators` shows all of FLTK's nifty widgets to change numeric values.

1.22.1.78 windowfocus

`windowfocus` shows a very special case when a new window is shown while the focus stays in the original window.

1.22.1.79 FLUID

`FLUID` is not only a big test program, but also a very useful visual UI designer. Many parts of `FLUID` were created using `FLUID`. Check out the FLUID User Manual and the tutorials that come with it at https://www.fltk.org/documentation.php .

1.22.2 Example Applications: Images

This chapter contains a few selected images of the test and example applications listed above. It is not meant to be complete or a full reference. The reason some images are included here is to show how the display **should** look when running the example programs.

1.22.2.1 cairo_test

The `cairo_test` demo program shows three shiny buttons drawn with Cairo in an Fl_Cairo_Window.

Figure 1.30 Buttons drawn with Cairo

1.22.2.2 icon

The `icon` program lets you set the program icon from an image (here an Fl_RGB_Image).

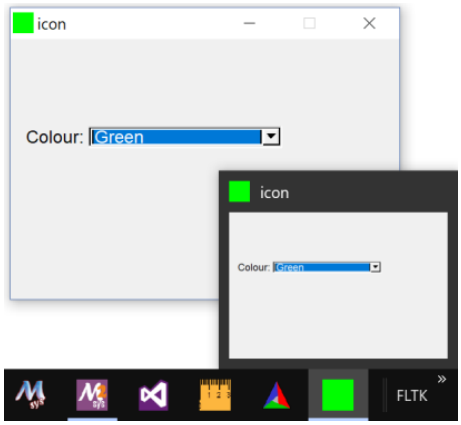


Figure 1.31 Green icon (Windows 10)

1.22.2.3 unittests

Select "drawing images" in the browser at the left side to see the image drawing example:

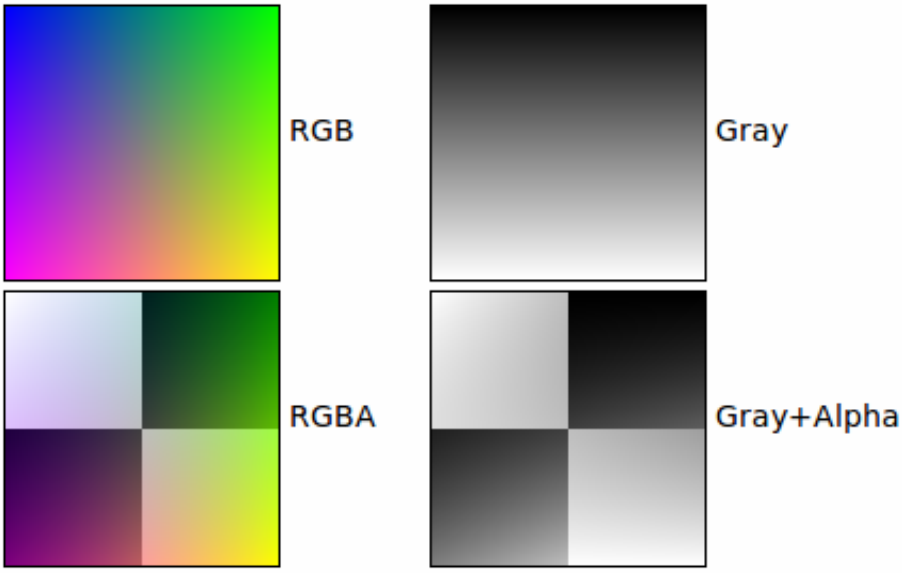


Figure 1.32 Image Drawing

1.23 FAQ (Frequently Asked Questions)

A list of frequently asked questions about FLTK.
This appendix describes various frequently asked questions regarding FLTK.

- Where do I start learning FLTK?
- How do I make a box with text?
- Can I use FLTK to make closed-source commercial applications?
- Hitting the 'Escape' key closes windows - how do I prevent this?

1.23.1 Where do I start learning FLTK?

It is assumed you know C++, which is the language all FLTK programs are written in, including FLTK itself.
If you like reading manuals to work your way into things, a good start is the FLTK documentation's Introduction to FLTK. Under the FLTK Basics section there's an example 'hello world' program that includes a line-by-line description.
If you like looking at simple code first to pique your interest, and then read up from there, start with the example programs in the test/ and examples/ directory that is included with the source code. A good place to start is the 'hello world' program in test/hello.cxx. Also do a google search for "FLTK example programs". "Erco's Cheat Page" is one that shows many simple examples of how to do specific things.
If you like to run example programs and look for ones that are like yours and then read them, download and build FLTK from the source, then run the test/demo program. Also, go into the 'examples/' directory and run 'make', then run some of those programs.
If you prefer watching TV to reading books and code, google search for "FLTK video tutorials" which has some introductory examples of how to write FLTK programs in C++ and build them.

1.23.2 How do I make a box with text?

The 'hello world' program shows how to make a box with text. All widgets have labels, so picking a simple widget like Fl_Box and setting its label() and using align() to align the label and labelfont() to set the font, and labelsize() to set the size, you can get text just how you want.
Labels are not selectable though; if you want selectable text, you can use Fl_Output or Fl_Multiline_Output for simple text that doesn't include scrollbars. For more complex text that might want scrollbars and multiple colors/fonts, use either Fl_Text_Display which handles plain text, or Fl_Help_View which handles simple HTML formatted text.

1.23.3 Can I use FLTK to make closed-source commercial applications?

Yes. The FLTK Software License is standard LGPL, but also includes a special clause ("exception") to allow for static linking. Specifically:

```
[from the top of the FLTK LGPL License section on exceptions]

3. Static linking of applications and widgets to the FLTK library does
not constitute a derivative work and does not require the author to
provide source code for the application or widget, use the shared FLTK
libraries, or link their applications or widgets against a user-supplied
version of FLTK.

If you link the application or widget to a modified version of FLTK,
then the changes to FLTK must be provided under the terms of the LGPL
in sections 1, 2, and 4.

4. You do not have to provide a copy of the FLTK license with programs
that are linked to the FLTK library, nor do you have to identify the
FLTK license in your program or documentation as required by section 6
of the LGPL.

However, programs must still identify their use of FLTK.  The following
example statement can be included in user documentation to satisfy
this requirement:

    [program/widget] is based in part on the work of the
    FLTK project (https://www.fltk.org).
```

1.23.4 Hitting the 'Escape' key closes windows - how do I prevent this?

[From FLTK article #378]

1. FLTK has a "global event handler" that makes Escape try to close the window, the same as clicking the close box. To disable this everywhere you can install your own that pretends it wants the escape key and thus stops the default one from seeing it (this may not be what you want, see below about the callbacks):

```
static int my_handler(int event) {
  if (event == FL_SHORTCUT) return 1; // eat all shortcut keys
  return 0;
```

```
}
...in main():
  Fl::add_handler(my_handler);
...
```

1. Attempts to close a window (both clicking the close box or typing Escape) call that window's callback. The default version of the callback does hide(). To make the window not close or otherwise do something different you replace the callback. To make the main window exit the program:

```
void my_callback(Fl_Widget*, void*) {
  exit(0);
}
...
  main_window->callback(my_callback);
...
```

If you don't want Escape to close the main window and exit you can check for and ignore it. This is better than replacing the global handler because Escape will still close pop-up windows:

```
void my_callback(Fl_Widget*, void*) {
  if (Fl::event() == FL_SHORTCUT && Fl::event_key() == FL_Escape)
    return; // ignore Escape
  exit(0);
}
```

The reason for calling a window callback can also be found using using the `Fl::callback_reason()` method:

```
void my_callback(Fl_Widget*, void*) {
  if (Fl::callback_reason() == FL_REASON_CANCELLED)
    return; // ignore that the user pressed the Escape key
  if (Fl::callback_reason() == FL_REASON_CLOSED)
    save_and_exit(); // user clicked the Close button in the window decoration
  exit(0); // fallback for other callback reasons
}
```

It is very common to ask for confirmation before exiting, this can be done with:

```
void my_callback(Fl_Widget*, void*) {
  if (fl_choice("Are you sure you want to quit?",
                "continue", "quit", NULL))
    exit(0);
}
```

Chapter 2

Fl_Terminal Technical Documentation

This chapter covers the vt100/xterm style "escape codes" used by Fl_Terminal for cursor positioning, text colors, and other display screen control features such as full or partial screen clearing, up/down scrolling, character insert/delete, etc.

2.1 The Escape Codes Fl_Terminal Supports

These are the escape codes Fl_Terminal actually supports, and is not the 'complete' list that e.g. xterm supports. Most of the important stuff has been implemented, but esoteric features (such as scroll regions) has not.
Features will be added as the widget matures.

```
-------------------------------------------------------
--- The CSI (Control Sequence Introducer, or "ESC[") ---
-------------------------------------------------------
ESC[#@ - (ICH) Insert blank Chars (default=1)
ESC[#A - (CUU) Cursor Up, no scroll/wrap
ESC[#B - (CUD) Cursor Down, no scroll/wrap
ESC[#C - (CUF) Cursor Forward, no wrap
ESC[#D - (CUB) Cursor Back, no wrap
ESC[#E - (CNL) Cursor Next Line (crlf) xterm, !gnome
ESC[#F - (CPL) Cursor Preceding Line: move to sol and up # lines
ESC[#G - (CHA) Cursor Horizontal Absolute positioning

   ESC[G   - move to column 1 (start of line, sol)
   ESC[#G  - move to column #
ESC[#H - (CUP) Cursor Position (#'s are 1 based)

   ESC[H    - go to row #1
   ESC[#H   - go to (row #) (default=1)
   ESC[#;#H - go to (row# ; col#)
ESC[#I - (CHT) Cursor Horizontal Tab: tab forward

   ESC[#I - tab # times (default 1)
ESC[#J - (ED) Erase in Display

   ESC[0J - clear to end of display (default)
   ESC[1J - clear to start of display
   ESC[2J - clear all lines
   ESC[3J - clear screen history
ESC[#K - (EL) Erase in line

   ESC[0K - clear to end of line (default)
   ESC[1K - clear to start of line
   ESC[2K - clear current line
ESC[#L - (IL)  Insert # Lines (default=1)
ESC[#M - (DL)  Delete # Lines (default=1)
ESC[#P - (DCH) Delete # Chars (default=1)
ESC[#S - (SU)  Scroll Up # lines (default=1)
ESC[#T - (SD)  Scroll Down # lines (default=1)
ESC[#X - (ECH) Erase Characters (default=1)
ESC[#Z - (CBT) Cursor Backwards Tab

   ESC[#Z - backwards tab # times (default=1)
ESC[#a - (HPR) move cursor relative [columns] (default=[row,col+1]) (NOT IMPLEMENTED)
ESC[#b - (REP) repeat prev graphics char # times                    (NOT IMPLEMENTED)
ESC[#d - (VPA) Line Position Absolute [row]                         (NOT IMPLEMENTED)
ESC[#e - (LPA) Line Position Relative [row]                         (NOT IMPLEMENTED)
ESC[#f - (CUP) cursor position (#'s 1 based), same as ESC[H
ESC[#g - (TBC)Tabulation Clear

   ESC[0g - Clear tabstop at cursor
   ESC[3g - Clear all tabstops
```

```
    ESC[#m - (SGR) Set Graphic Rendition

          *** Attribute Enable ***

        ESC[0m - reset: normal attribs/default fg/bg color  (VT100)
        ESC[1m - bold                                       (VT100)
        ESC[2m - dim
        ESC[3m - italic
        ESC[4m - underline                                  (VT100)
        ESC[5m - blink (NOT IMPLEMENTED)                    (VT100)
        ESC[6m - (unused)
        ESC[7m - inverse                                    (VT100)
        ESC[8m - (unused)
        ESC[9m - strikeout
        ESC[21m - doubly underline (Currently this just does single underline)

          *** Attribute Disable ***

        ESC[22m - disable bold/dim
        ESC[23m - disable italic
        ESC[24m - disable underline
        ESC[25m - disable blink (NOT IMPLEMENTED)
        ESC[26m - (unused)
        ESC[27m - disable inverse
        ESC[28m - disable hidden
        ESC[29m - disable strikeout

          *** Foreground Text "8 Color" ***

        ESC[30m - fg Black
        ESC[31m - fg Red
        ESC[32m - fg Green
        ESC[33m - fg Yellow
        ESC[34m - fg Blue
        ESC[35m - fg Magenta
        ESC[36m - fg Cyan
        ESC[37m - fg White
        ESC[39m - fg default

          *** Background Text "8 Color" ***

        ESC[40m - bg Black
        ESC[41m - bg Red
        ESC[42m - bg Green
        ESC[43m - bg Yellow
        ESC[44m - bg Blue
        ESC[45m - bg Magenta
        ESC[46m - bg Cyan
        ESC[47m - bg White
        ESC[49m - bg default

          *** Special RGB Color ***

        ESC [ 38 ; 2; Red ; Grn ; Blue m   - Foreground 8-bit Red,Grn,Blu values (in decimal 0-255)
        ESC [ 48 ; 2; Red ; Grn ; Blue m   - Background 8-bit Red,Grn,Blu values (in decimal 0-255)
    ESC[s     - save cursor pos (ansi.sys+xterm+gnome, but NOT vt100)
    ESC[u     - restore cursor pos (ansi.sys+xterm+gnome, but NOT vt100)
    ESC[>#q   - (DECSCA)  Set Cursor style (block/line/blink..) (NOT IMPLEMENTED)
    ESC[#;#r  - (DECSTBM) Set scroll Region top;bot             (NOT IMPLEMENTED)
    ESC[#..$t - (DECRARA)                                       (NOT IMPLEMENTED)
    ------------------------
    --- C1 Control Codes ---
    ------------------------
    <ESC>c    - (RIS) Reset term to Initial State
    <ESC>D    - (IND) Index: move cursor down a line, scroll if at bottom
    <ESC>E    - (NEL) Next Line: basically do a crlf, scroll if at bottom
    <ESC>H    - (HTS) Horizontal Tab Set: set a tabstop
    <ESC>M    - (RI)  Reverse Index (up w/scroll)
 NOTE: Acronyms in parens are Digital Equipment Corporation's names for these VT features.
```

2.2 Useful Terminal Escape Code Documentation

Useful links for reference:

- https://vt100.net/docs/vt100-ug/chapter3.html
- https://www.xfree86.org/current/ctlseqs.html
- https://www.x.org/docs/xterm/ctlseqs.pdf
- https://gist.github.com/justinmk/a5102f9a0c1810437885a04a07ef0a91 <– alphabetic!

- https://invisible-island.net/xterm/ctlseqs/ctlseqs.html

2.3 Fl_Terminal Design Document

When I started this project, I identified the key concepts needed to implement Fl_Terminal:

- Draw and manage multiline Unicode text in FLTK
- Allow per-character colors and attributes
- Efficient screen buffer to handle "scrollback history"
- Efficient scrolling with vertical scrollbar for even large screen history
- Mouse selection for copy/paste
- Escape code management to implement VT100 style / ANSI escape codes.

A class was created for each character, since characters can be either ASCII or UTF-8 encoded byte sequences. This class is called Utf8Char, and handles the character, its fg and bg color, and any attributes like dim, bold, italic, etc.
For managing the screen, after various experiments, I decided a ring buffer was the best way to manage things, the ring split in two:

- 'screen history' which is where lines scrolled off the top are saved
- 'display screen' displayed to the user at all times, and where the cursor lives

Scrolling the display, either by scrollbar or by new text causing the display to scroll up one line, would simply change an 'offset' index# of where in the ring buffer the top of the screen is, automatically moving the top line into the history, all without moving memory around.
In fact the only time screen memory is moved around is during these infrequent operations:

- during "scroll down" escape sequences, e.g. ESC[#T
- character insert/delete operations within a line, e.g. ESC[#@, ESC[#P
- changing the display size e.g. Fl_Terminal::resize()
- changing the history size e.g. Fl_Terminal::history_lines()

So a class "RingBuffer" is defined to manage the ring, and accessing its various parts, either as the entire entity ring, just the history, or just the display.
These three concepts, "ring", "history" and "display" are given abbreviated names in the RingBuffer class's API:

```
NOTE: Abbreviations "hist" and "disp"


"history" may be abbreviated as "hist", and "display" as "disp" in
both this text and the source code. 4 character names are used so
they line up cleanly in the source, e.g.

    ring_rows()     ring_cols()
    hist_rows()     hist_cols()
    disp_rows()     disp_cols()

        4 characters
```

These concepts were able to fit into C++ classes:

Utf8Char

Each character on the screen is a "Utf8Char" which can manage the UTF-8 encoding of any character as one or more bytes. Also in that class is a byte for an attribute (underline, bold, etc), and two integers for fg/bg color.

RingBuffer

The RingBuffer class keeps track of the buffer itself, a single array of Utf8Chars called "ring_chars" whose width is ring_cols() and whose height is ring_rows().
The "top" part of the ring is the history, whose width is hist_cols() and whose height is hist_rows(). hist_use() is used to define what part of the history is currently in use.
The "bottom" part of the ring is the display, whose width is disp_cols() and whose height is disp_rows().
An index number called "offset" points to where in the ring buffer the top of the ring currently is. This index changes each time the screen is scrolled, and affects both where the top of the display is, and where the top of the history is.
The memory layout of the Utf8Char character array is:

```
ring_chars[]:
              ___________________   _ _
             |                   |
             |                   |   |
             |                   |   |
             |   H i s t o r y   |   |  hist_rows
             |                   |   |
             |                   |   |
             |___________________|  _v_
             |                   |
             |                   |   |
             |   D i s p l a y   |   |  disp_rows
             |                   |   |
             |___________________|  _v_

             |<----------------->|
                   ring_cols
                   hist_cols
                   disp_cols
```

So it's basically a single continuous array of Utf8Char instances where any character can generally be accessed by index# using the formula:

```
ring_chars[ (row*ring_cols)+col ]
```

..where 'row' is the desired row, 'col' is the desired column, and 'ring_cols' is how many columns "wide" the buffer is.
The "offset" index affects that formula as an extra row offset, and the resulting index is then clamped within the range of the ring buffer using modulus.
Methods are used to allow direct access to the characters in the buffer that automatically handle the offset and modulus formulas, namely:

```
u8c_ring_row(row)+col   // access the entire ring by row/col
u8c_hist_row(row)+col   // access just the history buffer
u8c_disp_row(row)+col   // access just the display buffer
```

A key concept is the use of the simple 'offset' index integer to allow the starting point of the history and display to be moved around to implement 'text scrolling', such as when crlf at the screen bottom causes a 'scroll up'.
This is simply an "index offset" integer applied to the hist and disp indexes when drawing the display. So after scrolling two lines up, the offset is just increased by 2, redefining where the top of the history and display are, e.g.

```
  Offset is 0:           2    Offset now 2:

                             D i s p l a y
                       >

    H i s t o r y
                              H i s t o r y
                       2


                       >
    D i s p l a y
                               D i s p l a y
```

This 'offset' trivially implements "text scrolling", avoiding having to physically move memory around. Just the 'offset' changes, the text remains where it is in memory.

This also makes it appear the top line in the display is 'scrolled up' into the bottom of the scrollback 'history'.
If the offset exceeds the size of the ring buffer, it simply wraps around back to the beginning of the buffer with a modulo.
Indexes into the display and history are also modulo their respective rows, e.g.

```
act_ring_index = (hist_rows + disp_row + offset - scrollbar_pos) % ring_rows;
```

This way indexes for ranges can run beyond the bottom of the ring, and automatically wrap around the ring, e.g.

```
             > 2
                3    D i s p l a y
                4
                    <-- offset points here

        disp
        index         H i s t o r y
        wraps



                0    D i s p l a y
                1   <- ring_rows points to end of ring
              2 :                    :
                 3 :                  :
   disp_row(5) -> 4 :...................:
```

The dotted lines show where the display would be if not for the fact it extends beyond the bottom of the ring buffer (due to the current offset), and therefore wraps up to the top of the ring.
So to find a particular row in the display, in this case a 5 line display whose lines lie between 0 and 4, some simple math calculates the row position into the ring:

```
act_ring_index = (histrows      // the display exists AFTER the history, so offset the hist_rows
                  + offset      // include the scroll 'offset'
                  + disp_row    // add the desired row relative to the top of the display (0..disp_rows-1)
                 ) % ring_rows; // make sure the resulting index is within the ring buffer (0..ring_rows-1)
```

An additional bit of math makes sure if a negative result occurs, that negative value works relative to the end of the ring, e.g.

```
if (act_ring_index < 0) act_ring_index = ring_rows + act_ring_index;
```

This guarantees the act_ring_index is within the ring buffer's address space, with all offsets applied.
The math that implements this can be found in the u8c_xxxx_row() methods, where "xxxx" is one of the concept regions "ring", "hist" or "disp":

```
Utf8Char *u8c;
u8c = u8c_ring_row(rrow);    // address within ring, rrow can be 0..(ring_rows-1)
u8c = u8c_hist_row(hrow);    // address within hist, hrow can be 0..(hist_rows-1)
u8c = u8c_disp_row(drow);    // address within disp, drow can be 0..(disp_rows-1)
```

The small bit of math is only involved whenever a new row address is needed, so in a display that's 80x25, to walk all the characters in the screen, the math above would only be called 25 times, once for each row, and each column in the row is just a simple integer offset:

```
for ( int row=0; row<disp_rows(); row++ ) {     // walk rows: disp_rows = 25
  Utf8Char *u8c = u8c_disp_row(row);            // get first char in display 'row'
  for ( int col=0; col<disp_cols(); col++ ) {   // walk cols: disp_cols = 80
    u8c[col].do_something();                    // work with the char at row/col
  }
}
```

So to recap, the concepts here are:

- The ring buffer itself, a linear array that is conceptually split into a two-dimensional array of rows and columns whose height and width are:

  ```
  ring_rows -- how many rows in the entire ring buffer
  ring_cols -- how many columns in the ring buffer
  nchars    -- total chars in ring, e.g. (ring_rows * ring_cols)
  ```

- The "history" within the ring. For simplicity this is thought of as starting relative to the top of the ring buffer, occupying ring buffer rows:

  ```
  0 .. hist_rows()-1
  ```

- The "display", or "disp", within the ring, just after the "history". It occupies the ring buffer rows:

  ```
  hist_rows() .. hist_rows()+disp_rows()-1
  ```

  ..or similarly:

  ```
  (hist_rows)..(ring_rows-1)
  ```

  The following convenience methods provide access to the start and end indexes within the ring buffer for each entity:

Entire ring ring_srow() – start row index of the ring buffer (always 0) ring_erow() – end row index of the ring buffer
"history" part of ring hist_srow() – start row index of the screen history hist_erow() – end row index of the screen history
"display" part of ring disp_srow() – start row index of the display disp_erow() – end row index of the display
The values returned by these are as described above. For the hist_xxx() and disp_xxx() methods, the 'offset' is included into the formula. (For this reason hist_srow() won't always be zero the way ring_srow() is, due to the 'offset')
The values returned by these methods can all be passed to the u8c_ring_row() function to access the actual character buffer's contents.

- An "offset" used to move the "history" and "display" around within the ring buffer to implement the "text scrolling" concept. The offset is applied when new characters are added to the buffer, and during drawing to find where the display actually is within the ring.
- The "scrollbar", which only is used when redrawing the screen the user sees, and is simply an additional offset to all the above, where a scrollbar value of zero (the scrollbar tab at the bottom) shows the display rows, and as the scrollbar values increase as the user moves the scrollbar tab upwards, +1 per line, this is subtracted from the normal starting index to let the user work their way backwards into the scrollback history. Again, negative numbers wrap around within the ring buffer automatically.

The ring buffer allows new content to simply be appended to the ring buffer, and the index# for the start of the display and start of scrollback history are simply incremented. So the next time the display is "drawn", it starts at a different position in the ring.
This makes scrolling content at high speed trivial, without memory moves. It also makes the concept of "scrolling" with the scrollbar simple as well, simply being an extra index offset applied during drawing.

Mouse Selection

Dragging the mouse across the screen should highlight the text, allowing the user to extend the selection either beyond or before the point started. Extending the drag to the top of the screen should automatically 'scroll up' to select more lines in the scrollback history, or below the bottom to do the opposite.
The mouse selection is implemented as a class to keep track of the start/end row/col positions of the selection, and other details such as a flag indicating if a selection has been made, what color the fg/bg text should appear when text is selected, and methods that allow setting and extending the selection, clearing the selection, and "scrolling" the selection, to ensure the row/col indexes adjust correctly to track when the screen or scrollbar is scrolled.

Redraw Timer

Knowing when to redraw is tricky with a terminal, because sometimes high volumes of input will come in asynchronously, so in that case we need to determine when to redraw the screen to show the new content; too quickly will cause the screen to spend more time redrawing itself, preventing new input from being added. Too slowly, the user won't see new information appear in a timely manner.
To solve this, a rate timer is used to prevent too many redraws:

- When new data arrives, a short .01-sec initial timer is started, and a modified flag is set.
- redraw() is NOT called yet, allowing more data to continue to arrive quickly

- When the 1/10th second timer fires, the callback checks the modify flag:
  - if set, calls redraw(), resets the modify to 0, and calls Fl::repeat_timeout() to repeat the callback after the configured redraw_rate() interval.
  - if clear, no new data came in, so DISABLE the timer, done.

In this way, redraws don't happen more than 10x per second, and redraw() is called only when there's new content to see.
The redraw rate can be set by the user application using the Fl_Terminal::redraw_rate(), 0.10 being the default.
Some terminal operations necessarily call redraw() directly, such as interactive mouse selection, or during user scrolling the terminal's scrollbar, where it's important there's no delay in what the user sees while interacting directly with the widget.

Chapter 3

Development of the FLTK library

- FLTK Code Management Plan (CMP)
- The Wayland backend for its developer
- Developer info for bundled libs
- Developer Information

3.1 FLTK Code Management Plan (CMP)

*** **WORK IN PROGRESS** ***

3.1.1 CMP Coding Intro

This document defines the coding standards that all FLTK developers must follow when developing source code and related documentation for the FLTK library. The purpose of defining formal standards is to organize and focus our development efforts, ensure that all developers communicate and develop software with a common vocabulary/style, and make it possible for us to generate and release a high-quality GUI toolkit which can be used with a high degree of confidence.
Much of this file describes existing practices that have been used up through FLTK 1.1.x, however some new standards have been added for future code and releases.

3.1.2 Communication

The fltk.coredev mailing list/newsgroup are the primary means of communication between developers. All major design changes must be discussed prior to implementation.
We use GitHub Issues to manage bugs. Please see the CMP section on Managing GitHub Issues for how developers should manage GitHub issues.
It is wise for all developers to monitor these github related mailing list/newsgroups for bug and commit acitivity:

- fltk.commit — (nntp/web) All fltk developer commits on GitHub commit. Includes old pre-Oct 2018 SVN commits
- fltk.issues — (nntp/web) All new/current fltk bugs as GitHub Issues. See "Managing GitHub Issues"
- fltk.bugs — (nntp/web) The "Old STR Bug Management System" activity (replaced by GitHub Issues). See "Managing Old STR's"

To monitor these groups, either configure github to cc you on activity, or see the https://www.fltk.org/newsgroups.php page and use either the web interface or NNTP instructions.

3.1.3 Specific Goals

The specific goals of the FLTK are as follows:

- Develop a C++ GUI toolkit based upon sound object-oriented design principles and experience.
- Minimize CPU usage (fast).
- Minimize memory usage (light).
- Support multiple operating systems and windowing environments, including UNIX/Linux, macOS, Microsoft Windows, and X11, using the "native" graphics interfaces.
- Support OpenGL rendering in environments that provide it.
- Provide a graphical development environment for designing GUI interfaces, classes, and simple programs.
- Support UTF-8 text.
- Support printer rendering in environments that provide it.
- Support "schemes", "styles", "themes", "skinning", etc. to alter the appearance of widgets in the toolkit easily and efficiently. The purpose is to allow applications to tailor their appearance to the underlying OS or based upon personal/user preferences.
- Support newer C++ language features, such as templating via the Standard Template Library ("STL"), and certain Standard C++ library interfaces, such as streams. However, FLTK will not depend upon such features and interfaces to minimize portability issues.
- Support intelligent layout of widgets.

Many of these goals are satisfied by FLTK 1.1.x, and many complex applications have been written using FLTK on a wide range of platforms and devices.

3.1.4 Documentation

All widgets are documented using the Doxygen software; Doxygen comments are placed in the header file for the class comments and any inline methods, while non-inline methods should have their comments placed in the corresponding source file. The purpose of this separation is to place the comments near the implementation to reduce the possibility of the documentation getting out of sync with the code.
All widgets must have a corresponding test program which exercises all widget functionality and can be used to generate image(s) for the documentation. Complex widgets must have a written tutorial, either as full text or an outline for later publication
Starting with FLTK 1.3 Doxygen is used for HTML and PDF documentation.

3.1.5 Coding Standards

The following is a guide to the coding style that must be used when adding or modifying code in FLTK. Most of this should be obvious from looking at the code, but here it all is in one spot.

3.1.6 General Coding Style

The FLTK code basically follows the K&R coding style. While many of the developers are not entirely satisfied with this coding style, no one has volunteered to change all of the FLTK source code (currently about 54,000 lines of code!) to a new style.
The K&R coding style can be summarized with the following example code:

```
int function(int arg) {
  if (arg != 10) {
    printf("arg = %d\n", arg);
    return (0);
  } else {
    return 1;
  }
}
```

```
int function2(int arg) {
  for (int i = 0; i < arg; i++) {
    stuff();
  }
  while (something) {
    stuff();
  }
  switch (arg) {
    case 0:
      stuff_here();
      break;
    case 1: {
      int var;
      stuff_here();
      break;
    }
    case 2:
      stuff();
      /* FALLTHROUGH */
    case 3: simple_stuff1(); break;
    case 4: simple_stuff2(); break;
    default:
      break;
  }
  return (0);
}
```

To summarize:

- All curly braces must open on the same line as the enclosing statement and close at the same level of indentation.
- Each block of code must be indented 2 spaces.
- **Tabs are not allowed in source files**, please use only spaces for indenting.
- A space follows all reserved words.
- A space precedes and follows all operators except prefix and postfix operators (++i, j–, et al).
- Trailing whitespace (spaces and tabs) must be removed before committing. Please check your editor's documentation how to do this.

Note

Example Visual Studio Code: use `ctrl-k ctrl-x` before saving the file. On macOS: use `cmd-k cmd-x` instead.

3.1.7 Source File Documentation

Each source file must start with the standard FLTK header containing the description of the file, and FLTK copyright and license notice:

```
//
// Some descriptive text for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2020 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//
```

..or the equivalent comment block using the C or other comment delimiters appropriate for the source file language (shell, CMake, etc).

3.1.8 Doxygen Documentation (Comments)

FLTK 1.3 and up uses Doxygen with the JavaDoc comment style to document all classes, structures, enumerations, methods, and functions. Doxygen comments are **mandatory** for all FLTK header and source files, and no FLTK release will be made without complete documentation of public APIs. Here is an example of the Doxygen comment style:

```
/**
  The Foo class implements the Foo widget for FLTK.

  This description text appears in the documentation for
  the class and may include HTML tags as desired.
*/

class FL_EXPORT Foo : public Widget {
  int private_data_;

public:
  /**
    Creates a Foo widget with the given position and label.

    This description text appears in the documentation for the
    method's implementation.

    References to parameters \p X, \p Y, \p W, \p H are
    mentioned this way.

    \param[in] X,Y,W,H Position and size of widget
    \param[in] L Optional label (default is 0 for no label)
  */
  Foo(int X, int Y, int W, int H, const char *L = 0) {
    ..implementation here..
  }
};
```

Essentially, a comment starting with '/ $**$' and ending with '$*$ /' (both w/o spaces) before the class or method defines the documentation for that class or method. These comments should appear in the header file for classes and inline methods and in the code file for non-inline methods.
In addition to Doxygen comments, block comments must be used liberally in the code to describe what is being done. If what you are doing is not "intuitively obvious to a casual observer", add a comment! Remember, you're not the only one that has to read, maintain, and debug the code.
Never use C++ comments in C code files or in header files that may be included from a C program. (Otherwise builds on strict platforms like SGI will fail). Normally, fltk C files have ".c" and ".h" file extensions, and C++ have ".cxx" and ".H". Currently there are a few exceptions; filename.H and Fl_Exports.H both get interpreted by C and C++, so you must use C style comments in those.

3.1.9 General Developer Recommendations

Most important rule: **Put Doxygen comments where the code's implementation is.** This means don't put the docs with the prototypes in the .H file, unless that's where the code is implemented.

- `class`, `typedef`, `enum`, and `inline` docs go in the headers
- Most other docs go in the source files
- For doxygen syntax in C++ files, use:

  ```
  /** for standard doxygen comments */
  ///< for short single line post-declaration doxygen comments
  ```

- For doxygen syntax in C files, use:

  ```
  /**  for standard doxygen comments */
  /**< for short single line post-declaration doxygen comments */
  ```

- Use `\p` for parameters citation in the description
- Use `\param[in] xxx` and `\param[out] xxx` for input/output parameters.

- Don't use doxygen tags between the `\\htmlonly` and `\\endhtmlonly` pair of tags.
- When commenting out code or writing non-doxygen comments, be sure not to accidentally use doxygen comment styles, or your comments will be published! Beware doxygen recognizes other comment styles for itself:

```
/*! beware */
/*@ beware */
//! beware
//@ beware
```

There may be others. For this reason, follow all non-doxygen comment leaders with a space to avoid accidental doxygen parsing:

```
/* safe from doxygen */
// safe from doxygen
  ↑
  Space immediately after comment characters
```

Note: Several characters are 'special' within doxygen commments, and must be escaped with a backslash to appear in the docs correctly. Some of these are:

```
\<    -- disambiguates html tags
\>    -- ""
\&    -- ""
\@    -- disambiguates JavaDoc doxygen comments
\$    -- disambiguates environment variable expansions
\#    -- disambiguates references to documented entities
\%    -- suppresses prevention of auto-linking (escapes a single '%' that prevents auto-linking)
\\    -- escapes the escape character
```

Itemized lists can be specified two ways; both work, the first is preferred:
Preferred:
Here's a bullet list:

```
- Apples
- Oranges
```

Here's a numbered list:

```
-# First thing
-# Second thing
```

Old (HTML) style. Please use the style above instead.
Here's a bullet list:

```
<ul>
  <li> Apples</li>
  <li> Oranges</li>
</ul>
```

Here's a numbered list:

```
<ol>
  <li> First thing</li>
  <li> Second thing</li>
</ol>
```

3.1.10 Documenting Temporary Code or Issues

Temporary code and code that has a known issue MUST be documented in-line with the following (Doxygen) comment style:

```
/** \todo this code is temporary */
```

`\todo` items are listed by Doxygen making it easy to locate any code that has an outstanding issue or code that should be removed or commented out prior to a release.

3.1.11 Documenting Classes and Structures

Classes and structures start with a comment block that looks like the following:

```
/**
  A brief description of the class/structure.

  A complete description of the class/structure.
*/
class MyClass {
  ...implementation...
};
```

3.1.12 Documenting Enumerations

Enumerations start with a comment block that looks like the following:

```
/**
  A brief description of the enumeration.

  A complete description of the enumeration.
*/
enum MyEnum {
  ...implementation...
};
```

Each enumeration value must be documented in-line next to the corresponding definition as follows:

```
/* C++ STYLE */
enum MyEnum {
  BLACK,    ///< The color black.
  RED,      ///< The color red.
  GREEN,    ///< The color green.
  YELLOW,   ///< The color yellow.
  BLUE      ///< The color blue.
};
```

If the enum is included in a C file, be sure to use C style commenting:

```
/* C STYLE */
enum MyEnum {
  BLACK,    /**< The color black. */
  RED,      /**< The color red. */
  GREEN,    /**< The color green. */
  YELLOW,   /**< The color yellow. */
  BLUE      /**< The color blue. */
};
```

3.1.13 Documenting Functions and Methods

Functions and methods start with a comment block that looks like the following:

```
/**
  A brief description of the function/method.

  A complete description of the function/method.
  Optional passing mention of parameter \p a and \p out1.

  Optional code example goes here if needed:
  \code
  ..code showing how to use it..
  \endcode

  \param[in] a Description of input variable a
  \param[in] x,y Description of input variables x and y in one comment
  \param[out] out1 Description of output variable out1
  \param[out] out2 Description of output variable out2
  \return 0 on success, -1 on error
  \see other_func1(), other_func2()
*/
int my_function(int a, int x, int y, float &out1, float &out2) {
  ...implementation...
}
```

Some details on the above:

- **Parameters**

  Use `\param` to document function/method parameters using either of the following formats, the latter being preferred:

  ```
  \param var Some description
  \param[in|out] var Some description.
  ```

  Mention of parameters in docs should use "`\p varname`". (Use `\p` instead of `\a`)

  Note: Doxygen checks your `\param` variable names against the actual function signatures in your code. It does NOT check `\p` names for consistency.

- **Return Values**

  Use `\return` to document return values. Omit this if there is no return value.

- **Reference related methods**

  Use `\see` to help the reader find related methods. (Methods are sorted alphabetically by doxygen, so 'related' methods might not appear together)

  Locate `\see` references below `\param[]` and `\return` as shown in the above example.

- **Code examples**

  Use `\code` and `\endcode` when code examples are needed. Text within will be exempt from html and doxygen escape code parsing, so you don't have to escape characters <, >, &, etc. as you would normally.

  Be careful not to embed C style comments within `\code` and `\endcode` or it will break the outer doxygen comment block. (A good reason to always test build the code base before committing documentation-only mods.)

- **Where to put docs**

  Function/method documentation must be placed next to the corresponding code. (Rule: "Put the docs where the code implementation is.") Comments for in-line functions and methods are placed in the header file where they're defined.

3.1.14 Documenting Members and Variables

Members and variables can be documented in one of two ways; in block comment form:

```
/**
  A brief doxygen description of the member/variable.

  A complete description of the member/variable.
  More text goes here..
*/
int my_variable_;
```

or in the preferred form as in-line comments as follows:

```
int my_variable1_;    ///< C++ file's brief doxygen description of the member/variable
int my_variable2_;    /**< C file's brief doxygen description of the member/variable */
```

3.1.15 Methodology, Algorithms, Etc.

The goal of FLTK is to provide a robust GUI toolkit that is small, fast, and reliable. All public API functions and methods must be documented with the valid values for all input parameters - NULL pointers, number ranges, etc. - and no public API function may have undefined behaviors. Input validation should be performed only when the function or method is able to return an error to the caller.
When solving a particular problem, whether you are writing a widget or adding functionality to the library, please consider the following guidelines:

1. Choose the small, simple, easy-to-test algorithm over a more complex, larger one that is harder to debug and maintain.

2. Choose the fastest algorithm that satisfies #1
3. Break complex solutions into smaller, more manageable pieces.
4. If functionality can be separated from the main part of the FLTK library, separate it. The idea is to keep the FLTK "core" as small as possible so that programs use memory proportionate to their complexity rather than starting big.
5. Choose a general-purpose solution over a single-purpose solution, i.e. don't limit your design to what you think something will be used for.
6. Don't rely on functionality available on a particular platform or compiler; this ties in with #5.

3.1.16 C++ Portability

Since FLTK is targeted at platforms which often lack complete ISO C++ support or have limited memory, all C++ code in FLTK must use a subset of ISO C++. These restrictions shall be reviewed prior to each minor or major release of FLTK.

3.1.16.1 FLTK 1.1.x Restrictions

The following C++ features may be not used in FLTK 1.1.x code:

- Exceptions
- Namespaces
- Standard C++ headers and library
- Templates
- static_cast, dynamic_cast, const_cast

3.1.16.2 FLTK 1.3.x Restrictions

The following C++ features may be not used in FLTK 1.3.x code:

- Exceptions
- Namespaces
- Standard C++ headers and library
- Templates
- dynamic_cast

3.1.16.3 FLTK 1.4.x Restrictions

The following C++ features may be not used in FLTK 1.4.x code:

- Exceptions
- Namespaces
- Standard C++ headers and library
- Templates
- dynamic_cast

The `reinterpret_cast` keyword may be used but is not mandatory.
The `static_cast` and `const_cast` keywords should be used when casting pointers of different types.
The `dynamic_cast` keyword must not be used since run-time typing features may not be available at all times.

3.1.16.4 FLTK 1.5.x Restrictions

Todo This needs to be populated with the new restrictions/allowed c++ features.

3.1.16.5 Source File Naming

The current practice is to use an extension of ".c" for C source files, ".h" for C header files, ".cxx" for C++ source files, and ".H" for C++ header files in the "FL" directory (".h" otherwise.)

Todo Mention of the "fltk" namespace below is I think fltk2 specific, needs changes for 1.x.x

Function/Method/Variable Naming
All public (exported) functions and variables must be placed in the "fltk" namespace. Except for constructor and destructor methods, the names consist of lowercase words separated by the underscore ("_"), e.g. "fltk::some_↩ variable" and "text_color()". Private member variables of classes end with an extra underscore, e.g. "text_size_".

3.1.17 Structure/Class Naming

Todo The following needs modification; AFAIK we don't use the `fltk` namespace - I think that was an fltk2 thing - but we do recently make some use of `Fl` namespace (matt's recent tablet commits)

All public (exported) structures and classes must be placed in the "fltk" namespace and consist of capitalized words without the underscore, e.g. "fltk::SuperWidget".
Private members of classes must end with a trailing underscore ("_") and have corresponding public access methods without the underscore as applicable, e.g. "text_size_" and "text_size()".

3.1.18 Constant/Enumeration Naming

Todo The following needs modification regarding "fltk" namespace (this being an FLTK2 thing IIRC)

Public enumerations and constant variables must be placed inside the "fltk" namespace and consist of UPPERCASE WORDS separated by the underscore ("_"), e.g. "ALIGN_LEFT", "COLOR_RED", etc. Enumeration type names consist of capitalized words without underscores, e.g. "MyEnum". `#define` constants are prohibited aside from the include guard definitions.

3.1.19 Preprocessor Variables

File `config.h` and the C++ compilers define a few preprocessor variables that help organizing platform-specific code and control access to a few internal classes. Only code internal to the FLTK library can include the `config.h` header file. Thus, FLTK header files that are part of the public API must not, directly or indirectly, include `config.h`.

- `_WIN32` identifies the MS-Windows platform (both for the 32- and 64-bit versions). Note: FLTK 1.3.x and earlier used `WIN32` which had to be defined by the FLTK build system. FLTK 1.4.0 and later versions use `` `_WIN32` `` (with leading underscore) which should be defined by the build tools (preprocessor, compiler) on the Windows platform.
- `__CYGWIN__` is defined when FLTK runs on Windows but uses Cygwin's POSIX emulation features (cygwin1.dll). No longer officially supported but **may** work.
- `__APPLE__` identifies the macOS platform.
- `FLTK_USE_X11` is defined by `'FL/fl_config.h'` (since 1.4) when FLTK is built for the Xlib (aka `X`) graphics system. Thus, `FLTK_USE_X11` is defined
  - on Unix and Linux, if built for the X11 or the "hybrid" (X11 + Wayland) platform
  - on Windows, if `configure` used `--enable-cygwin` **and** `--enable-x11`.
    Note: FLTK 1.4 is the last minor FLTK version that supports `configure`.

- Note: Xlib-specific code was also often delimited without reference to the `FLTK_USE_X11` macro (thus without the requirement to include `FL/fl_config.h`) as follows:

```
#if defined(_WIN32)
  // Windows specific code ...
#elif defined(__APPLE__)
  // macOS specific code ...
#else
  // Xlib specific code ...
#endif
```

  This **must** no longer be done because it could be ambiguous since the introduction of Wayland.

- `FLTK_USE_WAYLAND` is defined by `'FL/fl_config.h'` (since 1.4) if FLTK is built with Wayland support (either with or without X11 support). This is only possible on Linux, Unix, or similar platforms.

- `USE_XFT` is defined by `config.h` if FLTK is built for X11 **and** CMake build option `FLTK_USE_XFT` has been enabled (`ON`). FLTK 1.4 and earlier: ... or the corresponding configure option `--use-xft`. It is set to 1 when the Xft library of scalable, anti-aliased fonts is used, and to 0 otherwise. Note: This variable is **not** accessible by user code (not in the public interface). It is considered an implementation detail.

- `FL_LIBRARY` is defined by all FLTK library build systems when the FLTK library itself is compiled. Application program developers **must not** define it when compiling their programs.

- `FL_INTERNALS` can be defined by application program developers to access certain internal FLTK library classes (e.g., the `Fl_X` class) and some global variables and definitions (macros) if they need it. APIs to these internal classes are highly subject to changes, though.

- `FL_DLL` is defined by the FLTK build system when building shared libraries (DLL's) with Visual Studio. Application program developers using Visual Studio and linking against the shared FLTK libraries (DLL's) built with Visual Studio must define this macro when compiling their source files. Note that this macro must be defined by the build system (VS project setup/properties) or on the compiler command line so it is "seen" by all included FLTK header files.

- `FL_DOXYGEN` is defined when the Doxygen program that builds the FLTK documentation processes the source code. This variable has two major uses:

  1. `#ifndef FL_DOXYGEN` / `#endif` allows to hide code from Doxygen.
  2. The Doxygen program does not define the platform-specific variables `__APPLE__` or `_WIN32` (even if it's run on macOS or Windows). Thus, platform-specific (say, macOS-specific) code must be bracketed as follows to be seen by Doxygen:

     ```
     #if defined(__APPLE__) || defined(FL_DOXYGEN)
     ... Doxygen-visible, macOS-specific code ...
     #endif
     ```

- `FL_ABI_VERSION` is used to allow developers to implement ABI breaking code in Patch Releases. Normally set to the default ABI version for each minor version (for instance 10400 for all 1.4.x releases), can be changed by users or devs with configure or CMake to enable ABI breaking features for testing or use by end users in static builds of FLTK.

  Note: This preprocessor variable was named FLTK_ABI_VERSION in FLTK 1.3.x and was renamed to FL_ABI_VERSION since FLTK 1.4.0.

  When set, the variable's value is expected to be the integer representation of the FLTK version number, where the Minor and Patch numbers are padded to two digits to allow for numbers 1 thru 99, e.g.

```
#define FL_ABI_VERSION 10401    // FLTK ABI version 1.4.1
      ..'1' is the major version (no padding; avoids octal issues)
      ..'04' is the minor version (2 digit padding)
      ..'01' is the patch version (2 digit padding)
```

ABI breaking features are by default '#ifdef'ed out with this variable during patch releases, and are merged in by developers during the next Minor Release.

Example: If the current patch release is 1.4.0, and the developer adds an ABI-breaking fix to what will be the next 1.4.1 release, then the new code would be implemented as:

```
#if FL_ABI_VERSION >= 10401  // FLTK 1.4.1, the next patch release #
... new ABI breaking code ...
#else
... old non-ABI breaking (default builds) ...
#endif
```

This variable solves several issues:

- Allows ABI breaking code to be implemented at any time by developers.
- Gets fixes into Git sooner, so users can see, test and access it.
- Testing ABI features during Patch Releases increases the stability of Minor Releases.
- Prevents devs having to defer ABI breaking code to the small window of time preceding Minor Releases.

3.1.20 Miscellaneous Coding Practices

3.1.20.1 Using switch() and 'FALLTHROUGH' Comments

When using switch/case statements, and your case statement does not end in break in order to fall through to the next case statement, a
`/* FALLTHROUGH */`
comment should be added where the break statement would be.

3.1.20.2 Useful Visual Studio C++ Macros (Windows)

Here's a list of Visual Studio compiler macros for main Visual Studio versions that can be used to conditionalize code based on the Visual Studio version:

Version	Product Name	_MSC_VER
18.0	Visual Studio 2026	1950
17.0	Visual Studio 2022	1930
16.0	Visual Studio 2019	1920
15.0	Visual Studio 2017	1910
14.0	Visual Studio 2015	1900
12.0	Visual Studio 2013	1800
11.0	Visual Studio 2012	1700
10.0	Visual Studio 2010	1600
9.0	Visual Studio 2008	1500
8.0	Visual Studio 2005	1400
7.1	Visual Studio 2003	1310
7.0	Visual Studio 7	1300
6.0	Visual Studio 6	1200
5.0	Visual Studio 5	1100

For details of other versions (patch releases like 15.8) see Microsoft docs.
https://learn.microsoft.com/en-us/cpp/overview/compiler-versions
Recommended usage:

```
#if defined(_MSC_VER) && (_MSC_VER <= 1300)  /* VS7 and older */
  ..
#else  /* _MSC_VER */
```

```
  ..
#endif /* _MSC_VER */
```

3.1.20.3 Useful macOS C++ Macros

These macOS macros may be used to write code compatible with different macOS versions. They are not necessary for standard FLTK user code but can be used if macOS specific code is needed for an application or inside the FLTK library.
Below is a list of operating system version compiler macros that can be used to conditionalize code based on the compile time (build host) and the runtime version of the macOS operating system.
These are made available by Apple's `AvailabilityVersions.h` inside the current / selected SDK.
outdated: For more info on these and other macros, see Apple's "TechNote 2064".

Todo Find current Apple / macOS docs for macOS version macros.

Version	Macro	Value	Product Name	Release
10.0	MAC_OS_X_VERSION_10_0	1000	Cheetah	2001
10.1	MAC_OS_X_VERSION_10_1	1010	Puma	2001
10.2	MAC_OS_X_VERSION_10_2	1020	Jaguar	2002
10.3	MAC_OS_X_VERSION_10_3	1030	Panther	2003
10.4	MAC_OS_X_VERSION_10_4	1040	Tiger	2005
10.5	MAC_OS_X_VERSION_10_5	1050	Leopard	2007
10.6	MAC_OS_X_VERSION_10_6	1060	Snow Leopard	2009
10.7	MAC_OS_X_VERSION_10_7	1070	Lion	2011
10.8	MAC_OS_X_VERSION_10_8	1080	Mountain Lion	2012
10.9	MAC_OS_X_VERSION_10_9	1090	Mavericks	2013
10.10	MAC_OS_X_VERSION_10_10	101000	Yosemite	2014
10.11	MAC_OS_X_VERSION_10_11	101100	El Capitan	2015
10.12	MAC_OS_X_VERSION_10_12	101200	Sierra	2016
10.13	MAC_OS_X_VERSION_10_13	101200	High Sierra	2017
10.14	MAC_OS_X_VERSION_10_14	101200	Mojave	2018
10.15	MAC_OS_X_VERSION_10_15	101200	Catalina	2019
11.0	MAC_OS_VERSION_11_0	110000	Big Sur	2020
12.0	MAC_OS_VERSION_12_0	120000	Monterey	2021
13.0	MAC_OS_VERSION_13_0	130000	Ventura	2022
14.0	MAC_OS_VERSION_14_0	140000	Sonoma	2023
15.0	MAC_OS_VERSION_15_0	150000	Sequoia	2024
26.0	MAC_OS_VERSION_26_0	260000	Tahoe	2025

Notes:

- The `Value` column is in decimal notation
- The macro name is MAC_OS_X_VERSION_* up to MacOS X 10.15 (Catalina)
- The macro name is MAC_OS_VERSION_* (without `'_X'`) since macOS 11 (Big Sur)
- There *is* a gap in version numbers from 15 to 26 because newer versions of macOS reflect the release year (+1), e.g. version 26 was released in 2025.
- The `Release` column (release year) has been included in this table for reference. We may use the release date to decide whether we support a particular macOS version.
- See README.macOS.md about which macOS versions FLTK supports.

Note: `MAC_OS_X_VERSION_MAX_ALLOWED` is defined automatically by the build system and represents the highest macOS version the library has been built for. The default is the SDK version used to build the library. It

can be modified by setting the CMake cache variable CMAKE_OSX_DEPLOYMENT_TARGET to a specific macOS version.
Example:

- CMAKE_OSX_DEPLOYMENT_TARGET = 15.4
- -> MAC_OS_X_VERSION_MAX_ALLOWED = 150400

Recommended usage and requirements (include files):

```
#define FL_INTERNALS 1        // required to define "internal" macros, define before:
#include <FL/platform.H>      // defines the MAC_OS_*VERSION_* macros (see table above)

#if (MAC_OS_X_VERSION_MAX_ALLOWED >= MAC_OS_X_VERSION_10_5)
if (fl_mac_os_version >= 100500) {
  // macOS 10.5 and newer..
} else
#else
{
  // macOS 10.4 and older..
}
#endif
```

This way, code compiled before and after macOS 10.5 will be able to run on computers running macOS versions before and after 10.5. Modify version numbers as applicable for specific macOS versions.

3.1.20.4 Useful GNU C++ Macros

Todo To be done.

3.1.21 Source Files

Each source file shall be placed in a sub-directory corresponding to the software sub-system it belongs to ("fltk", "OpenGL", etc.).

Todo rework paragraph above, explain "sub-systems" and directory names

Source files shall be documented and formatted as described in the Coding Standards section. To remain compatible with case-insensitive filesystems, no two filenames and/or directory names shall differ only by the case of the letters in the name.
C source files shall have an extension of ".c". C++ source files shall have an extension of ".cxx". Header files shall have an extension of ".h". Public FLTK 1.x headers (i.e. those in or below the `FL/` directory) shall have an extension of ".H", private headers in source directories shall have an extension of ".h".

Why use the ".cxx" extension?
C++ source files can have any of the following extensions on various platforms: ".C", ".cc", ".cpp", ".cxx". Only the ".cxx" extension is universally recognized by C++ compilers as a C++ source file - ".C" is not usable on macOS and Windows, ".cc" is not usable on Windows, and ".cpp" is historically considered C preprocessor output on UNIX. Since not all make programs handle C++ source files with the ".cxx" extension, the FLTK build system explicitly defines makefile rules for compiling C++ source files with an extension of ".cxx".

IDE/compiler support source files (project files, workspaces, makefiles, etc.) shall have extensions as required by the IDE/compiler tool.

Todo remove this paragraph in FLTK 1.5, remove this 'todo' in 1.4.

Header files must utilitize so-called "guard" definitions to prevent multiple inclusion. The guard definitions are named using the full path in the FLTK source tree, e.g.:

```
FL/Fl.H becomes _FL_Fl_H_
fluid/foo.h becomes _fluid_foo_h_
```

Any non-alphanumeric (letters and numbers) characters are replaced with the underscore (_) character, and leading and trailing underscores are added to limit global namespace pollution.

3.1.22 Build System (CMake)

The FLTK build system uses CMake to tailor the library to the local operating system. Since FLTK 1.4 the primary and recommended build system is CMake. The older autoconf/configure builds are no longer supported as of FLTK 1.5.x.
To improve portability, Makefiles must not make use of the unique features offered by GNU make. See the Makefile Standards section for a description of the allowed make features and makefile guidelines.

Todo Remove this paragraph in 1.5,

Additional GNU build programs such as GNU automake and GNU libtool must not be used. GNU automake produces non-portable Makefiles which depend on GNU-specific extensions, and GNU libtool is not portable or reliable enough for FLTK.
Note: Starting with FLTK 1.4.0 we do no longer bundle IDE files for Microsoft Windows (Visual Studio) and mac↩OS (Xcode). IDE environments (project files) can be generated with CMake. See README.CMake.txt for more information about using CMake.

3.2 The Wayland backend for its developer

This chapter describes how the Wayland backend of FLTK works from a developer's viewpoint.

3.2.1 Introduction to Wayland

Wayland usage involves communication via a Unix domain socket between a client application and another process called the Wayland compositor which creates, moves and resizes windows and displays their graphical content. Diverse Wayland compositors exist which follow diverse strategies to size, decorate and position windows on the screens. Compositors follow either the client-side decoration (CSD) rule where client apps draw window titlebars, or the server-side decoration (SSD) rule where the compositor draws titlebars. FLTK supports both CSD and SSD compositors. It uses a library called `libdecor` charged of determining whether a CSD or a SSD compositor is active, and of drawing titlebars in the first case.
Wayland is divided in various protocols that a given compositor may or may not support, although they all support the `core` protocol. Each protocol adds functionality not available in the core protocol and determines a well defined API the client must follow to gain such functionality. `Wayland Explorer` lists all protocols and details their APIs. Wayland compositors evolve in time by supporting more protocols or newly defined protocols. The core protocol allows a client app to discover what protocols the connected compositor supports. Protocols can be stable, which means they have a defined API that will not change but can be expanded, or unstable. For example, mapping a window on a screen is not done by the core protocol but by the `xdg shell` protocol which is stable. The names of symbols used by unstable protocols always begin with letter 'z'. For example, FLTK uses unstable protocol `Text input` to support CJK input methods; its symbol names begin with `zwp_text_input_v3`.
Wayland makes intensive use of the *listener* mechanism. A listener is a small array of pointers to FLTK-defined callback functions associated to a Wayland-defined object; Wayland calls these functions when defined events occur (more at Listeners below).
Wayland differs noticeably from X11 in that rendering is left to clients: Wayland provides no drawing API. Instead, Wayland provides objects of type `struct wl_buffer` which encapsulate a memory array of pixel values shared between the client and the compositor. The client app is expected to draw to that memory buffer with whatever means it chooses, and to instruct the compositor to map those pixels to the screen when the buffer is ready for display. The Wayland platform of FLTK draws with the Cairo library to `Fl_Window`'s and `Fl_Image_Surface`'s, and with OpenGL to `Fl_Gl_Window`'s.
Wayland differs also from X11 in that the position of a window in the display is completely hidden to the client app. This prevents function `Fl_Window::position()` from having any effect on a top-level window. Wayland also prevents a client app from knowing whether a window is minimized: `Fl_Window::show()` has no effect on a minimized window. Subwindows can be positioned as usual relatively to their parent window. Wayland allows to create popup windows positioned relatively to a previously mapped other window. This allows FLTK to position adequately menu and tooltip windows (see Menu windows and other popups). FLTK uses also popups for the small, yellow windows that display the new scale factor value when it's changed: these are created as short-lived popups centered above `Fl::first_window()`.
Wayland offers a means to handle lists of linked records. It defines type `struct wl_list` and a few macros (`wl_list_init()`, `wl_list_for_each()`, `wl_list_insert()`, `wl_list_for_each_safe()`, `wl_list_remove()`) to manage linked lists. Memory records put in these lists must have the same type and

contain a member variable of type `struct wl_list` used to link records together and often named 'link'. Access to such a list is possible memorizing a value of type `struct wl_list` computed by macro `wl_list_init()`. Macro `wl_list_for_each(arg1, arg2, arg3)` allows to run through all list elements with:

- `arg1` is a pointer variable to the type of elements of the linked list;
- `arg2` is the address of a variable of type `struct wl_list` identifying the targeted list;
- `arg3` is the name of the member variable of these elements used to link them together.

For example, `wl_list_for_each()` can be used as follows to scan the linked list of all displays of the system (see Fl_Wayland_Screen_Driver::output):

```
Fl_Wayland_Screen_Driver::output *output;
Fl_Wayland_Screen_Driver *scr_driver = (Fl_Wayland_Screen_Driver*)Fl::screen_driver();
wl_list_for_each(output, &(scr_driver->outputs), link) {
  // ... work with output, an item of the linked list of all displays in the system ...
}
```

Overall, and ignoring for now OpenGL usage, FLTK interacts with Wayland as follows :

- When opening the display: FLTK calls `Fl::add_fd()` in `FL_READ` mode to associate a callback function called `wld_socket_callback()` to the socket connecting the client and the compositor.
- Client to compositor: FLTK calls C functions which operate the core Wayland protocol and things related to keyboard support and window decoration. These functions have names beginning with `wl_`, `xkb_` or `libdecor_` and belong to the `libwayland-client.so`, `libwayland-cursor.so`, `libxkbcommon.so` and `libdecor.so` shared libraries. FLTK calls also several other functions which operate other used Wayland protocols. These have names beginning with `xdg_`, `zwp_text_input_v3`, `zxdg_toplevel_decoration_`, `gtk_shell1_`, `gtk_surface1_`, `wp_cursor_shape_`, `xdg_activation_`, `zwp_tablet_v2` and `zwp_primary_selection_`. FLTK never writes itself to the socket; this is done inside the functions just mentioned.
- Compositor to client: the callback function `wld_socket_callback()` runs when there are data to read in the socket; it calls `libdecor_dispatch()` which reads available data from the socket, interprets these data and calls corresponding listeners.

The core protocol defines also a number of mostly opaque structures whose names begin with `wl_`. FLTK defines a few structures holding Wayland-related data. The names of FLTK-defined structures don't begin with `wl_`. For example, `struct wld_window` (see wld_window) is used to store all Wayland-specific data associated to a mapped Fl_Window.

3.2.2 Building libfltk as a Wayland client

Classes `Fl_Wayland_Window_Driver`, `Fl_Wayland_Screen_Driver`, `Fl_Wayland_Graphics_Driver`, `Fl_Wayland_Copy_Surface_Driver`, `Fl_Wayland_Image_Surface_Driver` and `Fl_Wayland_Gl_Window_Driver` and files `Fl_Wayland_Pen_Driver.cxx` and `fl_wayland_platform_init.cxx` contain all the Wayland-specific code of the FLTK library. This code is located at `src/drivers/Wayland/` in the FLTK source tree. A single C++ source file generally contains all the code of a given class. The code for class `Fl_Wayland_Screen_Driver`, however, is split in two source files, `Fl_Wayland_Screen_Driver.cxx` and `fl_wayland_clipboard_dnd.cxx` that contains all code related to copy, paste and drag-and-drop. Furthermore, class `Fl_Unix_System_Driver` is used by both the Wayland and the X11 FLTK platforms. File FL/fl_config.h defines preprocessor variables `FLTK_USE_WAYLAND` and `FLTK_USE_CAIRO`.

The public C API to Wayland, xkb, EGL and libdecor libraries are obtained with

```
#include <wayland-client.h>
#include <wayland-cursor.h>
#include <xkbcommon/xkbcommon.h>
#include <xkbcommon/xkbcommon-compose.h>
#include <linux/input.h> // for BTN_LEFT, BTN_RIGHT, BTN_MIDDLE
#include "../../../libdecor/build/fl_libdecor.h"
#if HAVE_GL
#  include <wayland-egl.h>
#  include <EGL/egl.h>
#endif // HAVE_GL
```

as necessary.

File `README.Wayland.txt` details what software packages are needed on Debian-based, Fedora and FreeBSD systems for FLTK to use Wayland.

Wayland protocols are packaged as XML files accompanied by a utility program, `wayland-scanner`, able to generate a header file and a necessary glue C source file from a given XML file. For example, for FLTK to use the `XDG shell` protocol, these commands are run at library build time to generate, using file `xdg-shell.xml`, a .c file (`xdg-shell-protocol.c`) that will be compiled into `libfltk` and a header file (`xdg-shell-client-protocol.h`) that the FLTK code will include:

```
PROTOCOLS=`pkg-config --variable=pkgdatadir wayland-protocols`
wayland-scanner private-code $PROTOCOLS/stable/xdg-shell/xdg-shell.xml xdg-shell-protocol.c
wayland-scanner client-header $PROTOCOLS/stable/xdg-shell/xdg-shell.xml xdg-shell-client-protocol.h
```

Similar operations are performed for FLTK to use protocols `XDG dialog windows`, `XDG activation`, `XDG decoration`, `Cursor shape`, `Text input`, `Primary selection`, `GTK Shell` and `Tablet`.

3.2.3 The hybrid Wayland/X11 platform

The Wayland platform of FLTK is normally a two-legged hybrid able to use either Wayland or X11 and to choose between these possibilities at run-time, without any change to the client application. The Wayland/X11 hybrid is essentially a version of the FLTK library containing both all Wayland-specific and all X11-specific code. That's reflected in file `FL/fl_config.h` which defines both `FLTK_USE_WAYLAND` and `FLTK_USE_X11`. This creates the constraint that FLTK's Wayland-specific and X11-specific source code cannot use the same type name for different purposes or the same symbol name. That is why function `fl_xid(const Fl_Window*)` is deprecated in FLTK 1.4 and replaced by `fl_wl_xid()` for Wayland and `fl_x11_xid()` for X11. Also, global variable `Window fl_window` is not used by the Wayland platform which instead uses `static struct wld_window *Fl_Wayland_Window_Driver:: wld_window`. The FLTK library contains also a short source file, `fl_wayland_platform_init.cxx`, that determines, at startup time, whether the app will run as a Wayland or as an X11 client. Function `attempt_wayland()` therein performs this choice as follows :

- if the app defines a global bool variable called `fl_disable_wayland` and this variable is true, the X11 leg is chosen;
- if environment variable FLTK_BACKEND is defined to string "wayland", the Wayland leg is chosen;
- if environment variable FLTK_BACKEND is defined to string "x11", the X11 leg is chosen;
- otherwise, a connection to a Wayland compositor is attempted; if it's successful, the Wayland leg is chosen; if it's not, the X11 leg is chosen.

The first condition listed above is meant to facilitate transition to FLTK 1.4 of source code written for FLTK 1.3 and containing X11-specific code : it's enough to put

```
FL_EXPORT bool fl_disable_wayland = true;
```

anywhere in the source code, for the app to run with 1.4, using the x11 leg of the hybrid platform, without any other change in the source code nor to the application's environment.

Function `attempt_wayland()` must be called before the very first platform-dependent operation FLTK performs to make sure that this operation gets done the Wayland or the X11 way, as appropriate. That's why 5 locations of the FLTK source code call `attempt_wayland()`: `Fl_Graphics_Driver::newMainGraphicsDriver()`, `Fl_Screen_Driver::newScreenDriver()`, `Fl_Window_Driver::newWindowDriver(Fl_Window*)`, `Fl_Image_Surface_Driver::newImageSurfaceDriver()`, and `Fl::Pen::newPenDriver()`.

In special situations, such as with embedded systems equipped with the Wayland software but lacking the X11 library, it's possible to build the FLTK library such as it contains only the Wayland backend. This is achieved building FLTK with `cmake -DFLTK_BACKEND_X11=OFF`. In that case, FL/fl_config.h does not define `FLTK_USE_X11`.

The rest of this chapter describes what happens when the Wayland leg has been chosen.

3.2.4 Listeners

A Wayland 'listener' is a small array of pointers to FLTK-defined callback functions associated to a Wayland-defined object. A call to a specific Wayland function with a name of the form `wl_XXX_add_listener` associates the Wayland object and its listener, usually right after creation of this object. After defined events have occurred, the Wayland compositor sends appropriate commands to the client through the socket; the event loop detects the availability of data in the socket and calls function `wld_socket_callback()` which calls `libdecor_dispatch()`; this last function calls the appropriate member of the listener and transmits relevant information to the client app as parameters of this call. For example, this code:

```
static void surface_enter(......) { ...... } // called when a surface enters a display
static void surface_leave(......) { ...... } // called when a surface leaves a display

static struct wl_surface_listener surface_listener = {
  surface_enter,
  surface_leave,
};

some_pointer_type pter_to_data;
struct wl_surface *my_wl_surface;
my_wl_surface = wl_compositor_create_surface(scr_driver->wl_compositor);
wl_surface_add_listener(my_wl_surface, &surface_listener, pter_to_data);
```

creates a Wayland object of type `struct wl_surface` (roughly, a window), and associates it with a 2-member listener called `surface_listener`. After this, Wayland is expected to call the 2 listener members, `surface_enter` or `surface_leave`, each time `my_wl_surface` will enter or leave, respectively, a display. The arguments of these calls, not detailed here, allow the member functions to identify which surface enters or leaves which display. The `wl_surface_add_listener()` call above also associates `pter_to_data` to `my_wl_surface` as *user data*. The `wl_surface` object's "user data" can be obtained later calling function `wl_surface_get_user_data()`.
Wayland function `wl_proxy_get_listener()` returns a pointer to a Wayland object's listener provided that object is transmitted cast to type `struct wl_proxy *`. This gives a handy way to distinguish FLTK-created Wayland objects from objects of other origin: the listener of an FLTK-created object is a known FLTK listener. For example, function `Fl_Wayland_Window_Driver::surface_to_window()` uses this possibility calling `wl_proxy_get_listener( (struct wl_proxy *)wl_surface )` for any object of type `struct wl_surface`: if that object was created as in the example above, this call returns a pointer to FLTK's `surface_listener` static variable.

3.2.5 Opening a Wayland connection

Establishing a Wayland connection requires environment variable `XDG_RUNTIME_DIR` to be defined and to point to a directory containing a socket connected to a Wayland compositor. This variable is usually set by the login procedure of Wayland-friendly desktops. Which socket-file to use within that directory is determined as follows:

- the client may call `Fl::display(const char *display_name)` before `fl_open_display()` runs or use the `-display` command line argument and transmit there the socket name;
- environment variable `WAYLAND_DISPLAY` can be defined to the socket name;
- otherwise, `"wayland-0"` is used.

Which socket is selected determines the compositor used by the client application: that at the other end of the socket.
Establishing the connection begins with a call to `wl_display_connect(const char *display_name)`. That call is done inside function `attempt_wayland()` mentioned before with a NULL argument, or when a non default Wayland display name is specified as explained above. That call returns a `struct wl_display` pointer or NULL in case of failure. Such NULL return is the hint that allows the FLTK display opening procedure of the Wayland/X11 hybrid to recognize when Wayland access is not possible and to fallback to X11. If the call is successful, its non-NULL return is assigned to class variable `Fl_Wayland_Screen_Driver::wl_display`.
The rest of the work is done in function `Fl_Wayland_Screen_Driver::open_display_platform()`.
A call to `wl_registry_add_listener()` associates a 2-member listener, whose 1st member, `registry_handle_global()`, will be called by Wayland a number of times to indicate each time a protocol supported by the compositor or a system feature such as displays and keyboards.
FLTK runs this code to receive calls to `registry_handle_global()`:

```
static void sync_done(void *data, struct wl_callback *cb, uint32_t time) {
  *(struct wl_callback **)data = NULL;
  wl_callback_destroy(cb);
}

static const struct wl_callback_listener sync_listener = {
  sync_done
};

struct wl_callback *registry_cb = wl_display_sync(wl_display);
wl_callback_add_listener(registry_cb, &sync_listener, &registry_cb);
while (registry_cb) wl_display_dispatch(wl_display);
```

A pointer to an object of type `struct wl_callback` created by function `wl_display_sync()` is assigned to variable `registry_cb`. Then a 1-member listener is attached to this object. Wayland will run this listener's member function, `sync_done()`, after all calls to `registry_handle_global()` have occurred. Function

sync_done() sets to null variable registry_cb and destroys the wl_callback. Finally, function wl_display_dispatch() is called as long as variable registry_cb is not null. Thus, when sync_done() runs, FLTK has received all due calls to registry_handle_global().
The prototype of function registry_handle_global is:

```
static void registry_handle_global(void *user_data, struct wl_registry *wl_registry,
          uint32_t id, const char *interface, uint32_t version)
```

Each time Wayland calls registry_handle_global(), interface and version give the name and version of a supported Wayland protocol or a system feature. Here is the list of the interface value for all protocols and system features relevant for FLTK:

Interface	How FLTK uses this interface
wl_compositor	create wl_surface objects
wl_subcompositor	create subwindows
wl_shm	create shared memory pools and buffers
wl_seat	create the unique "seat"
wl_data_device	support of copy/paste/drag-n-drop
wl_output	received once for each display
xdg_wm_base	create mapped windows
xdg_activation_v1	programmatically activate a window
xdg_wm_dialog_v1	create dialogs related to other windows
wp_cursor_shape_manager_v1	set cursor images using enumerated cursors
gtk_shell1	indicates Mutter is in use + titlebar gestures
weston_desktop_shell	indicates Weston is in use
org_kde_plasma_shell	indicates KDE/Plasma is in use
zwp_text_input_manager_v3	interface with Text Input Methods
zxdg_decoration_manager_v1	select between CSD and SSD modes
zwp_tablet_manager_v2	support of graphics tablets
zwp_primary_selection_device_manager_v1	middle-click paste of current selection

Wayland compositors typically support several other protocols (e.g., zxdg_output_manager_v1) that FLTK does not use.
Each time registry_handle_global runs with an interface from the table above, FLTK calls wl_registry_bind() which returns a pointer to a Wayland structure that will be the client's access point to the corresponding Wayland protocol or system feature. This pointer is stored in a dedicated member variable of the unique Fl_Wayland_Screen_Driver object of an FLTK app, or of another object accessible from this object. For example, when interface equals "wl_compositor", the value returned by wl_registry_bind() is stored as member wl_compositor of the Fl_Wayland_Screen_Driver object. Function registry_handle_global() also identifies whether the Mutter, Weston, KWin or Wayfire compositor is connected and stores this information in static member variable Fl_Wayland_Screen_Driver::compositor. Other compositors (e.g., sway, labwc) are not specifically identified by FLTK and store value unspecified in member variable compositor.
Wayland calls registry_handle_global() with its parameter interface equals to "wl_output" once for each screen connected to the system. Each time, a pointer to a Wayland object of type struct wl_output is obtained by a call to wl_registry_bind() and an FLTK record of type Fl_Wayland_Screen_Driver::output is created. This FLTK record is added to the end of the linked list of known screens that starts at member outputs of the Fl_Wayland_Screen_Driver object. A 4-member listener is associated to the struct wl_output by function wl_output_add_listener(). The 3rd member of this 4-function listener, output_done(), runs after all initialization steps of the screen have completed and turns to true member done of the record of type Fl_Wayland_Screen_Driver::output. Function sync_done() mentioned above therefore also calls wl_display_dispatch() until the done member of all Fl_Wayland_Screen_Driver::output records are true. Overall, after return from function sync_done(), FLTK has been made aware of all optional protocols and features of its connected Wayland compositor, and has initialized all screens of the system.
Finally, function libdecor_get_fd() is called to obtain the file descriptor of the Wayland socket and a call to Fl::add_fd() makes FLTK listen to this descriptor in FL_READ mode and associates function wld_socket_callback() from file Fl_Wayland_Screen_Driver.cxx with it. This function reads and interprets data available from the file descriptor, and calls corresponding listeners.

The event loop is run by function `Fl_Unix_System_Driver::wait()` which is used by both the Wayland and X11 FLTK backends. Among various tasks, this function waits for data arriving on the file descriptors FLTK is listening. Overall, the Wayland backend's event loop differs from that of the X11 backend only in the callback function handling data read from the Wayland connection socket and in overridden functions `Fl_Wayland_Screen_Driver::poll_or_select_with_delay()` and `Fl_Wayland_Screen_Driver::poll_or_select()`.

3.2.6 Wayland windows and surfaces

Wayland defines objects called surfaces of type `struct wl_surface`. A Wayland surface "has a rectangular area which may be displayed on zero or more displays, present buffers, receive user input, and define a local coordinate system". In short, surface is the name Wayland uses for a window. Buffers allow the client app to define the graphical content of surfaces (see Wayland buffers). FLTK creates a surface each time an Fl_Window is show()'n calling function `wl_compositor_create_surface()`. Static member function `Fl_Wayland_Window_Driver::surface_to_window(struct wl_surface *)` gives the `Fl_Window*` corresponding to the surface given in argument.
FLTK recognizes 4 mutually exclusive kinds of surfaces :

- DECORATED are toplevel windows with a titlebar;
- UNFRAMED are toplevel windows without titlebar;
- POPUP correspond to menus and tooltips;
- SUBWINDOW correspond to an Fl_Window embedded in another Fl_Window.

Function `Fl_Wayland_Window_Driver::makeWindow()` creates all these surfaces, creates for each a record of type `struct wld_window` (see wld_window), and stores the window kind in member variable `kind` of this record. Member variable `xid` of the window's `Fl_X` record stores the adress of this record.
Except for SUBWINDOW's, each surface needs a Wayland object of type `struct xdg_surface` used to make it become a mapped window and stored in member `xdg_surface` of the window's wld_window record. For DECORATED windows, this object is created inside libdecor and transmitted to FLTK by function `libdecor_frame_get_xdg_surface()`. For UNFRAMED and POPUP windows, it's created by function `xdg_wm_base_get_xdg_surface()`. Finally, each surface is also associated to one more Wayland object whose type varies with the window's kind. These explain this part of the wld_window record:

```
union {
  struct libdecor_frame *frame; // created for DECORATED by libdecor_decorate()
  struct wl_subsurface *subsurface; // created for SUBWINDOW by wl_subcompositor_get_subsurface()
  struct xdg_popup *xdg_popup; // created for POPUP by xdg_surface_get_popup()
  struct xdg_toplevel *xdg_toplevel; // created for UNFRAMED by xdg_surface_get_toplevel()
};
```

Except for SUBWINDOW's, each surface is associated to a 'configure' function that Wayland calls one or more times when the window is going to be mapped on the display. The 'configure' function of DECORATED surfaces is `handle_configure()` which is the 1st member of a 4-member listener named `libdecor_frame_iface` associated to a decorated window when it's created calling `libdecor_decorate()`. Finally, a call to `libdecor_frame_map()` triggers the process of mapping the newly created DECORATED surface on a display. Wayland calls `handle_configure()` twice during this process. The first `handle_configure()` run allows to set the window's `xdg_surface` object which is returned by function `libdecor_frame_get_xdg_surface()`. FLTK distinguishes the first from the second run of `handle_configure()` by looking at the `xdg_surface` member variable that's NULL at the beginning of the 1st run and not NULL later. Wayland calls `handle_configure()` also during operations such as resizing, minimizing (see below). With the help of a few calls to libdecor functions, FLTK obtains in this function all needed information about the size and state of the mapped window. The 'configure' functions of UNFRAMED and POPUP surfaces are `xdg_surface_configure()`, `xdg_toplevel_configure()` and `popup_configure()`. The mapping process of these surfaces is triggered by a call to `wl_surface_commit()`. These 'configure' functions transmit effective window size information to FLTK. Also, they are where the window's `Fl_Window_Driver::wait_for_expose_value` member variable is set to 0 to indicate that the window has been mapped to display. **Caution:** there are some small differences between how and when the various Wayland compositors call `handle_configure()`.
When a decorated window changes size, whatever the cause of it, Wayland calls `handle_configure()` which sets member variable `Fl_Wayland_Window_Driver::in_handle_configure` to true and calls the window's virtual `resize()` function which ultimately runs `Fl_Wayland_Window_Driver::resize()`

which calls Fl_Group::resize() to perform FLTK's resize operations and `Fl_Wayland_Graphics_Driver::buffer_release()` to delete the existing window buffer that's not adequate for the new window size. At the end of the run of `handle_configure()`, `in_handle_configure` is set back to false. When the window size change is caused by the app itself calling the window's `resize()` function, `Fl_Wayland_Window_Driver::in_handle_configure` is false. This allows `Fl_Wayland_Window_Driver::resize()` to detect that Wayland needs be informed of the desired size change, which gets done by a call to `libdecor_frame_commit()`. Wayland later calls `handle_configure()` and events described above unfold.
Wayland does not provide a way to control where the compositor should map a `DECORATED` or `UNFRAMED` window in a display. When the system has several displays, it's not even possible to control on what display should the compositor map a new window, except for a fullscreen window. That is done inside function `handle_configure()` which calls `libdecor_frame_set_fullscreen()` for DECORATED windows and inside function `xdg_toplevel_configure()` which calls `xdg_toplevel_set_fullscreen()` for UNFRAMED. The `struct wl_output` pointer for the targeted display is transmitted as 2nd argument of these calls.

3.2.7 Menu windows and other popups

Menu windows, tiny menu title windows, and tooltip windows are implemented using Wayland's popup mechanism which allows to position a popup window relatively to a previously mapped window, itself a popup or another kind of window, with the restriction that any popup must overlap or at least touch that other window. Member function `Fl_Wayland_Window_Driver::makeWindow` calls member function `Fl_Wayland_Window_Driver::process_menu_or_tooltip` to create all popups.
This function gets called after FLTK has computed using a given algorithm the desired (x,y) position of the popup window's top-left corner, using coordinates centered on the top-left corner of the toplevel window from which the popup originates. This algorithm is able to prevent popups from being positioned beyond the screen borders under the assumption that the position of a toplevel window inside a screen is known. While this assumption holds for other platforms, it does not for the Wayland platform. The FLTK code for the Wayland platform therefore modifies the algorithm that FLTK uses to compute the position of menu windows. The key information used by this algorithm is obtained by virtual member function `Fl_Window_Driver::menu_window_area` which computes the coordinates of the rectangle where menu windows are allowed to be positioned. Under other platforms, this function just returns the origin and size of the work area of the screen in use. In contrast, the Wayland platform handles two situations differently :

- For menu windows that are not taller than the display in use, the Wayland-overridden member function `Fl_Wayland_Window_Driver::menu_window_area` returns large negative origin and large width and height values. This lets the standard FLTK algorithm position the menu relatively to its window of origin without concern about screen limits, and relies on Wayland's constraint mechanism described below to prevent the menu from going beyond these limits, without FLTK having to know where they are.
- Menu windows taller than the screen where they are mapped need special handling described in detail in a comment above the source code of function `Fl_Wayland_Window_Driver::process_menu_or_tooltip`.

Function `Fl_Wayland_Window_Driver::process_menu_or_tooltip` first computes `origin_win`, pointer to the `Fl_Window` relatively to which the popup is to be positioned. Window `origin_win` is the parent menu window when the popup is a sub-menu; it's the tiny windowtitle when the popup is a menu with a title; otherwise, it's the window containing the point of origin of the popup. An object of type `struct xdg_positioner` created by function `xdg_wm_base_create_positioner()` is used to express the rules that will determine the popup position relatively to `origin_win` as follows:

- Function `xdg_positioner_set_anchor_rect()` determines a rectangle in `origin_win` relatively to which the popup is to be positioned. When the popup to be created is a menu window spawned by an Fl_Menu_Bar, that rectangle is the full area of the menu title window. Otherwise, that rectangle is an adequately located point.
- Function `xdg_positioner_set_size()` sets the popup size.
- The `xdg_positioner_set_anchor(positioner, XDG_POSITIONER_ANCHOR_BOTTOM_LEFT);` and `xdg_positioner_set_gravity(positioner, XDG_POSITIONER_GRAVITY_BOTTOM_RIGHT);` calls position the popup so that its top-left corner is initially below and at right of the bottom-left corner of the `origin_win` 's anchor rectangle.

- The call to xdg_positioner_set_offset() further changes the popup vertical position.
- The call to xdg_positioner_set_constraint_adjustment() uses constraint flags XDG_POSITIONER_CONSTRAINT_ADJUSTMENT_SLIDE_X and XDG_POSITIONER_CONSTRAINT_ADJUSTMENT_SLIDE_Y which mean that the compositor will move the popup horizontally and vertically if its initial position would make it expand beyond the edges of the screen. Furthermore, flag XDG_POSITIONER_CONSTRAINT_ADJUSTMENT_FLIP_Y is added when the popup is a menu window spawned by an Fl_Menu_Bar; this has the popup flipped above the Fl_Menu_Bar if there's not enough screen room below it for the popup.
- Finally, a call to function xdg_surface_get_popup() creates the popup accounting for position rules listed above. The positioner is then deleted by xdg_positioner_destroy(), a listener is associated to the popup surface with xdg_popup_add_listener(), and a call to wl_surface_commit() triggers the mapping of the popup on the display.

Overall, the expected coordinates of the top-left corner of the popup relatively to origin_win are popup_x, popup_y. They are memorized in a record of FLTK-defined type struct win_positioner that's associated to the popup listener. When the compositor maps the popup, function popup_configure, the first element of the popup listener, runs and receives as arguments the coordinates of the popup top left and its size. These values account for the positioning constraints of the popup which may have moved it to avoid screen borders. This function can therefore detect whether constraints applied have modified the effective popup location in comparison to the expected coordinates which are available as member variables of the struct win_positioner record mentioned above. That's key to the handling by FLTK of tall menu windows.
Groups of popups containing a menutitle, the associated menuwindow, and optionally a submenu window and that don't belong to an Fl_Menu_Bar are mapped in a different order: the menuwindow is mapped first, and the menutitle is mapped second above it as a child popup. Function Fl_Window_Driver::is_floating_title() detects when such a menutitle is created, static member variable previous_floatingtitle is assigned the value of this menutitle, and the menutitle is mapped only after the menuwindow has been mapped, as a child of it. This positions better the popup group in the display relatively to where the popup was created.

3.2.8 Fl_Wayland_Graphics_Driver and Fl_Cairo_Graphics_Driver

The Wayland platform of FLTK uses an Fl_Wayland_Graphics_Driver object for all its on-screen drawing operations. This object is created by function Fl_Graphics_Driver::newMainGraphicsDriver() called by Fl_Display_Device::display_device() when the library opens the display. New Fl_Wayland_Graphics_Driver objects are also created for each Fl_Image_Surface and each Fl_Copy_Surface used, and deleted when these objects are deleted.
Class Fl_Wayland_Graphics_Driver derives from class Fl_Cairo_Graphics_Driver which implements all the FLTK drawing API for a Cairo surface. Function Fl_Wayland_Graphics_Driver::cairo_init() creates the Cairo surface used by each Fl_Wayland_Graphics_Driver object by calling cairo_image_surface_create_for_data() for the window's or offscreen's draw_buffer (see below).
Class Fl_Cairo_Graphics_Driver is also used by the X11 leg of the hybrid Wayland-X11 platform because this leg draws to the display with an Fl_X11_Cairo_Graphics_Driver object which derives from class Fl_Cairo_Graphics_Driver. Finally, Fl_Cairo_Graphics_Driver is also used, in the form of an object from its derived class Fl_PostScript_Graphics_Driver, when the hybrid Wayland-X11 platform draws PDF or PostScript, or when the classic X11 platform uses Pango and draws PDF or PostScript. This happens when classes Fl_PDF_File_Surface, Fl_PostScript_File_Device and Fl_Printer are used.

3.2.9 Wayland buffers

Wayland uses buffers, objects of type struct wl_buffer, to draw to surfaces. In principle, one or more buffers can be associated to a surface, and functions wl_surface_attach() and wl_surface_commit() are called to first attach one such buffer to the surface and then inform the compositor to map this buffer's graphics content on the display. Wayland buffers can use various memory layouts. FLTK uses WL_SHM_FORMAT_ARGB8888, which is the same layout as what Cairo calls CAIRO_FORMAT_ARGB32.
FLTK calls function Fl_Wayland_Window_Driver::make_current() before drawing to any Fl_Window. Member buffer of this Fl_Window's struct wld_window (see wld_window) is NULL when the window has just been created or resized. In that case, FLTK calls Fl_Wayland_Graphics_Driver::create_wld_buffer() which returns a pointer to a struct wld_buffer containing

- a Wayland buffer, member `wl_buffer`;
- a Cairo image surface, created by a call to `Fl_Wayland_Graphics_Driver::cairo_init()`.

Each of these two objects encapsulates a byte array of the same size and the same memory layout destined to contain the Fl_Window's graphics. The Cairo image surface object is where FLTK draws. The Wayland buffer is what Wayland maps on the display. FLTK copies the Cairo surface's byte array to the Wayland buffer's byte array before beginning the mapping operation. If `width` and `height` are a window's dimensions in pixels,

```
int stride = cairo_format_stride_for_width(CAIRO_FORMAT_ARGB32, width);
int size = stride * height;
```

give `size`, the common size of both byte arrays.
The effective creation of the `wl_buffer` object is delayed until function `Fl_Wayland_Graphics_Driver::buffer_commit()` gets called. Section Buffer factory below details how FLTK creates `wl_buffer` objects. The `struct Fl_Wayland_Graphics_Driver::wld_buffer` (see wld_buffer) contains a pointer to the byte array of the Cairo image surface (member `draw_buffer.buffer`), information about the Wayland buffer (members `wl_buffer` and `data`), the common size of the Cairo surface's and Wayland buffer's byte arrays (member `draw_buffer.data_size`), and other information. A pointer to this `struct Fl_Wayland_Graphics_Driver::wld_buffer` is memorized as member `buffer` of the Fl_Window's wld_window. All drawing operations to the Fl_Window then modify the content of the Cairo image surface.
Function `Fl_Wayland_Window_Driver::flush()` is in charge of sending FLTK graphics data to the display. That is done by calling function `Fl_Wayland_Graphics_Driver::buffer_commit()` which creates the `struct wl_buffer` object calling `create_shm_buffer()` if that was not done before, copies the byte array of the Cairo surface to the Wayland buffer's starting memory address, and calls functions `wl_surface_attach()` and `wl_surface_commit()`. Before calling Fl_Window::flush(), FLTK has computed a damaged region. If that region is not null, `Fl_Wayland_Graphics_Driver::buffer_commit()` copies only the damaged parts of the Cairo surface to the Wayland buffer and calls function `wl_surface_damage_buffer()` for these parts to inform the compositor of what parts of the surface need its attention.

Wayland buffer deletion

Each wld_buffer record contains boolean member `in_use` which is set to `true` just before the buffer gets committed, and boolean member `released` which is set to `true` when FLTK no longer needs the buffer and calls `Fl_Wayland_Graphics_Driver::buffer_release()`. FLTK's buffer-creating function, `Fl_Wayland_Graphics_Driver::create_shm_buffer()`, attaches a 1-member listener to each buffer which Wayland calls after a commit operation to indicate the client is allowed to re-use the buffer. This listener's member function, `buffer_release_listener()`, turns to false member `in_use` of the buffer's wld_buffer record. Since the two events 'FLTK no longer needs the buffer' and 'the client is allowed to re-use the buffer' can arrive in any order, FLTK deletes the `struct wl_buffer` object by running `do_buffer_release()` only after both events happened, that is, when `in_use` is `false` and `released` is `true`. That's why function `do_buffer_release()` is called by both functions `Fl_Wayland_Graphics_Driver::buffer_release()` and `buffer_release_listener()`.

3.2.10 Throttling window redraws

FLTK uses Wayland's synchronization mechanism to make sure any committed `wl_buffer` is not changed while the compositor is using it and to refrain from calling `wl_surface_commit()` more frequently than the system can process it. Firstly, as seen above, Wayland calls function `buffer_release_listener()` when the client is free to reuse or destroy a given `wl_buffer`. FLTK won't change or destroy a committed `wl_buffer` before that call. Second, this 2-step mechanism prevents Wayland clients from committing new buffer states too frequently:

- `Fl_Wayland_Graphics_Driver::buffer_commit()` first calls function `wl_surface_frame()` to obtain a pointer to a `struct wl_callback` object and stores it as member `frame_cb` of the surface's wld_window. Then it calls `wl_callback_add_listener()` to associate this object to the FLTK-defined, callback function `surface_frame_done()`. It next calls `wl_surface_commit()`. Together, these 3 calls instruct Wayland to start mapping the buffer content to the display and to call `surface_frame_done()` later, when it will have become ready for another mapping operation.
- Later, `surface_frame_done()` runs and destroys the `wl_callback` object by function `wl_callback_destroy()` and sets member `frame_cb` to NULL.

Member variable draw_buffer_needs_commit of the wld_buffer is also important in this mechanism : it informs FLTK that the graphics buffer has changed and needs being committed. This variable is turned true every time a graphics operation changes the buffer content and turned false when the buffer gets committed.
This procedure ensures that FLTK never calls wl_surface_commit() before the compositor becomes ready for a new commit because Fl_Wayland_Window_Driver::flush() calls Fl_Wayland_Graphics↩_Driver::buffer_commit() only if frame_cb is NULL. If it's not NULL, the exact content of function surface_frame_done() :

```
static void surface_frame_done(void *data, struct wl_callback *cb, uint32_t time) {
  struct wld_window *window = (struct wld_window *)data;
  wl_callback_destroy(cb);
  window->frame_cb = NULL;
  if (window->buffer && window->buffer->draw_buffer_needs_commit) {
    Fl_Wayland_Graphics_Driver::buffer_commit(window);
  }
}
```

has the effect that when the mapping operation eventually completes, Wayland runs surface_frame_done(), which, if the buffer's draw_buffer_needs_commit member is true, calls Fl_Wayland_Graphics_↩Driver::buffer_commit() anew. The net result is that the screen shows the most recent surface content.
This synchronization mechanism is also used when performing an interactive window resize operation. During such operation, the compositor informs the client an interactive resize is being performed and sends window resize commands at high rate (∼60 Hz) to the client via the socket. Libdecor turns on flag LIBDECOR_WINDOW↩_STATE_RESIZING to inform the client, and runs function handle_configure() for each received resize command. Before calling Fl_Group::resize() and later Fl_Window::draw(), handle_configure() tests whether window->frame_cb is NULL. When it's not because a previous resize operation is being performed, the current resize command is skipped. At the end of the interactive resize, flag LIBDECOR_WINDOW_STATE_RESIZING is off and Wayland sends a final resize command which is not skipped. Overall, this ensures the client program resizes its window as frequently as it can without falling behind resize commands sent by the compositor.
To account for a bug in Mutter (issue #878), the window->frame_cb object is not created when a toplevel window is being resized and is entirely covered by one subwindow.

Progressive window drawing

FLTK supports progressive drawing when an app calls function Fl_Window::make_current() at any time and then calls the FLTK drawing API. This is made possible in function Fl_Wayland_Window_Driver::make_↩current() with

```
// to support progressive drawing
if ( (!Fl_Wayland_Window_Driver::in_flush_) && window->buffer && (!window->frame_cb)
    && (!wait_for_expose_value) ) {
  Fl_Wayland_Graphics_Driver::buffer_commit(window);
}
```

Thus, buffer_commit() runs only when frame_cb is NULL. If an app rapidly performs calls to Fl_Window::make_current() and to drawing functions, FLTK will copy draw_buffer to the Wayland buffer and instruct Wayland to map it to the display when frame_cb is NULL which means that the compositor is ready to start performing a mapping operation. This occurs when the progressive drawing operation begins. Later, frame_cb is generally found non NULL when Fl_Wayland_Window_Driver::make_current() runs because the compositor is busy processing the previous Wayland buffer. When the compositor has completed this processing, the client app runs surface_frame_done() which, provided member variable draw_buffer↩_needs_commit is true, calls Fl_Wayland_Graphics_Driver::buffer_commit(). This makes the compositor map the Wayland buffer in its new, more advanced, state.
An example of progressive drawing is given by FLTK's mandelbrot test app. When maximized, this app can be seen to progressively fill its window from top to bottom by blocks of lines, each block appearing when the compositor is ready to map a new buffer. When the compositor is not ready, the app does not block but continues computing and drawing in memory but not on display more lines of the desired Mandelbrot graph.

3.2.11 Buffer factory

Wayland names *buffer factory* a software procedure that constructs objects of type struct wl_buffer, that is, Wayland buffers, for use by a client application. FLTK creates a wl_buffer object each time a non-GL Fl_Window is mapped on a display or resized. FLTK triggers the creation of a Wayland buffer as follows:

```
Fl::flush() calls, if the window is damaged:
   Fl_Wayland_Window_Driver::flush() calls, if the window's frame_cb is NULL:
     Fl_Wayland_Graphics_Driver::buffer_commit() calls, if the window's buffer->wl_buffer is NULL:
       Fl_Wayland_Graphics_Driver::create_shm_buffer() creates a Wayland buffer.
```

Member function `Fl_Wayland_Graphics_Driver::create_shm_buffer()` follows this 3-step procedure to create one or more *shared memory pools* and to construct Wayland buffers from them:

- Libdecor function `libdecor_os_create_anonymous_file(off_t size)` creates an adequate file and mmap's it. This file lives in RAM because it is created by function `memfd_create()`. FLTK sets this file size to 10 MB unless the size of the buffer to be created is larger; in that case the anonymous file is sized to twice the buffer size.

- Wayland function `wl_shm_create_pool()` shares this mmap'ed memory with the Wayland compositor and returns an object of type `struct wl_shm_pool`, a shared memory pool, which encapsulates this memory. A record of type `struct Fl_Wayland_Graphics_Driver::wld_shm_pool_data` is created and associated to the newly created `wl_shm_pool` by `wl_shm_pool_set_user_data()`. This record stores the starting address (`pool_memory`) and size (`pool_size`) of the pool's encapsulated memory. The record also contains member `buffers` of type `struct wl_list` which stores the access point to the linked list of `wl_buffer` objects that will be created from the `wl_shm_pool`.

- A variable named `chunk_offset` represents the offset within the pool's shared memory available for the buffer being constructed. It equals 0 when the pool has just been created and is updated as detailed below each time a buffer is created from the pool. A record of type `struct Fl_Wayland_Graphics_Driver::wld_buffer` is created. This record will contain (member `wl_buffer`) the address of a `wl_buffer` object that's created by function `wl_shm_pool_create_buffer()`. This `wl_buffer` object encapsulates a section of a given size of the pool's shared memory beginning at offset `chunk_offset` in it. Quantity `pool_memory + chunk_offset` is therefore the address of the beginning of the mmap'ed memory section encapsulated by this `wl_buffer`. Member `shm_pool` of the newly constructed `Fl_Wayland_Graphics_Driver::wld_buffer` object is set to the address of the current `wl_shm_pool` object. This record is added to the head of the linked list of current pool's buffers by a call to `wl_list_insert()`. At that point, a `struct Fl_Wayland_Graphics_Driver::wld_buffer` record is part of the linked list of all such records corresponding to `wl_buffer` objects created from the same `wl_shm_pool` object, and member `shm_pool` of this record gives the address of this `wl_shm_pool`. When a new `struct Fl_Wayland_Graphics_Driver::wld_buffer` record is to be created,

```
struct wld_shm_pool_data *pool_data =
    (struct wld_shm_pool_data *)wl_shm_pool_get_user_data(pool);
struct Fl_Wayland_Graphics_Driver::wld_buffer *record = wl_container_of(pool_data->buffers.next,
record, link);
int chunk_offset = ((char*)record->data - pool_data->pool_memory) + record->data_size;
```

  gives the offset within the current pool's mmap'ed memory available for a new `wl_buffer`. Macro `wl_container_of()` gives the address of a record belonging to a linked list of records of the same type.

A window's `wl_buffer` is re-filled by graphics data and committed each time the window gets redrawn, and is set to be destroyed by function `Fl_Wayland_Graphics_Driver::buffer_release()` when `Fl_Window::hide()` runs or the window is resized. When the `wl_buffer` is no longer in use, function `do_buffer_release()` gets called as explained above. It destroys the `wl_buffer` with `wl_buffer_destroy()`, and removes the corresponding `Fl_Wayland_Graphics_Driver::wld_buffer` record from the linked list of buffers from the same `wl_shm_pool`. Since new `Fl_Wayland_Graphics_Driver::wld_buffer` records are added at the head of the linked list, and since the record at the head of this list is used to compute the offset within the pool's mmap'ed memory available for a new `wl_buffer`, destruction of the last created `wl_buffer` allows to re-use the destroyed buffer's pool's memory for a new `wl_buffer`.
When function `do_buffer_release()` finds the list of buffers from a given pool empty, two situations can occur. 1) This pool is the current pool. Its mmap'ed memory will be re-used from offset 0 to create future `wl_buffer` objects. 2) This pool is not current. It gets destroyed with `wl_shm_pool_destroy()`, the pool's mmap'ed memory is munmap'ed, and the pool's associated `struct wld_shm_pool_data` is freed. In situation 1) above, the next `wl_buffer` to be created can need more memory than the current pool's memory size. If so, the current pool gets destroyed and replaced by a new, larger pool.
If the sum of `chunk_offset` plus the buffer size is larger than the current pool's size when function `create_shm_buffer()` is called, `chunk_offset` is reset to 0, and a new `wl_shm_pool` object is created and used by FLTK's "buffer factory". This mechanism allows to access new mmap'ed memory when `chunk_offset` reaches the end of the previous mmap'ed section.
Wayland uses also `wl_buffer` objects to support cursors. FLTK uses the "buffer factory" described here when creating custom cursors (see custom-cursor) with function `Fl_Wayland_Window_Driver::set_cursor(const Fl_RGB_Image *,...)` which calls `create_shm_buffer()` via `set_cursor_4args()`, `custom_offscreen()` and `create_wld_buffer()`. In contrast, standard shaped-cursors

(e.g., FL_CURSOR_INSERT) use their own "buffer factory" inside Wayland functions such as `wl_cursor_theme_get_cursor()`. Therefore, the fact that the `wl_buffer` objects behind standard cursors are never destroyed doesn't prevent disused `struct wl_shm_pool` objects from being freed because those buffers come from a distinct "buffer factory". The "buffer factory" described here is also used by function `offscreen_from_text()` when displaying dragged text in a DnD operation.

3.2.12 Displays and HighDPI support

Wayland uses the concept of *seat* of type `struct wl_seat` which encompasses displays, a keyboard, a mouse, and a trackpad. Although Wayland may be in principle able to deal with several seats, FLTK's Wayland platform is conceived for one seat only. That seat may contain one or more displays, which Wayland calls *outputs*, of type `struct wl_output`.
As written above, function `registry_handle_global()` discovers the available seat at start-up time. This function also associates a listener to each display connected to the system by calling function `wl_output_add_listener()`. This listener's member functions run at program startup when Wayland discovers its displays (see Opening a Wayland connection). Member `output_mode` runs also when the display is resized and member `output_scale` also when the Wayland scale factor (see below) is changed. FLTK defines type `struct` Fl_Wayland_Screen_Driver::output to store display size and scaling information. One such record is created for each display. These records are put in a `struct wl_list` accessible from member `outputs` of the single `Fl_Wayland_Screen_Driver` object.
FLTK uses 2 distinct scaling parameters for each display:

- `int wld_scale;`. This member variable of the `struct` Fl_Wayland_Screen_Driver::output record typically equals 1 for standard, and 2 for HighDPI displays. The effect of value `n` of variable `wld_scale` is that 1 Wayland graphics unit represents a block of `nxn` pixels. Another effect is that a drawing buffer for a surface of size WxH units contains `W * n * H * n * 4` bytes. Member function `output_scale()` mentioned above sets this value for each system's display at startup time. Member function `Fl_Wayland_Graphics_Driver::buffer_commit()` informs the Wayland compositor of the value of `wld_scale` calling `wl_surface_set_buffer_scale()` which is enough to make FLTK apps HighDPI-aware. Under the gnome and KDE desktops, this parameter is visible in the "Settings" app, "Displays" section, "Scale" parameter which is 200% on HighDPI displays.

- `float gui_scale;`. This other member variable is where FLTK's own GUI scaling mechanism with Ctrl/+/-/0/ keystrokes and with environment variable FLTK_SCALING_FACTOR operates: when FLTK is scaled at 150%, `gui_scale` is assigned value 1.5. Function `Fl_Wayland_Screen_Driver::scale(int n, float f)` assigns value `f` to the `gui_scale` member variable of display # `n`. This variable is used by function `Fl_Wayland_Window_Driver::make_current()` when it calls `Fl_Wayland_Graphics_Driver::set_buffer()` that scales the graphics driver by this factor with `cairo_scale()`.

Overall, an FLTK object, say an Fl_Window, of size `WxH` FLTK units occupies `int(W * gui_scale) * wld_scale x int(H * gui_scale) * wld_scale` pixels on the display.
When an `Fl_Window` is to be show()'n, `Fl_Wayland_Window_Driver::makeWindow()` creates a `struct wl_surface` with `wl_compositor_create_surface()` and associates it calling `wl_surface_add_listener()` with a 2-member listener called `surface_listener` encharged of managing as follows the list of displays where this `wl_surface` will map. The `Fl_Window` possesses an initially empty linked list of displays accessible at member `outputs` of the window's wld_window record. When the `Fl_Window`, or more exactly its associated `struct wl_surface` is mapped on a display, member `surface_enter()` of `surface_listener` runs. This function adds the display where the surface belongs to the end of the linked list of displays for this surface. When a surface is dragged or enlarged across the edge of a display in a multi-display system and expands on a second display, `surface_enter()` runs again, and this surface's list of displays contains 2 items. When a surface leaves a display, member `surface_leave()` of `surface_listener` runs. It removes that display from the surface's list of displays. Each time the first item of a surface's list of displays changes, function `change_scale()` is called and applies that display's `gui_scale` value to that surface calling `Fl_Window_Driver::screen_num(int)`. When a window is unmapped by function `Fl_Wayland_Window_Driver::hide()`, the surface's list of displays is emptied.

Fractional scaling

The KWin and gnome compositors allow to use *fractional scaling* that can take values between 100% and 400% that are not a multiple of 100%. Wayland implements this rendering all `wl_surface`'s as if the scaling had the next value above that is a multiple of 100% (e.g., 175% --> 200%), and downsizing them to the desired fractional scale value at the compositing stage. Seen from FLTK, everything runs with `wld_scale` having an integer value (1, 2, 3 or 4).
Recent gnome versions natively support fractional scaling. Older ones require to use these commands to make them accept/refuse fractional scaling:

```
gsettings set org.gnome.mutter experimental-features "['scale-monitor-framebuffer']"
gsettings reset org.gnome.mutter experimental-features
```

3.2.13 Mouse and trackpad handling

FLTK receives information about mouse and pointer events via a 'listener' made up of 5 pointers to functions which Wayland calls when events listed in table below occur. These functions receive from Wayland enough information in their parameters to generate corresponding FLTK events, that is, calls to `Fl::handle(int event_type, Fl_Window *)`.

listener function	called by Wayland when	resulting FLTK events
`pointer_enter`	pointer enters a window	FL_ENTER
`pointer_leave`	pointer leaves a window	FL_LEAVE
`pointer_motion`	pointer moves inside a window	FL_MOVE
`pointer_button`	state of mouse buttons changes	FL_PUSH, FL_RELEASE
`pointer_axis`	trackpad is moved vertically or horizontally	FL_MOUSEWHEEL

`pointer_listener` is installed by a call to function `wl_pointer_add_listener()` made by function `seat_capabilities()` which is itself another 'listener' made up of 2 function pointers

```
static struct wl_seat_listener seat_listener = {
  seat_capabilities,
  seat_name
};
```

installed by a call to function `wl_seat_add_listener()` made by function `registry_handle_global()` when it receives a `"wl_seat"` interface.

Handling middle mouse button clicks on window titlebars

The gnome desktop, via its `gnome-tweaks` application, allows to determine what happens when a middle mouse button click occurs on a window titlebar. To obey this setting, FLTK implements part of the `GTK Shell` protocol as follows. Mutter, gnome's Wayland compositor, declares its support of the `GTK Shell` protocol calling `registry_handle_global()` with its `interface` argument equal to `"gtk_shell1"`. FLTK initializes then member variable `seat->gtk_shell` of type `struct gtk_shell1*`.
Member functions of `pointer_listener` mentioned above run for all mouse events on all `wl_surface` objects. The table above describes what these functions do for mouse events on FLTK-created `wl_surface` objects. But they also run for the libdecor-created `wl_surface` objects corresponding to window titlebars. Thus, member function `pointer_enter()` runs when the mouse enters a titlebar. It calls `Fl_Wayland_Screen_Driver::event_coords_from_surface()` which calls `Fl_Wayland_Window_Driver::surface_to_window()` which, as mentioned above, can distinguish FLTK-created from non FLTK-created `wl_surface` objects. This allows `pointer_enter()` to identify the entered surface as a titlebar and to assign static global variable `gtk_shell_surface` with the titlebar's `wl_surface` when the mouse enters a titlebar. Similarly, member function `pointer_leave()` sets `gtk_shell_surface` to NULL when the mouse leaves this titlebar. When there's a click on a titlebar, member function `pointer_button()` runs this code

```
if (gtk_shell_surface && state == WL_POINTER_BUTTON_STATE_PRESSED && button == BTN_MIDDLE) {
  struct gtk_surface1 *gtk_surface = gtk_shell1_get_gtk_surface(seat->gtk_shell, gtk_shell_surface);
  gtk_surface1_titlebar_gesture(gtk_surface, serial, seat->wl_seat, GTK_SURFACE1_GESTURE_MIDDLE_CLICK);
  gtk_surface1_release(gtk_surface);
  return;
}
```

which ensures that what `gnome-tweaks` has assigned to middle-click events is executed. At this point in time, FLTK obeys what `libdecor` decides for right-click (display the window menu) and double-click (maximize the window) events on titlebars which may diverge from `gnome-tweaks` settings.

3.2.14 Tablet and pen handling

FLTK implements the Tablet protocol to support graphics tablets unless CMake option FLTK_OPTION_PEN↩_SUPPORT was turned Off at library build time.

3.2.15 Wayland cursors

When the running compositor supports it, FLTK uses protocol Cursor shape that offers an optional way to set cursor images using enumerated cursor shapes. With this protocol, it's enough to call function wp_cursor↩_shape_device_v1_set_shape() inside pointer_enter() to set the cursor shape when the pointer enters a given surface.
If the running compositor doesn't support this protocol or to define custom cursor shapes, FLTK uses another approach as follows.
Wayland defines types struct wl_cursor and struct wl_cursor_theme to hold cursor-related data. FLTK uses function init_cursors() from file Fl_Wayland_Screen_Driver.cxx to obtain the 'cursor theme' name using function libdecor_get_cursor_settings() of library libdecor. Function wl_↩cursor_theme_load() then returns a pointer to an object of type struct wl_cursor_theme stored in member variable cursor_theme of the Fl_Wayland_Screen_Driver::seat record. Function init_cursors() is itself called by a 'listener' called seat_capabilities() installed when function registry_handle_↩global() receives a "wl_seat" interface, at program startup. It is also called when the value of the Wayland scaling factor changes: output_done() calls try_update_cursor() calls init_cursors(). Function output_done() belongs to a 'listener' installed when function registry_handle_global() receives a "wl_output" interface.
Each time Fl_Window::cursor(Fl_Cursor) runs, FLTK calls Fl_Wayland_Window_Driver↩::set_cursor(Fl_Cursor) which calls wl_cursor_theme_get_cursor() to set the current cursor shape to one of the standard shapes from the Fl_Cursor enumeration. This Wayland function selects a cursor shape based on the current wl_cursor_theme object and a cursor name and returns a pointer to a struct wl_cursor. Under the gnome desktop, cursor names are the files of directory /usr/share/icons/↩XXXX/cursors/ where XXXX is the 'gnome cursor theme' (default= Adwaita). For example, what FLTK calls FL_CURSOR_INSERT corresponds to file xterm therein. The full correspondance between Fl_Cursor values and names of files therein is found in function Fl_Wayland_Window_Driver::set_cursor(Fl↩_Cursor). FLTK stores in member variable default_cursor of the Fl_Wayland_Screen_Driver::seat record a pointer to the currently used wl_cursor object, and the current Fl_Cursor value in member standard↩_cursor_ of the Fl_Wayland_Window_Driver object.
Finally, function do_set_cursor() of file Fl_Wayland_Screen_Driver.cxx makes the system pointer use the current wl_cursor object to draw its shape on screen. That's done with a call to wl_pointer_set_↩cursor() and a few other functions.

Custom cursor shapes

To support custom cursors, FLTK presently uses a non-public type, struct cursor_image, defined in file Fl_Wayland_Window_Driver.cxx as follows:

```
struct cursor_image {
  struct wl_cursor_image image;
  struct wl_cursor_theme *theme;
  struct wl_buffer *buffer;
  int offset;
};
```

This definition has been copied to the FLTK source code from file wayland-cursor.c of the Wayland project source code because it's not accessible via Wayland header files. It shows that a pointer to a cursor_image object can also be viewed as a pointer to the embedded struct wl_cursor_image object, this one being part of the public Wayland API. It also shows that a struct cursor_image object has an associated struct wl_buffer object used to contain the cursor's graphics.
Function Fl_Wayland_Window_Driver::set_cursor(const Fl_RGB_Image *rgb, int hotx, int hoty) gives FLTK support of custom cursor shapes. It calls Fl_Wayland_Window_Driver↩::set_cursor_4args() that creates a cursor_image object, allocates the corresponding wl_buffer by a call to Fl_Wayland_Graphics_Driver::create_shm_buffer() via custom_offscreen() and create_wld_buffer() and draws the cursor shape into that buffer using the offscreen-drawing method of FLTK.
The public type struct wl_cursor is essentially an array of wl_cursor_image objects and a name:

```
struct wl_cursor {
  unsigned int image_count;
```

```
  struct wl_cursor_image **images;
  char *name;
};
```

Function `Fl_Wayland_Window_Driver::set_cursor_4args()` also creates a `struct wl_cursor` object containing a single `wl_cursor_image`, which is in fact the `cursor_image`. Finally, a `struct Fl_Wayland_Window_Driver::custom_cursor` (see wld_window) is allocated and used to memorize the `struct wl_cursor` and the cursor's image and hotspot. A pointer to this `struct Fl_Wayland_Window_Driver::custom_cursor` object is stored in member `custom_cursor` of the window's wld_window.
Function `Fl_Wayland_Window_Driver::set_cursor_4args()` is also called when a window with a custom cursor is moved between distinct displays or when a display is rescaled to adapt the cursor size to the new display's scale factor.
Static member function `Fl_Wayland_Window_Driver::delete_cursor()` is used to delete any custom cursor shape. This occurs when a window associated to a custom cursor is un-mapped and when such a window gets associated to a standard cursor or to a new custom cursor.

3.2.16 Keyboard support

The "Mouse handling" section above mentioned function `seat_capabilities()` that Wayland calls when the app discovers its "seat". Presence of flag `WL_SEAT_CAPABILITY_KEYBOARD` in argument `capabilities` of this function indicates that a keyboard is available. In that case, a call to `wl_seat_get_keyboard()` returns a pointer stored in member `wl_keyboard` of the Fl_Wayland_Screen_Driver::seat object, and a call to `wl_keyboard_add_listener()` installs a 6-member listener of type `struct wl_keyboard_listener`. These 6 FLTK-defined, callback functions are used as follows.
1) Function `wl_keyboard_keymap()` runs when the app starts and also if the keyboard layout is changed during run-time. It allows initialization of access to this keyboard. Noticeably, member `xkb_state` of type `struct xkb_state*` of the current Fl_Wayland_Screen_Driver::seat record is adequately initialized.
2-3) Functions `wl_keyboard_enter()` and `wl_keyboard_leave()`, called when focus enters and leaves a surface, send `FL_FOCUS` and `FL_UNFOCUS` events to the `Fl_Window` object corresponding to this surface.
4) Function `wl_keyboard_key()` runs each time a keyboard key is pressed or released. Its argument `key`, to which 8 must be added, provides the keycode via function `xkb_state_key_get_one_sym()` and then the corresponding text via function `xkb_state_key_get_utf8()` which is put in `Fl::e_text`. Then, a few calls to functions whose name begin with `xkb_compose_` are necessary to support dead and compose keys. Finally a call to `Fl::handle()` sends an `FL_KEYDOWN` or `FL_KEYUP` event to the appropriate `Fl_Window`. Also, function `wl_keyboard_key()` uses global variable `std::vector<int> key_vector` to record all currently pressed keys. This is the base of the implementation of `Fl_Wayland_Screen_Driver::event_key(int)`.
5) Function `wl_keyboard_modifiers()` runs when a modifier key (e.g., shift, control) is pressed or released. Calls to functions `xkb_state_update_mask()` and `xkb_state_mod_name_is_active()` allow FLTK to set `Fl::e_state` adequately.
6) Function `wl_keyboard_repeat_info()` runs before the compositor sends any key press event and informs FLTK of the delay and keyboard repetition rate set by the desktop environment.

3.2.17 Support of text input methods

When the connected Wayland compositor supports text input methods, function `registry_handle_global()` gets called with its `interface` argument equal to `zwp_text_input_manager_v3_interface.name`. The following call to `wl_registry_bind()` returns a pointer to type `struct zwp_text_input_manager_v3` that is stored as member `text_input_base` of the `Fl_Wayland_Screen_Driver` object.
Later, when function `seat_capabilities()` runs, `text_input_base` is found not NULL, which triggers a call to function `zwp_text_input_manager_v3_get_text_input()` returning a value of type `struct zwp_text_input_v3 *` and stored as member `text_input` of the Fl_Wayland_Screen_Driver::seat object. Next, a call to `zwp_text_input_v3_add_listener()` associates this `text_input` with a 6-member listener of type `struct zwp_text_input_v3_listener`. These 6 FLTK-defined, callback functions are used as follows.
1-2) Functions `text_input_enter()` and `text_input_leave()` run when text input enters or leaves a surface.
3-4) Functions `text_input_preedit_string()` and `text_input_commit_string()` are called

when the text input method prepares the client app to later insert 'marked' text or regular text, respectively. Complex text input often begins by inserting temporary text which is said to be 'marked' before replacing it with the text that will stay in the document. FLTK underlines or selects marked text to distinguish it from regular text.
5) Function `text_input_done()` runs when it's time to send either regular or marked text to the client app. This is done by function `send_text_to_fltk()` which uses static variables `current_pre_edit`, `pending_pre_edit` and `pending_commit` to determine the sent text.
6) Function `text_input_delete_surrounding_text()` has no effect at present, without this preventing input methods that have been tested with FLTK from working satisfactorily.
It's necessary to inform the running text input method of the current location of the insertion point in the active surface. This information allows the input method to map its auxiliary window close to the insertion point. The flow of information on this topic is as follows:

- The two FLTK widgets supporting text input, Fl_Input_ and Fl_Text_Display, transmit to FLTK the window coordinates of the bottom of the current insertion point and the line height each time they change calling function `fl_set_spot()`.
- fl_set_spot() calls the platform override of virtual member function `Fl_Screen_Driver::set_spot()`. Under Wayland, this just calls `Fl_Wayland_Screen_Driver::insertion_point_location(int x, int y, int height)` which calls `zwp_text_input_v3_set_cursor_rectangle()` to inform the text input method about the surface position and size of the insertion point and also memorizes this information in static member variables of class `Fl_Wayland_Screen_Driver`.
- Callback function `text_input_enter()` calls `Fl_Wayland_Screen_Driver::insertion_point_location(int *x, int *y, int *height)` which gives it the stored position information, and then calls `zwp_text_input_v3_set_cursor_rectangle()` to inform the text input method about the position of the insertion point.

3.2.18 Interface with libdecor

FLTK uses a library called `libdecor` to determine whether the Wayland compositor uses CSD or SSD mode, and also to draw window titlebars when in CSD mode (see libdecor:). `Libdecor` is conceived to be present in a shared library linked to the Wayland client application which itself, and if the running Wayland compositor uses CSD mode, loads another shared library intended to draw titlebars in a way that best matches the Desktop. As of late 2023, libdecor contains two titlebar-drawing plugins:

- `libdecor-gtk` intended for the Gnome desktop;
- `libdecor-cairo` for other situations.

On recent Linux distributions, FLTK uses the system `libdecor-0.so` shared library available via packages `libdecor-0-dev` and `libdecor-0-plugin-1-gtk`. On earlier Linux versions, or if CMake option `FLTK_USE_SYSTEM_LIBDECOR` is set to OFF, FLTK bundles the most recent source code of `libdecor` and its plugins. The libdecor code bundled inside FLTK is compiled and put in libfltk. FLTK uses `libdecor-gtk` when software package `libgtk-3-dev` is present in the build system, and `libdecor-cairo` otherwise. FLTK prefixes all symbols of its bundled libdecor with "fl_". This allows an FLTK client app to link to other libraries which may use the system version of libdecor.
`Libdecor` uses the Wayland protocol `XDG decoration` to request being decorated by a supporting compositor. If the running compositor supports SSD, `libdecor` doesn't draw window titlebars because the compositor does it. That is what happens with the `KWin` and `Sway` compositors. However, if environment variable `LIBDECOR_FORCE_CSD` is defined to value `1` when an FLTK app runs, `libdecor` instructs an SSD-able compositor to refrain from decorating its windows and decorates windows itself.
Whatever the value of `FLTK_USE_SYSTEM_LIBDECOR`, FLTK and `libdecor` use environment variable `LIBDECOR_PLUGIN_DIR` as follows: if this variable is defined and points to the name of a directory, this directory is searched for a potential `libdecor` plugin in the form of a shared library; if one is found, FLTK and `libdecor` load it and use it.
The `libdecor` source code bundled in FLTK is identical to that of the `libdecor` repository. Nevertheless, FLTK uses this code with some minor changes. For example, when `FLTK_USE_SYSTEM_LIBDECOR` is 0, FLTK needs to modify function `libdecor_new()` charged of loading the plugin, to make it use the plugin code that is included in libfltk if none is found as a dynamic library. This is done as follows in file `libdecor/build/fl_libdecor.c`:

```
#define libdecor_new libdecor_new_orig
```

```
#include "../src/libdecor.c"
#undef libdecor_new

void libdecor_new() { // FLTK rewrite of this function
  ......
}
```

FLTK compiles file `fl_libdecor.c` which includes `libdecor.c` to the effect that all of the `libdecor` code becomes part of libfltk except that function `libdecor_new()` is substituted by its FLTK rewrite, without file `libdecor.c` being modified at all. This trick is also used to modify function `libdecor_frame_set_minimized()` to bypass a bug in the Weston compositor before version 10. Similarly, FLTK compiles file `fl_libdecor-plugins.c` which includes either `libdecor-gtk.c` or `libdecor-cairo.c` to the effect that the desired plugin becomes part of libfltk.

To support function `Fl_Widget_Surface::draw_decorated_window()` that draws a mapped window and its titlebar, FLTK needs to perform two operations: 1) identify what plugin is operating, and 2) call a function that is specific of that plugin and that returns the pixels of the drawn titlebar.

FLTK performs operation 1) above using its function `get_libdecor_plugin_description()` of file `fl_libdecor-plugins.c` that returns a human readable string describing the running plugin. Each plugin puts its own string in member `description` of a record of type `struct libdecor_plugin_description`. Although this type is public in header file `libdecor-plugin.h`, accessing the symbol defined by the plugin to store a pointer to a value of this type is complicated for a reason and solved by a method detailed in a comment before function `get_libdecor_plugin_description()`.

Operation 2) above is done by FLTK-defined function `fl_libdecor_titlebar_buffer()` from file `fl_libdecor-plugins.c`. This function calls `get_libdecor_plugin_description()` seen above to get the running plugin's descriptive string. That is `"GTK3 plugin"` with `libdecor-gtk`. FLTK function `gtk_titlebar_buffer()` is then called, and returns a pointer to the start of a byte buffer containing the titlebar graphics. That is, again, not possible with the public `libdecor` API. Therefore, FLTK copies to `fl_libdecor-plugins.c` the definitions of several types given in `libdecor-gtk.c` or `libdecor-cairo.c` such as type `struct border_component`.

3.2.19 Copy/Paste/Drag-n-Drop

FLTK follows the procedure that is very well described in item "Wayland clipboard and drag & drop" of the Documentation resources. FLTK also implements the 'Primary selection' Wayland protocol to allow middle-click pasting of the current text selection. All corresponding source code is in file `src/drivers/Wayland/fl_wayland_clipboard_dnd.cxx`.

This part of the Fl_Wayland_Screen_Driver::seat record stores pointers to Wayland objects used for clipboard, Drag-n-Drop and middle-click-paste operations:

```
  struct wl_data_device_manager *data_device_manager;
  struct wl_data_device *data_device;
  struct wl_data_source *data_source;
#if HAVE_PRIMARY_SELECTION
  struct zwp_primary_selection_device_manager_v1 *primary_selection_device_manager;
  struct zwp_primary_selection_device_v1 *primary_selection_device;
  struct zwp_primary_selection_source_v1 *primary_selection_source;
#endif
```

FLTK can copy or paste plain UTF-8 text or image data to/from the clipboard. Images are copied to the clipboard as `image/bmp` mime type. Images in `image/bmp` or `image/png` mime types from the clipboard can be pasted to FLTK apps. It can also middle-click paste the current text selection of the current app or any other running app.

Files dropped are received one pathname per line, with no `'\n'` after the last pathname.

3.2.20 EGL as support for OpenGL

Wayland uses EGL™ to interface OpenGL with the underlying native platform window system. OpenGL-using FLTK apps are therefore linked to `libwayland-egl.so` and `libEGL.so` in addition to `libGL.so` and `libGLU.so`.

EGL completely hides the `wl_buffer` objects it uses to draw to GL windows. The wld_buffer structure and the 'buffer factory' described previously are not used for `Fl_Gl_Window's` : the `buffer` member of an `Fl_Gl_Window's` wld_window structure is always NULL.

EGL is initialized calling member function `Fl_Wayland_Gl_Window_Driver::init()` once, the first time the `Fl_Wayland_Gl_Window_Driver` c'tor runs. That is done with calls to `eglGetDisplay()`, `eglInitialize()`, and `eglBindAPI()`.

Member function `Fl_Wayland_Gl_Window_Driver::find()` calls `eglChooseConfig()` to filter the set of GL configurations that match the `Fl_Gl_Window's` mode(), and puts in the returned `Fl_Gl_Choice`

object the first matching configuration. The filtering gets done with bits EGL_WINDOW_BIT, to support the creation of window surfaces, and EGL_OPENGL_BIT, to support the creation of OpenGL contexts.
EGL needs 2 more objects created for each Fl_Gl_Window. They have types struct wl_egl_window and EGLSurface, and are created by member function Fl_Wayland_Gl_Window_Driver::make_current_before() which runs at the beginning of Fl_Gl_Window::make_current(). The first argument of the call to wl_egl_window_create() therein has type struct wl_surface * and is what connects EGL with the targeted Wayland window.
EGL creates with eglCreateContext() an object of type EGLContext via member function Fl_Wayland_Gl_Window_Driver::create_gl_context() called by Fl_Gl_Window::make_current().
Types EGLContext and GLContext are 2 names for the same object. The call to eglCreateContext() is made asking for a GL context of version at least 2. This does not prevent from obtaining contexts of higher versions, namely above 3.2, which are compatible with version 2 (the so-called compatibility profile) under all tested Linux systems.
FLTK function Fl_Gl_Window::make_current() calls overridden function Fl_Wayland_Gl_Window_Driver::set_gl_context() which calls EGL function eglMakeCurrent() when the cached context changes.
FLTK calls function Fl_Wayland_Gl_Window_Driver::swap_buffers() each time it wants a GL context to be sent to the display. This function contains some pure GL code to emulate an overlay buffer to support Fl_Gl_Window objects overriding their draw_overlay() member function. Then, it calls function eglSwapBuffers().
The overridden Fl_Wayland_Gl_Window_Driver::resize() function is implemented with calls to wl_egl_window_get_attached_size() and wl_egl_window_resize().
Class Fl_Wayland_Gl_Plugin exists to allow libfltk to call functions from libfltk_gl, libwayland-egl.so or libEGL.so and without having libfltk force linking any FLTK app with these GL-related libraries. For example, Fl_Wayland_Window_Driver::flush() needs to call Fl_Gl_Window::valid(0).

Throttling GL window redraws

Although no documentation covering this subject was found, the EGL library internally uses wl_callback objects to throttle GL window redraws, and FLTK needs not interfere with these operations. Nevertheless FLTK creates and uses wl_callback objects for GL windows in 2 cases:

- when a decorated GL window is being interactively resized. Function Fl_Wayland_Gl_Window_Driver::resize() creates a wl_callback object, assigns it to xid->frame_cb and calls wl_callback_add_listener() before calling wl_egl_window_resize(). This allows the mechanism described above that prevents surfaces from being resized too frequently to operate with decorated Fl_GL_Window's too.
- when a GL subwindow is being refreshed by Fl_Wayland_Gl_Window_Driver::swap_buffers(). FLTK checks that xid->frame_cb is NULL and if so creates a wl_callback calling wl_surface_frame() before calling eglSwapBuffers(). This is useful if the GL subwindow becomes entirely out from the screen area. In that case, the Mutter compositor stops signaling that the subwindow is ready for new commits which FLTK detects because xid->frame_cb remains non-NULL. If the subwindow eventually re-appears partially on-screen, xid->frame_cb becomes NULL and FLTK calls eglSwapBuffers() to redraw the GL scene.

3.2.21 FLTK-defined, Wayland-specific types

struct wld_window

Defined in Fl_Wayland_Window_Driver.H. One such record is created for each shown()'n Fl_Window by Fl_Wayland_Window_Driver::makeWindow(). Function fl_wl_xid(Fl_Window*) returns a pointer to the struct wld_window of its argument.

```
struct wld_window {
  Fl_Window *fl_win;
  struct wl_list outputs; // linked list of displays where part or whole of window maps
  struct wl_surface *wl_surface; // the window's surface
  struct wl_callback *frame_cb; // non-NULL until Wayland can process new surface commit
  struct Fl_Wayland_Graphics_Driver::wld_buffer *buffer; // see \ref wld_buffer
  struct xdg_surface *xdg_surface;
  enum Fl_Wayland_Window_Driver::kind kind; // DECORATED or POPUP or SUBWINDOW or UNFRAMED
  union {
```

```
    struct libdecor_frame *frame; // for DECORATED windows
    struct wl_subsurface *subsurface; // for SUBWINDOW windows
    struct xdg_popup *xdg_popup; // for POPUP windows
    struct xdg_toplevel *xdg_toplevel; // for UNFRAMED windows
  };
  struct Fl_Wayland_Window_Driver::custom_cursor {
    struct wl_cursor *wl_cursor;
    const Fl_RGB_Image *rgb;
    int hotx, hoty;
  } *custom_cursor; // non-null when using custom cursor
  int configured_width; // used when negotiating window size with the compositor
  int configured_height;
  int floating_width; // helps restoring size after un-maximizing
  int floating_height;
  int state; // indicates whether window is fullscreen, maximized. Used otherwise for POPUPs
  bool covered; // specially for Mutter and issue #878
}
```

struct Fl_Wayland_Graphics_Driver::draw_buffer

Defined in file `Fl_Wayland_Graphics_Driver.H`. One such record is created when an Fl_Image_Surface object is created. One such record is also embedded inside each `struct Fl_Wayland_Graphics_Driver::wld_buffer` record (see wld_buffer).

```
struct Fl_Wayland_Graphics_Driver::draw_buffer {
  unsigned char *buffer; // address of the beginning of the Cairo image surface's byte array
  cairo_t *cairo_; // used when drawing to the Cairo image surface
  size_t data_size; // of buffer and wl_buffer, in bytes
  int stride; // bytes per line
  int width; // in pixels
};
```

FLTK gives offscreen buffers the platform-dependent type `Fl_Offscreen` which is in fact member `cairo_` of `struct Fl_Wayland_Graphics_Driver::draw_buffer`. Thus, a variable with type `Fl_Offscreen` needs be cast to type `cairo_t*`. Static member function `struct draw_buffer *offscreen_buffer(Fl_Offscreen)` of class `Fl_Wayland_Graphics_Driver` returns the `draw_buffer` record corresponding to an `Fl_Offscreen` value.

struct Fl_Wayland_Graphics_Driver::wld_buffer

Defined in file `Fl_Wayland_Graphics_Driver.H`. One such record is created by `Fl_Wayland_Graphics_Driver::create_wld_buffer()` when an Fl_Window is show()'n or resized, when a custom cursor shape is created, or when text is dragged.

```
struct Fl_Wayland_Graphics_Driver::wld_buffer {
  struct draw_buffer draw_buffer; // see draw_buffer
  struct wl_list link; // links all buffers from the same wl_shm_pool
  struct wl_buffer *wl_buffer; // the Wayland buffer
  void *data; // address of the beginning of the Wayland buffer's byte array
  struct wl_shm_pool *shm_pool; // pter to wl_shm_pool from which this wl_buffer comes
  bool draw_buffer_needs_commit; // true when draw_buffer has been modified but not yet committed
  bool in_use; // true while being committed
  bool released; // true after buffer_release() was called
};
```

struct Fl_Wayland_Screen_Driver::output

Defined in Fl_Wayland_Screen_Driver.H. One such record is created for each display of the system by function `registry_handle_global()` when it receives a `"wl_output"` interface. These records are kept in a linked list of them all, and an identifier of this linked list is stored in member `outputs` of the unique `Fl_Wayland_Screen_Driver` object FLTK uses. Thus,

```
Fl_Wayland_Screen_Driver *scr_driver = (Fl_Wayland_Screen_Driver*)Fl::screen_driver();
struct wl_list list_of_all_displays = scr_driver->outputs;
```

gives access, the Wayland way, to the linked list of displays in the system.

```
struct Fl_Wayland_Screen_Driver::output { // one record for each display
  uint32_t id; // an identifier of the display
  int x, y; // logical position of the top-left of display
  int pixel_width; // in pixels
  int pixel_height; // in pixels
  int width; // in pixels, account for fractional scaling
  int height; // in pixels, account for fractional scaling
  float dpi; // at this point, always 96.
  struct wl_output *wl_output; // the Wayland object for this display
  int wld_scale; // Wayland scale factor
  float gui_scale; // FLTK scale factor
  bool done; // true means record members have been initialized
  struct wl_list link; // links these records together
};
```

It's possible to get the FLTK-defined record associated to a display from the Wayland-associated object for the same display, say `struct wl_output *wl_output`, by this call: `(struct Fl_Wayland_Screen_Driver::output *)wl_output_get_user_data(wl_output)`.

struct Fl_Wayland_Screen_Driver::seat

Defined in file `Fl_Wayland_Screen_Driver.H`. One record is created by function `registry_handle_global()` when it receives a `"wl_seat"` or `wl_data_device_manager_interface.name` interface. A pointer to this struct is stored in member `seat` of the client's unique `Fl_Wayland_Screen_Driver` object.

```
struct Fl_Wayland_Screen_Driver::seat {
  struct wl_seat *wl_seat;
  struct wl_pointer *wl_pointer;
  struct wl_keyboard *wl_keyboard;
  uint32_t keyboard_enter_serial;
  struct wl_surface *keyboard_surface;
  struct wl_list pointer_outputs;
  struct wl_cursor_theme *cursor_theme;
  struct wl_cursor *default_cursor;
  struct wl_surface *cursor_surface;
  struct wl_surface *pointer_focus;
  int pointer_scale;
  uint32_t serial;
  uint32_t pointer_enter_serial;
  struct wl_data_device_manager *data_device_manager;
  struct wl_data_device *data_device;
  struct wl_data_source *data_source;
  struct xkb_state *xkb_state;
  struct xkb_context *xkb_context;
  struct xkb_keymap *xkb_keymap;
  struct xkb_compose_state *xkb_compose_state;
  char *name;
  struct zwp_text_input_v3 *text_input;
  struct gtk_shell1 *gtk_shell;
};
```

3.2.22 Documentation resources

The Wayland book	Extensive introduction to Wayland programming written by the author of the *sway* compositor, unfortunately unachieved.	
Wayland Explorer	Documentation of all Wayland protocols (stable, staging and unstable).	
Wayland Protocol Specification	Documentation for all functions of the Wayland core protocol.	
Wayland clipboard and drag & drop	Detailed explanation of how clipboard and drag-and-drop work under Wayland.	
Wayland and input methods	Blog article introducing to the issue of text input methods under Wayland.	
Input Method Hub	Entry page for input method support giving newcomers a first understanding of what input methods are and how they are implemented in Wayland.	

3.3 Developer info for bundled libs

This chapter details the procedure to update the libraries which are bundled inside FLTK.

3.3.1 Introduction

This file is mainly intended for FLTK developers and contains information about the current versions of all bundled libraries and about how to upgrade these bundled libraries.
Starting with FLTK 1.4.0 the bundled libraries jpeg, png, and zlib use "symbol prefixing" with the prefix 'fltk_' for all external symbols to distinguish the bundled libraries from existing system libraries and to avoid runtime errors.
User code compiled correctly with the header files provided by the bundled image libraries need not be changed.
The nanosvg library is not affected.

3.3.2 Current status

```
Current versions of bundled libraries (as of Jul 8, 2026):
  Library       Version/git commit Release date         FLTK Version
  ---------------------------------------------------------------------------
  jpeg          jpeg-10            2026-01-26           1.5.0
  nanosvg       7aeda550a8 [1]     2023-12-02           1.4.0
  png           libpng-1.6.58      2026-04-15           1.5.0
  zlib          zlib-1.3.2         2026-02-17           1.5.0
  libdecor      59c498af   [2]     2026-03-22           1.5.0
  ---------------------------------------------------------------------------

Previous versions of bundled libraries (FLTK 1.3.x):

  Library       Version            Release date         FLTK Version
  ---------------------------------------------------------------------------
  jpeg          jpeg-9e            2022-01-16           1.3.9
  png           libpng-1.6.40      2023-06-21           1.3.9
  zlib          zlib-1.3           2023-08-18           1.3.9

  jpeg          jpeg-9d            2020-01-12           1.3.6 - 1.3.8
  png           libpng-1.6.37      2019-04-14           1.3.6 - 1.3.8
  zlib          zlib-1.2.11        2017-01-15           1.3.6 - 1.3.8
  ---------------------------------------------------------------------------


[1] Git commit in branch 'fltk' of https://github.com/fltk/nanosvg
    See also git tag 'fltk_yyyy-mm-dd' where yyyy-mm-dd == "Release date"
    and file nanosvg/README.txt.
[2] Git commit in https://gitlab.freedesktop.org/libdecor/libdecor
```

General information:

FLTK does not include the entire library distributions. We only provide the source files necessary to build the FLTK library and some README and/or CHANGELOG files. There are no test programs or other contributed files.
We use our own build files, hence a few files MUST NOT be upgraded when the library source files are upgraded.
We strive to keep changes to the library source files as small as possible. Patching library code to work with FLTK should be a rare exception. Symbol prefixing with prefix 'fltk_' is one such exception to the rule.
If patches are necessary all changes in the library files should be marked with "FLTK" in a comment so a developer who upgrades the library later is aware of changes in the source code for FLTK. Look for 'FLTK' and/or 'fltk_' to find the differences.
Additional comments should be added to show the rationale, i.e. why a particular change was necessary. If applicable, add a reference to a Software Trouble Report, GitHub Issue or Pull Request (PR) like "STR 3456", "Issue #123", or "PR #234".

3.3.3 How to update the bundled libraries

It is generally advisable to use a graphical merge program. I'm using 'meld' under Linux, but YMMV.
Do not add any source files unless they are required to build the library.
Some config header files may be pre-generated in the FLTK sources. These header files should be left untouched, but it may be necessary to update these files if new items were added to the new library version. In this case the new header should be pre-generated on a Linux system with default options unless otherwise mentioned below for a specific library. Currently there are no known exceptions.

Merging source files:

Please check if some source and header files contain "FLTK" comments and/or 'fltk_' symbol prefixing to be aware of necessary merges. It is also good to download the distribution tar ball or Git source files of the previous version and to run a (graphical) diff or merge tool on the previous version and the bundled version of FLTK to see the "previous" differences.
Files that were not patched in previous versions should be copied to the new version w/o changes. Files that had FLTK specific patches must be merged manually. FLTK patches should be verified (if still necessary) and should be kept in the new source files.
Source and header files that have been added in the new library version should be added in FLTK as well if they are necessary to build the library. A simple "trial and error" should be sufficient to find files that need to be added. Added files must be added to FLTK's build files as well, usually 'CMakeLists.txt', to be used in CMake based builds.

Upgrade order:

There is only one dependency between all bundled libraries: libpng depends on zlib. Hence zlib should be upgraded first, then all other libs can be upgraded in arbitrary order.

Tests after merge:

Tests should be done on as many platforms as possible. Windows (Visual Studio) and macOS (Xcode) builds need CMake to generate the IDE files.

Upgrade notes for specific libraries:

The following chapters contain informations about specific files and how they are upgraded. Since the changes in all bundled libraries are not known in advance this information may change in the future. Please verify that no other changes are necessary.

3.3.4 zlib:

Website: https://zlib.net/
Download: See website and follow links.
Repository: git clone https://github.com/madler/zlib.git
zlib should be upgraded first because libpng depends on zlib.
Download the latest zlib sources, 'cd' to /path-to/zlib and run

```
$ ./configure --zprefix
```

This creates the header file 'zconf.h' with definitions to enable the standard 'z_' symbol prefix.
Unfortunately zlib requires patching some source and header files to convert this 'z_' prefix to 'fltk_z_' to be more specific. As of this writing (Nov. 2021) three files need symbol prefix patches:

- gzread.c
- zconf.h
- zlib.h

You may want to compare these files and/or the previous version to find out which changes are required. The general rule is to change all occurrences of 'z_' to 'fltk_z_' but there *are* exceptions.
The following files need special handling:

- CMakeLists.txt: Keep FLTK version, update manually if necessary.
- gzread.c: Merge changes (see above, manual merge recommended).
- zconf.h: Merge changes (see above, manual merge recommended).
- zlib.h: Merge changes (see above, manual merge recommended).

3.3.5 png:

Website: http://libpng.org/pub/png/libpng.html
Download: See website and follow links.
Repository: git clone https://git.code.sf.net/p/libpng/code libpng
libpng should be upgraded after zlib because it depends on zlib.
Download the latest libpng sources, 'cd' to /path-to/libpng and run

```
$ ./configure --with-libpng-prefix=fltk_
$ make
```

This creates the header files 'pnglibconf.h' and 'pngprefix.h' with the 'fltk_' symbol prefix.
The following files need special handling:

- CMakeLists.txt: Keep FLTK version, update manually if necessary.
- pnglibconf.h: Generate on a Linux system and merge (see above).
- pngprefix.h: Generate on a Linux system and merge (see above).

3.3.6 jpeg:

Website: https://ijg.org/
Download: See website and follow links.
Repository: N/A
Download the latest jpeg-xy sources on a Linux (or Unix) system, 'cd' to /path-to/jpeg-xy and run

```
$ ./configure
$ make [-jN]
```

This builds the library and should create the static library file '.libs/libjpeg.a'.
Execute the following command to extract the libjpeg symbol names used to build the 'prefixed' libfltk_jpeg library:

```
$ nm --extern-only --defined-only .libs/libjpeg.a | awk '{print $3}' \
  | sed '/^$/d' | sort -u | awk '{print "#define "$1" fltk_"$1}' \
  > fltk_jpeg_prefix.h
```

This creates the header file 'fltk_jpeg_prefix.h' with the '# define' statements using the 'fltk_' symbol prefix.
The following files need special handling:

- CMakeLists.txt: Keep FLTK version, update manually if necessary.
- fltk_jpeg_prefix.h: Generate on a Linux system and merge (see above).
- jconfig.h: keep changes flagged with

  ```
  /* FLTK */
  ```

  Note: more to come...

3.3.7 nanosvg:

Website: https://github.com/memononen/nanosvg
Download: See website and follow links.
Repository: git clone https://github.com/memononen/nanosvg.git
FLTK Fork: git clone https://github.com/fltk/nanosvg.git
FLTK has its own GitHub fork of the original repository (see above).
The intention is to update this fork from time to time so the FLTK specific patches are up-to-date with the original library. Hopefully the FLTK patches will be accepted upstream at some time in the future so we no longer need our own patches. AlbrechtS, 04 Feb 2018.
Update (Feb 22, 2021): The upstream library is officially no longer maintained (see README.md) although updates appear from time to time.
Use this fork (branch 'fltk') to get the nanosvg library with FLTK specific patches:

```
$ git clone https://github.com/fltk/nanosvg.git nanosvg-fltk
$ cd nanosvg-fltk
$ git checkout fltk
$ cd src
$ cp nanosvg.h nanosvgrast.h /path/to/fltk-1.4/nanosvg/
```

This library does not have its own build files since it is a header-only library. The headers are included in FLTK where necessary.
The following files need special handling:
nanosvg.h: Merge or download from FLTK's fork (see above).
nanosvgrast.h: Merge or download from FLTK's fork (see above).
Maintaining branch 'fltk' in FLTK's fork of nanosvg (fltk/nanosvg):
Only maintainers with write access on fltk/nanosvg can do this. Others can fork our fltk/nanosvg fork in their own GitHub account and either open a PR on fltk/nanosvg or tell us about their changes in fltk.development.
Use something similar to the following commands to update FLTK's fork of nanosvg to the latest version. Commands are only examples, you may need to change more or less, depending on the outstanding updates.
Step 1: clone the fltk/nanosvg fork, set the remote 'upstream', and update the 'master' branch:

```
$ cd /to/your/dev/dir
$ git clone https://github.com/fltk/nanosvg.git nanosvg-fltk
$ cd nanosvg-fltk
$ git remote add upstream https://github.com/memononen/nanosvg
$ git checkout master
$ git pull upstream master
```

Note: the 'master' branch must never be changed, i.e. it must always be the same as 'upstream/master'. Never commit your own (FLTK specific) changes to branch 'master'.
Step 2: rebase branch 'fltk' on the new master (upstream/master), fix potential conflicts, and tag the new branch.
It is important to keep the individual FLTK specific patches intact (one commit per patch) because this will preserve the history and the committer and make it easier to skip single patches when they are accepted upstream.

```
$ git checkout fltk
$ git rebase upstream/master
```

At this point you may need to fix conflicts! Do whatever is necessary to update the branch 'fltk'.
Now 'git tag' the 'fltk' branch for later reference.
Hint: use 'git show <any-older-tag-name>' to see its contents. I like to write a summary of commits in the tag comment.

```
$ git tag -a fltk_yyyy-mm-dd fltk
```

Replace 'yyyy-mm-dd' with the current date and add a comment when asked for it (your editor will open an empty file).
Step 3: at this point it is recommended to copy the changed header files to your working copy of the FLTK library and test the changes. If anything is wrong, go back, fix the bugs and change the git tag (delete and create a new one).
Step 4: push the new branch 'fltk' and the tag to the fltk/nanosvg repository:

```
$ git push -f origin fltk
$ git push origin fltk_yyyy-mm-dd
```

Step 5: copy the changed files to your working copy of the FLTK repository (if not done already), update this file accordingly, and commit/push the update to the fltk/fltk repository.

3.3.8 libdecor:

Website: https://gitlab.freedesktop.org/libdecor/libdecor
Download: See website and follow links.
Repository: git clone https://gitlab.freedesktop.org/libdecor/libdecor.git
The Wayland/X11 hybrid platform of FLTK uses a software component called *libdecor* when FLTK apps run as Wayland clients. It allows FLTK to detect whether the Wayland compositor in use requires client-side window decoration and to decorate windows if it does.
In recent Linux distributions, FLTK is linked with the *libdecor-0.so* system library installed from software package *libdecor-0-dev*. FLTK bundles a copy of the libdecor source code to support Linux/Unix configurations where the *libdecor-0-dev* package is not available, not recent enough, or not installed.
FLTK uses libdecor source files without any modification. This part of the libdecor source tree is copied to directory libdecor/ of the FLTK source tree:

```
LICENSE
README.md
src/    ... and files below except meson.build files
```

Furthermore, directory libdecor/build/ of the FLTK source tree does not originate from the libdecor source tree but contains 4 FLTK-created files. File src/CMakeLists.txt may need changes if a libdecor update adds or renames source files.

3.4 Developer Information

This chapter describes FLTK development and documentation.

Example

```
/** \file
   Fl_Clock, Fl_Clock_Output widgets. */


/**
  \class Fl_Clock_Output
  \brief This widget can be used to display a program-supplied time.

  The time shown on the clock is not updated. To display the current time,
  use Fl_Clock instead.

  \image html clock.png
  \image latex clock.png "" width=10cm
  \image html round_clock.png
  \image latex clock.png "" width=10cm
  \image html round_clock.png "" width=10cm */
```

```
  /**
    Returns the displayed time.
    Returns the time in seconds since the UNIX epoch (January 1, 1970).
    \see value(ulong)
   */
  ulong value() const {return value_;}

/**
  Set the displayed time.
  Set the time in seconds since the UNIX epoch (January 1, 1970).
  \param[in] v seconds since epoch
  \see value()
 */
void Fl_Clock_Output::value(ulong v) {
 [...]
}

/**
  Create an Fl_Clock widget using the given position, size, and label string.
  The default boxtype is \c FL_NO_BOX.
  \param[in] X, Y, W, H position and size of the widget
  \param[in] L widget label, default is no label
 */
Fl_Clock::Fl_Clock(int X, int Y, int W, int H, const char *L)
  : Fl_Clock_Output(X, Y, W, H, L) {}

/**
  Create an Fl_Clock widget using the given boxtype, position, size, and
  label string.
  \param[in] t boxtype
  \param[in] X, Y, W, H position and size of the widget
  \param[in] L widget label, default is no label
 */
Fl_Clock::Fl_Clock(uchar t, int X, int Y, int W, int H, const char *L)
  : Fl_Clock_Output(X, Y, W, H, L) {
  type(t);
  box(t==FL_ROUND_CLOCK ? FL_NO_BOX : FL_UP_BOX);
}
```

Note

From Duncan: (will be removed later, just for now as a reminder)
I've just added comments for the fl_color_chooser() functions, and in order to keep them and the general Function Reference information for them together, I created a new doxygen group, and used \ingroup in the three comment blocks. This creates a new Modules page (which may not be what we want) with links to it from the File Members and Fl_Color_Chooser.H pages. It needs a bit more experimentation on my part unless someone already knows how this should be handled. (Maybe we can add it to a functions.dox file that defines a functions group and do that for all of the function documentation?)
Update: the trick is not to create duplicate entries in a new group, but to move the function information into the doxygen comments for the class, and use the navigation links provided. Simply using \relatesalso as the first doxygen command in the function's comment puts it in the appropriate place. There is no need to have \defgroup and \ingroup as well, and indeed they don't work. So, to summarize:

```
Gizmo.H
  /** \class Gizmo
      A gizmo that does everything
    */
  class Gizmo {
    etc
  };
  extern int popup_gizmo(...);

Gizmo.cxx:
  /** \relatesalso Gizmo
      Pops up a gizmo dialog with a Gizmo in it
    */
  int popup_gizmo(...);
```

Comments Within Doxygen Comment Blocks

You can use HTML comment statements to embed comments in doxygen comment blocks. These comments will not be visible in the generated document.

```
The following text is a developer comment.
<!-- *** This *** is *** invisible *** -->
This will be visible again.
```

will be shown as:
The following text is a developer comment.
This will be visible again.

Note

> Since an unknown Doxygen version indentation by four or more bytes is automatically displayed as if it was framed by \code and \endcode. You need to take care that "normal" text is not indented that much to avoid mis-interpretation and formatting as code.

Different Headlines

You can use HTML tags <H1> ... <H4> for headlines with different sizes. As of doxygen 1.8.x there must not be more than three spaces at the beginning of the line for this to work. Currently (doxygen 1.8.6) there seems to be no difference in the font sizes of <H3> and <H4> in the pdf output, whereas the html output uses different font sizes.

```
<H1>Headline in big text (H1)</H1>
<H2>Headline in big text (H2)</H2>
<H3>Headline in big text (H3)</H3>
<H4>Headline in big text (H4)</H4>
```

Headline in big text (H1)

Headline in big text (H2)

Headline in big text (H3)

Headline in big text (H4)

3.4.1 Non-ASCII Characters

```
Doxygen understands many HTML quoting characters like
&quot;, &uuml;, &ccedil;, &Ccedil;, but not all HTML quoting characters.
```

This will appear in the document:
Doxygen understands many HTML quoting characters like ", ü, ç, Ç, but not all HTML quoting characters.
For further informations about HTML quoting characters see
http://www.doxygen.org/manual/htmlcmds.html
Alternatively you can use **UTF-8** encoding within Doxygen comments.

3.4.2 Document Structure

- **page** creates a named page
- **section** creates a named section within that page
- **subsection** creates a named subsection within the current section
- **subsubsection** creates a named subsubsection within the current subsection

All these statements take a "name" as their first argument, and a title as their second argument. The title can contain spaces.
The page, section, and subsection titles are formatted in blue color and a size like **"<H1>"**, **"<H2>"**, and **"<H3>"**, and **"<H4>"**, respectively.
By **FLTK documentation convention**, a file like this one with a doxygen documentation chapter has the name **"<chapter>.dox".** The **page** statement at the top of the page is **"\page <chapter> This is the title"**. Sections within a documentation page must be called **"<chapter>_<section>"**, where **"<chapter>"** is the name part of the file, and **"<section>"** is a unique section name within the page that can be referenced in links. The same for subsections and subsubsections.

These doxygen page and section commands work only in special documentation chapters, not within normal source or header documentation blocks. However, links **from** normal (e.g. class) documentation **to** documentation sections **do work**.
This page has
`\page development Developer Information`
at its top.
This section is
`\section development_structure Document Structure`
The following section is
`\section development_links Creating Links`

3.4.3 Creating Links

Links to other documents and external links can be embedded with

- doxygen \ref links to other doxygen \page, \section, \subsection and \anchor locations
- HTML links without markup - doxygen creates "http://..." links automatically
- standard, non-Doxygen, HTML links

```
-   see chapter \ref unicode creates a link to the named chapter
    unicode that has been created with a \\page statement.

-   For further informations about quoting see
    http://www.doxygen.org/manual/htmlcmds.html

-   see <a href="https://www.fltk.org/">FLTK Library</a> creates
    a standard HTML link
```

appears as:

- see chapter Unicode and UTF-8 Support creates a link to the named chapter unicode that has been created with a \page statement.
- For further informations about quoting see http://www.doxygen.org/manual/htmlcmds.html
- see FLTK Library creates a standard HTML link

3.4.4 Paragraph Layout

There is no real need to use HTML <P> and </P> tags within the text to tell doxygen to start or stop a paragraph. In most cases, when doxygen encounters a blank line or some, but not all, **commands** in the text it knows that it has reached the start or end of a paragraph. Doxygen also offers the **par** command for special paragraph handling. It can be used to provide a paragraph title and also to indent a paragraph. Unfortunately **par** won't do what you expect if you want to have doxygen links and sometimes html tags don't work either.

```
\par Normal Paragraph with title

This paragraph will have a title, but because there is a blank line
between the \par and the text, it will have the normal layout.

\par Indented Paragraph with title
This paragraph will also have a title, but because there is no blank
line between the \par and the text, it will be indented.

\par
It is also possible to have an indented paragraph without title.
This is how you indent subsequent paragraphs.

\par No link to Fl_Widget::draw()
Note that the paragraph title is treated as plain text.
Doxygen type links will not work.
HTML characters and tags may or may not work.

Fl_Widget::draw() links and &quot;html&quot; tags work<br>
\par
Use a single line ending with <br> for complicated paragraph titles.
```

The above code produces the following paragraphs:

Normal Paragraph with title

This paragraph will have a title, but because there is a blank line between the \par and the text, it will have the normal layout.

Indented Paragraph with title

This paragraph will also have a title, but because there is no blank line between the \par and the text, it will be indented.

It is also possible to have an indented paragraph without title. This is how you indent subsequent paragraphs.

No link to Fl_Widget::draw()

Note that the paragraph title is treated as plain text. Doxygen type links will not work. HTML characters and tags may or may not work.

Fl_Widget::draw() links and "html" tags work

Use a single line ending with <br> for complicated paragraph titles.

3.4.5 Navigation Elements

Each introduction (tutorial) page ends with navigation elements. These elements must only be included in the html documentation, therefore they must be separated with \htmlonly and \endhtmlonly.
The following code creates the navigation bar at the bottom of the FLTK Basics HTML page as an example of all pages in the "FLTK Programming Manual" section. Note that **this page** is one of the appendices w/o navigation bar.

```
\htmlonly
<hr>
<table summary="navigation bar" width="100%" border="0">
<tr>
  <td width="45%" align="LEFT">
    <a class="el" href="intro.html">
    [Prev]
    Introduction to FLTK
    </a>
  </td>
  <td width="10%" align="CENTER">
    <a class="el" href="index.html">[Index]</a>
  </td>
  <td width="45%" align="RIGHT">
    <a class="el" href="common.html">
    Common Widgets and Attributes
    [Next]
    </a>
  </td>
</tr>
</table>
\endhtmlonly
```

Chapter 4

Todo List

Page Adding and Extending Widgets

Clarify Fl_Window::damage(uchar) handling - seems confused/wrong? ORing value doesn't match setting behavior in Fl_Widget.H!

Member Fl::now (double offset=0)

Fl::system_driver()->gettime() was implemented for the Forms library and has a limited resolution (on Windows: milliseconds). On POSIX platforms it uses gettimeofday() with microsecond resolution. A new function could use a better resolution on Windows with its multimedia timers which requires a new dependency: winmm.lib (dll). This could be a future improvement, maybe set as a build option or generally (requires Win95 or 98?).

Class Fl_Chart

Refactor Fl_Chart::type() information.

Member fl_filename_list (const char *d, struct dirent ***l, Fl_File_Sort_F *s=fl_numericsort)

should support returning OS error messages

Class Fl_Grid

This (relative group coordinates of nested groups of Fl_Grid) needs explanation and maybe an example.

Member Fl_Grid::Cell::~Cell ()=default

Fl_Grid's cell destructor should remove the cell from the grid. Currently it does nothing!

Member Fl_Grid::clear_layout ()

Fl_Grid::clear() needs to be implemented as documented above!

Member Fl_Grid::debug (int level=127)

Add more information about cells and children.

Control output by using `level`.

Member Fl_Grid::Fl_Grid (int X, int Y, int W, int H, const char *L=0)

More documentation of Fl_Grid constructor?

Member Fl_Grid::layout ()

Document when and why to call layout() w/o args. See Fl_Flex::layout()

Member Fl_Grid::layout (int rows, int cols, int margin=-1, int gap=-1)

Document when and why to call layout() w/o args. See Fl_Flex::layout()

Member Fl_Group::delete_child (int n)

Reimplementation of Fl_Group::delete_child(int) in more FLTK subclasses. This is not yet complete.

Member fl_height (int font, int size)

In the future, when the XFT issues are resolved, this function should simply return the 'size' value.

Member Fl_Input_::handle_mouse (int, int, int, int, int keepmark=0)

Add comment and parameters

Member Fl_Input_::handletext (int e, int, int, int, int)

Add comment and parameters

Struct Fl_Label

There is an aspiration that the Fl_Label type will become a widget by itself. That way we will be avoiding a lot of code duplication by handling labels in a similar fashion to widgets containing text. We also provide an easy interface for very complex labels, containing html or vector graphics. However, this re-factoring is not in place in this release.

Member Fl_Menu_::add (const char *, int shortcut, Fl_Callback *, void *=0, int=0)

Raw integer shortcut needs examples. Dependent on responses to https://www.fltk.org/newsgroups.php?gfltk.coredev+v:10086 and results of STR#2344

Member Fl_Shortcut

Discuss and decide whether we can "shift" these special keyboard flags to the upper byte to enable full 21-bit Unicode characters (U+0000 .. U+10FFFF) plus the keyboard indicator bits as this was originally intended. This would be possible if we could rely on **all** programs being coded with symbolic names and not hard coded bit values.

Member Fl_Terminal::scrollbar

Support scrollbar_left/right() - See Fl_Browser_::scrollbar docs

Support new ScrollbarStyle

Member Fl_Text_Display::extend_range_for_styles (int *start, int *end)

Unicode?

Member Fl_Text_Display::handle_vline (int mode, int lineStart, int lineLen, int leftChar, int rightChar, int topClip, int bottomClip, int leftClip, int rightClip) const

we need to handle hidden hyphens and tabs here!

we handle all styles and selections

we must provide code to get pixel positions of the middle of a character as well

Member Fl_Text_Display::overstrike (const char *text)

Unicode? Find out exactly what we do here and simplify.

Member Fl_Text_Display::position_to_linecol (int pos, int *lineNum, int *column) const

a column number makes little sense in the UTF-8/variable font width environment. We will have to further define what exactly we want to return. Please check the functions that call this particular function.

Member Fl_Text_Display::scroll (int topLineNum, int horizOffset)

Column numbers make little sense here.

Member Fl_Text_Display::scrollbar_width (int width)

This method should eventually be removed

Member Fl_Text_Display::scrollbar_width () const

This method should eventually be removed.

Member Fl_Text_Display::shortcut (int s)

FIXME : get set methods pointing on shortcut_ have no effects as shortcut_ is unused in this class and derived!

Member Fl_Text_Display::shortcut () const

FIXME : get set methods pointing on shortcut_ have no effects as shortcut_ is unused in this class and derived!

Member Fl_Text_Display::wrapped_column (int row, int column) const

Unicode?

What does this do and how is it useful? Column numbers mean little in this context. Which functions depend on this one? Function TextDXYToUnconstrainedPosition does not exist (nedit port?)

Member Fl_Text_Display::wrapped_row (int row) const

What does this do and how is it useful? Column numbers mean little in this context. Which functions depend on this one? Function TextDXYToUnconstrainedPosition does not exist (nedit port?)

Member Fl_Tiled_Image::Fl_Tiled_Image (Fl_Image *i, int W=0, int H=0)

Fix Fl_Tiled_Image as background image for widgets and windows and fix the implementation of Fl::scheme(const char *).

Member Fl_Tree::handle (int e) override

add Fl_Widget_Tracker (see Fl_Browser_.cxx::handle())

Member Fl_Tree::is_scrollbar (Fl_Widget *w)

should be const

Member Fl_Tree::show_self ()

should be const

Member Fl_Window::show () override

Check if we can remove resetting the current group in a later FLTK version (after 1.3.x). This may break "already broken" programs though if they rely on this "feature".

Page FLTK Basics

This section needs a major rework. Add a chapter "Building FLTK with CMake".

Page FLTK Code Management Plan (CMP)

This needs to be populated with the new restrictions/allowed c++ features.

Remove this paragraph in 1.5,

remove this paragraph in FLTK 1.5, remove this 'todo' in 1.4.

rework paragraph above, explain "sub-systems" and directory names

To be done.

Find current Apple / macOS docs for macOS version macros.

The following needs modification regarding "fltk" namespace (this being an FLTK2 thing IIRC)

The following needs modification; AFAIK we don't use the `fltk` namespace - I think that was an fltk2 thing - but we do recently make some use of `Fl` namespace (matt's recent tablet commits)

Mention of the "fltk" namespace below is I think fltk2 specific, needs changes for 1.x.x

Page Handling Events

Add details on how to detect repeating keys, since on some X servers a repeating key will generate both FL_↩ KEYUP and FL_KEYDOWN, such that to tell if a key is held, you need Fl::event_key(int) to detect if the key is being held down during FL_KEYUP or not.

Page Migrating Code from FLTK 1.4 to 1.5

This table of deprecated functions since FLTK 1.5 may be incomplete and should be completed before the release of FLTK 1.5.0.

Chapter 5

Deprecated List

Member Fl::awake (void ∗message)

"since 1.5.0 - use Fl::awake() or Fl::awake(handler, user_data) instead"

Use Fl::awake() or Fl::awake(Fl_Awake_Handler, void∗) instead.

Member Fl::thread_message ()

"since 1.5.0 - use Fl::awake() or Fl::awake(handler, user_data) instead"

Use Fl::awake(Fl_Awake_Handler, void∗) or Fl::awake_once(Fl_Awake_Handler, void∗) instead.

Member Fl::version ()

Use int Fl::api_version() instead.

Member Fl_Browser_::position () const

"since 1.4.0 - use vposition() instead"

Member Fl_Browser_::position (int pos)

"since 1.4.0 - use vposition(pos) instead"

Member fl_find (Window xid)

Kept in the X11, Windows, and macOS platforms for compatibility with FLTK versions before 1.4. Please use fl_x11_find(Window), fl_wl_find(struct wld_window∗), fl_win32_find(HWND) or fl_mac_find(FLWindow∗) with FLTK 1.4.0 and above.

Member Fl_GIF_Image::Fl_GIF_Image (const char ∗imagename, const unsigned char ∗data)

Please use Fl_GIF_Image(const char ∗imagename, const unsigned char ∗data, const size_t length) instead.

Member Fl_Group::sizes ()

Deprecated since 1.4.0. Please use bounds() instead.

Member Fl_Image::draw_scaled (int X, int Y, int W, int H)

Only for API compatibility with FLTK 1.3.4.

Member Fl_Image::label (Fl_Widget ∗w)

Please use Fl_Widget::image() or Fl_Widget::deimage() instead.

Member Fl_Image::label (Fl_Menu_Item ∗m)

Please use Fl_Menu_Item::image() instead.

Member Fl_Image_Surface::highres_image ()

Use image() instead.

Member Fl_Input_::position (int p)

"since 1.4.0 - use insert_position(p) instead"

Member Fl_Input_::position (int p, int m)

"since 1.4.0 - use insert_position(p, m) or Fl_Widget::position(x, y) instead"

Member Fl_Input_::position () const

"since 1.4.0 - use insert_position() instead"

Member Fl_Menu_Item::setonly (Fl_Menu_Item const ∗first)

Please use Fl_Menu_Item::setonly(Fl_Menu_Item const∗ first) instead.

Member Fl_Multi_Label::label (Fl_Menu_Item ∗)

since 1.4.0: please use Fl_Menu_Item::label(Fl_Multi_Label ∗)

Member Fl_Preferences::Fl_Preferences (const char ∗path, const char ∗vendor, const char ∗application)

"since 1.4.0 - use Fl_Preferences(path, vendor, application, flags) instead"

Member Fl_Text_Display::scrollbar_width (int width)

"since 1.4.0 - use scrollbar_size() instead"

Use scrollbar_size() instead.

Member Fl_Text_Display::scrollbar_width () const

"since 1.4.0 - use scrollbar_size() instead"

Use scrollbar_size() instead.

Member Fl_Text_Selection::position (int ∗startpos, int ∗endpos) const

"since 1.4.0 - use selected(startpos, endpos) instead"

Member Fl_Tile::position (int oldx, int oldy, int newx, int newy)

"since 1.4.0 - use move_intersection(p) instead"

Member Fl_Tree::first_visible ()

in 1.3.3 ABI – use first_visible_item() instead.

Member Fl_Tree::item_clicked (Fl_Tree_Item ∗val)

in 1.3.3 ABI – use callback_item() instead.

Member Fl_Tree::item_clicked ()

in 1.3.3 ABI – use callback_item() instead.

Member Fl_Tree::last_visible ()

in 1.3.3 – use last_visible_item() instead.

Member Fl_Tree_Item::Fl_Tree_Item (const Fl_Tree_Prefs &prefs)

in 1.3.3 ABI – you must use Fl_Tree_Item(Fl_Tree∗) for proper horizontal scrollbar behavior.

Member Fl_Tree_Item::next_displayed (Fl_Tree_Prefs &prefs)

in 1.3.3 for confusing name, use next_visible() instead

Member Fl_Tree_Item::prev_displayed (Fl_Tree_Prefs &prefs)

in 1.3.3 for confusing name, use prev_visible()

Member FL_VERSION

This `double` version number is retained for compatibility with existing program code. New code should use *int* FL_API_VERSION instead. FL_VERSION is deprecated because comparisons of floating point values may fail due to rounding errors. However, there are currently no plans to remove this deprecated constant.

Member Fl_Window::icon () const

in 1.3.3

Member Fl_Window::icon (const void *ic)

in 1.3.3 in favor of platform-independent methods Fl_Window::icon(const Fl_RGB_Image *icon) and Fl_Window::icons(const Fl_RGB_Image *icons[ ], int count).

Page Operating System Issues

Kept for compatibility with FLTK versions before 1.4. Use preferentially fl_x11_xid(const Fl_Window *) with versions 1.4 and above.

Kept for compatibility with FLTK versions before 1.4. Use preferentially fl_x11_find(Window) with versions 1.4 and above.

Chapter 6

Topic Index

6.1 Topics

Here is a list of all topics with brief descriptions:

Version Numbers 247
Events handling functions 249
Pen and tablet event handling 265
Callback Function Typedefs 276
Windows handling functions 278
Selection & Clipboard functions 280
Screen functions 284
Color & Font functions 291
Drawing functions 307
Multithreading support functions 342
Cairo Support Functions and Classes 345
Unicode and UTF-8 functions 347
String handling functions 372
Safe widget deletion support functions 372
Mac OS X-specific symbols 375
Common Dialog Classes and Functions 376
File names and URI utility functions 393

Chapter 7

Namespace Index

7.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Fl
Fl is the FLTK global namespace containing state information and global methods for the current application 405
Fl::Pen
FLTK Pen/Stylus/Tablet input driver API 438

Chapter 8

Hierarchical Index

8.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

- Fl_Grid::Cell . . . 441
- Fl_Terminal::CharStyle . . . 442
- Fl_GIF_Image::GIF_FRAME::CPAL . . . 443
- Fl_Terminal::Cursor . . . 443
- Fl_Shared_Image::Deleter . . . 443
- Fl_Preferences::Entry . . . 444
- Fl_Terminal::EscapeSeq . . . 444
- Fl_Cairo_State . . . 554
- Fl_Callback_User_Data . . . 569
- FL_CHART_ENTRY . . . 582
- Fl_End . . . 691
- Fl_File_Chooser . . . 711
- Fl_File_Icon . . . 718
- Fl_Gl_Choice . . . 816
- Fl_Glut_Bitmap_Font . . . 837
- Fl_Glut_StrokeChar . . . 838
- Fl_Glut_StrokeFont . . . 838
- Fl_Glut_StrokeStrip . . . 838
- Fl_Glut_StrokeVertex . . . 838
- Fl_Help_Dialog . . . 894
- Fl_Image . . . 967
  - Fl_Bitmap . . . 469
    - Fl_XBM_Image . . . 2030
  - Fl_Pixmap . . . 1244
    - Fl_GIF_Image . . . 811
      - Fl_Anim_GIF_Image . . . 454
    - Fl_XPM_Image . . . 2033
  - Fl_RGB_Image . . . 1359
    - Fl_BMP_Image . . . 475
      - Fl_ICO_Image . . . 963
    - Fl_JPEG_Image . . . 1046
    - Fl_PNG_Image . . . 1255
    - Fl_PNM_Image . . . 1259
    - Fl_SVG_Image . . . 1540
  - Fl_Shared_Image . . . 1457
  - Fl_Tiled_Image . . . 1782
- Fl_Image_Reader . . . 977
- Fl_Label . . . 1050
- Fl_Mac_App_Menu . . . 1068

Fl_Menu_Item . . . 1115
Fl_Multi_Label . . . 1156
Fl_Native_File_Chooser . . . 1179
Fl_Plugin . . . 1249
  Fl_Device_Plugin . . . 665
Fl_Preferences . . . 1279
  Fl_Plugin_Manager . . . 1250
Fl_Rect . . . 1339
Fl_Scheme . . . 1392
Fl_Surface_Device . . . 1534
  Fl_Display_Device . . . 676
  Fl_Widget_Surface . . . 1982
    Fl_Copy_Surface . . . 651
    Fl_EPS_File_Surface . . . 692
    Fl_Image_Surface . . . 977
    Fl_Paged_Device . . . 1231
      Fl_PDF_File_Surface . . . 1237
      Fl_PostScript_File_Device . . . 1271
      Fl_Printer . . . 1300
    Fl_SVG_File_Surface . . . 1536
Fl_Text_Buffer . . . 1673
Fl_Text_Selection . . . 1761
Fl_Timeout . . . 1787
Fl_Tooltip . . . 1811
Fl_Tree_Item . . . 1866
Fl_Tree_Item_Array . . . 1887
Fl_Tree_Prefs . . . 1890
Fl_Widget . . . 1940
  Fl_Box . . . 479
  Fl_Button . . . 543
    Fl_Light_Button . . . 1051
      Fl_Check_Button . . . 598
      Fl_Radio_Light_Button . . . 1324
      Fl_Round_Button . . . 1376
        Fl_Radio_Round_Button . . . 1331
    Fl_Radio_Button . . . 1316
    Fl_Repeat_Button . . . 1342
    Fl_Return_Button . . . 1350
    Fl_Shortcut_Button . . . 1467
    Fl_Toggle_Button . . . 1803
  Fl_Chart . . . 570
  Fl_Clock_Output . . . 627
    Fl_Clock . . . 618
      Fl_Round_Clock . . . 1384
  Fl_FormsBitmap . . . 780
  Fl_FormsPixmap . . . 788
  Fl_FormsText . . . 795
  Fl_Free . . . 802
  Fl_Group . . . 875
    Fl_Browser_ . . . 519
      Fl_Browser . . . 488
        Fl_File_Browser . . . 696
        Fl_Hold_Browser . . . 913
        Fl_Multi_Browser . . . 1142
        Fl_Select_Browser . . . 1443
      Fl_Check_Browser . . . 583

- Fl_Color_Chooser . . . 638
- Fl_Flex . . . 752
- Fl_Grid . . . 854
- Fl_Help_View . . . 896
- Fl_Input_Choice . . . 1022
- Fl_Pack . . . 1220
- Fl_Scroll . . . 1405
- Fl_Spinner . . . 1511
- Fl_Table . . . 1564
  - Fl_Table_Row . . . 1590
- Fl_Tabs . . . 1606
- Fl_Terminal . . . 1629
- Fl_Text_Display . . . 1690
  - Fl_Text_Editor . . . 1738
- Fl_Tile . . . 1765
- Fl_Tree . . . 1816
- Fl_Window . . . 1988
  - Fl_Double_Window . . . 677
    - Fl_Cairo_Window . . . 555
    - Fl_Overlay_Window . . . 1205
  - Fl_Gl_Window . . . 816
    - Fl_Glut_Window . . . 838
  - Fl_Single_Window . . . 1485
    - Fl_Menu_Window . . . 1129
- Fl_Wizard . . . 2021

Fl_Input_ . . . 996
- Fl_Input . . . 983
  - Fl_File_Input . . . 723
  - Fl_Float_Input . . . 770
  - Fl_Int_Input . . . 1035
  - Fl_Multiline_Input . . . 1159
  - Fl_Output . . . 1195
    - Fl_Multiline_Output . . . 1169
  - Fl_Secret_Input . . . 1432
  - Fl_Spinner::Fl_Spinner_Input . . . 1524

Fl_Menu_ . . . 1069
- Fl_Choice . . . 606
  - Fl_Scheme_Choice . . . 1395
- Fl_Menu_Bar . . . 1092
  - Fl_Sys_Menu_Bar . . . 1548
- Fl_Menu_Button . . . 1104

Fl_Positioner . . . 1262

Fl_Progress . . . 1308

Fl_Timer . . . 1794

Fl_Valuator . . . 1895
- Fl_Adjuster . . . 444
- Fl_Counter . . . 655
  - Fl_Simple_Counter . . . 1476
- Fl_Dial . . . 667
  - Fl_Fill_Dial . . . 735
  - Fl_Line_Dial . . . 1060
- Fl_Roller . . . 1366
- Fl_Slider . . . 1498
  - Fl_Fill_Slider . . . 743
  - Fl_Hor_Fill_Slider . . . 927
  - Fl_Hor_Nice_Slider . . . 936
  - Fl_Hor_Slider . . . 945

Fl_Nice_Slider 1186
Fl_Scrollbar 1420
Fl_Value_Slider 1928
Fl_Hor_Value_Slider 954
Fl_Value_Input 1906
Fl_Value_Output 1917
Fl_Widget_Tracker 1986
Fl_GIF_Image::GIF_FRAME 2036
Fl_ICO_Image::IconDirEntry 2037
Fl_Text_Editor::Key_Binding 2037
Fl_Terminal::Margin 2038
Fl_Preferences::Name 2038
Fl_Preferences::Node 2039
Fl_Paged_Device::page_format 2040
Fl_Terminal::PartialUtf8Buf 2040
Fl_Terminal::RingBuffer 2040
Fl_Preferences::RootNode 2041
Fl_Scroll::Scrollbar_Data 2041
Fl_Scroll::ScrollInfo 2042
Fl_Terminal::Selection 2043
Fl_Tile::Size_Range 2043
Fl_Text_Display::Style_Table_Entry 2044
Fl::TimeoutData 2044
Fl_Terminal::Utf8Char 2044

Chapter 9

Class Index

9.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Fl_Grid::Cell . . . 441
Fl_Terminal::CharStyle . . . 442
Fl_GIF_Image::GIF_FRAME::CPAL . . . 443
Fl_Terminal::Cursor . . . 443
Fl_Shared_Image::Deleter
Custom deleter for std::unique_ptr<> . . . 443
Fl_Preferences::Entry . . . 444
Fl_Terminal::EscapeSeq . . . 444
Fl_Adjuster
Was stolen from Prisms, and has proven to be very useful for values that need a large dynamic range . . . 444
Fl_Anim_GIF_Image
Supports loading, caching, and drawing of animated Compuserve GIF^{SM} images . . . 454
Fl_Bitmap
Supports caching and drawing of mono-color (bitmap) images . . . 469
Fl_BMP_Image
Supports loading, caching, and drawing of Windows Bitmap (BMP) image files . . . 475
Fl_Box
This widget simply draws its box, and possibly its label . . . 479
Fl_Browser
Displays a scrolling list of text lines, and manages all the storage for the text . . . 488
Fl_Browser_
This is the base class for browsers . . . 519
Fl_Button
Buttons generate callbacks when they are clicked by the user . . . 543
Fl_Cairo_State
Contains all the necessary info on the current cairo context . . . 554
Fl_Cairo_Window
This defines an FLTK window with Cairo support . . . 555
Fl_Callback_User_Data
A class prototype that allows for additional data in callbacks . . . 569
Fl_Chart
Fl_Chart displays simple charts . . . 570
FL_CHART_ENTRY
For internal use only . . . 582
Fl_Check_Browser
Displays a scrolling list of text lines that may be selected and/or checked by the user . . . 583
Fl_Check_Button
A button with a "checkmark" to show its status . . . 598

Fl_Choice
A button that is used to pop up a menu . . . 606
Fl_Clock
This widget provides a round analog clock display . . . 618
Fl_Clock_Output
This widget can be used to display a program-supplied time . . . 627
Fl_Color_Chooser
Standard RGB color chooser . . . 638
Fl_Copy_Surface
Supports copying of graphical data to the clipboard . . . 651
Fl_Counter
Controls a single floating point value with button (or keyboard) arrows . . . 655
Fl_Device_Plugin
This plugin socket allows the integration of new device drivers for special window or screen types 665
Fl_Dial
Circular dial to control a single floating point value . . . 667
Fl_Display_Device
The computer's display . . . 676
Fl_Double_Window
The Fl_Double_Window provides a double-buffered window . . . 677
Fl_End
This is a dummy class that allows you to end a Fl_Group in a constructor list of a class: . . . 691
Fl_EPS_File_Surface
Encapsulated PostScript drawing surface . . . 692
Fl_File_Browser
Displays a list of filenames, optionally with file-specific icons . . . 696
Fl_File_Chooser
Displays a standard file selection dialog that supports various selection modes . . . 711
Fl_File_Icon
Manages icon images that can be used as labels in other widgets and as icons in the FileBrowser widget . . . 718
Fl_File_Input
This widget displays a pathname in a text input field . . . 723
Fl_Fill_Dial
Draws a dial with a filled arc . . . 735
Fl_Fill_Slider
Widget that draws a filled horizontal slider, useful as a progress or value meter . . . 743
Fl_Flex
Fl_Flex is a container (layout) widget for one row or one column of widgets . . . 752
Fl_Float_Input
Subclass of Fl_Input that only allows the user to type floating point numbers (sign, digits, decimal point, more digits, 'E' or 'e', sign, digits) . . . 770
Fl_FormsBitmap
Forms compatibility Bitmap Image Widget . . . 780
Fl_FormsPixmap
Forms pixmap drawing routines . . . 788
Fl_FormsText . . . 795
Fl_Free
Emulation of the Forms "free" widget . . . 802
Fl_GIF_Image
Supports loading, caching, and drawing of Compuserve GIF^{SM} images . . . 811
Fl_Gl_Choice . . . 816
Fl_Gl_Window
Sets things up so OpenGL works . . . 816
Fl_Glut_Bitmap_Font
Fltk glut font/size attributes used in the glutXXX functions . . . 837
Fl_Glut_StrokeChar . . . 838
Fl_Glut_StrokeFont . . . 838

Fl_Glut_StrokeStrip . . . 838
Fl_Glut_StrokeVertex . . . 838
Fl_Glut_Window
GLUT is emulated using this window class and these static variables (plus several more static variables hidden in glut_compatibility.cxx): . . . 838
Fl_Grid
Fl_Grid is a container (layout) widget with multiple columns and rows . . . 854
Fl_Group
Main FLTK container widget . . . 875
Fl_Help_Dialog
Displays a standard help dialog window using the Fl_Help_View widget . . . 894
Fl_Help_View
A widget to display formatted text, formatted in a subset of HTML . . . 896
Fl_Hold_Browser
The Fl_Hold_Browser is a subclass of Fl_Browser which lets the user select a single item, or no items by clicking on the empty space . . . 913
Fl_Hor_Fill_Slider . . . 927
Fl_Hor_Nice_Slider
Single thumb tab slider . . . 936
Fl_Hor_Slider
Horizontal Slider class . . . 945
Fl_Hor_Value_Slider . . . 954
Fl_ICO_Image
Supports loading, caching, and drawing of Windows icon (.ico) files . . . 963
Fl_Image
Base class for image caching, scaling and drawing . . . 967
Fl_Image_Reader . . . 977
Fl_Image_Surface
Directs all graphics requests to an Fl_Image . . . 977
Fl_Input
This is the FLTK text input widget . . . 983
Fl_Input_
This class provides a low-overhead text input field . . . 996
Fl_Input_Choice
A combination of the input widget and a menu button . . . 1022
Fl_Int_Input
Subclass of Fl_Input that only allows the user to type decimal digits (or hex numbers of the form 0xaef) . . . 1035
Fl_JPEG_Image
Supports loading, caching, and drawing of Joint Photographic Experts Group (JPEG) File Interchange Format (JFIF) images . . . 1046
Fl_Label
This struct stores all information for a text or mixed graphics label . . . 1050
Fl_Light_Button
This subclass displays the "on" state by turning on a light, rather than drawing pushed in . . . 1051
Fl_Line_Dial . . . 1060
Fl_Mac_App_Menu . . . 1068
Fl_Menu_
Base class of all widgets that have a menu in FLTK . . . 1069
Fl_Menu_Bar
This widget provides a standard menubar interface . . . 1092
Fl_Menu_Button
This is a button that when pushed pops up a menu (or hierarchy of menus) defined by an array of Fl_Menu_Item objects . . . 1104
Fl_Menu_Item
The Fl_Menu_Item structure defines a single menu item that is used by the Fl_Menu_ class . . . 1115
Fl_Menu_Window
Window type originally used for menus . . . 1129

Fl_Multi_Browser
Subclass of Fl_Browser which lets the user select any set of the lines 1142
Fl_Multi_Label
Allows a mixed text and/or graphics label to be applied to an Fl_Menu_Item or Fl_Widget . . . 1156
Fl_Multiline_Input
This input field displays '\n' characters as new lines rather than ^J, and accepts the Return, Tab, and up and down arrow keys 1159
Fl_Multiline_Output
This widget is a subclass of Fl_Output that displays multiple lines of text 1169
Fl_Native_File_Chooser
This class lets an FLTK application easily and consistently access the operating system's native file chooser 1179
Fl_Nice_Slider 1186
Fl_Output
This widget displays a piece of text 1195
Fl_Overlay_Window
This window provides double buffering and also the ability to draw the "overlay" which is another picture placed on top of the main image 1205
Fl_Pack
This widget was designed to add the functionality of compressing and aligning widgets 1220
Fl_Paged_Device
Represents page-structured drawing surfaces 1231
Fl_PDF_File_Surface
To send graphical output to a PDF file 1237
Fl_Pixmap
Supports caching and drawing of colormap (pixmap) images, including transparency 1244
Fl_Plugin
Fl_Plugin allows link-time and run-time integration of binary modules 1249
Fl_Plugin_Manager
Fl_Plugin_Manager manages link-time and run-time plugin binaries 1250
Fl_PNG_Image
Supports loading, caching, and drawing of Portable Network Graphics (PNG) image files . . . 1255
Fl_PNM_Image
Supports loading, caching, and drawing of Portable Anymap (PNM, PBM, PGM, PPM) image files 1259
Fl_Positioner
This class is provided for Forms compatibility 1262
Fl_PostScript_File_Device
To send graphical output to a PostScript file 1271
Fl_Preferences
Fl_Preferences store user settings between application starts 1279
Fl_Printer
OS-independent print support 1300
Fl_Progress
Displays a progress bar for the user 1308
Fl_Radio_Button 1316
Fl_Radio_Light_Button 1324
Fl_Radio_Round_Button 1331
Fl_Rect
Rectangle with standard FLTK coordinates (X, Y, W, H) 1339
Fl_Repeat_Button
The Fl_Repeat_Button is a subclass of Fl_Button that generates a callback when it is pressed and then repeatedly generates callbacks as long as it is held down 1342
Fl_Return_Button
The Fl_Return_Button is a subclass of Fl_Button that generates a callback when it is pressed or when the user presses the Enter key 1350
Fl_RGB_Image
Supports caching and drawing of full-color images with 1 to 4 channels of color information . . 1359

Fl_Roller
"dolly" control commonly used to move 3D objects . . . 1366
Fl_Round_Button
Buttons generate callbacks when they are clicked by the user . . . 1376
Fl_Round_Clock
A clock widget of type FL_ROUND_CLOCK . . . 1384
Fl_Scheme
Base class with static methods for future Fl_Scheme classes . . . 1392
Fl_Scheme_Choice . . . 1395
Fl_Scroll
This container widget lets you maneuver around a set of widgets much larger than your window 1405
Fl_Scrollbar
Displays a slider with arrow buttons at the ends of the scrollbar . . . 1420
Fl_Secret_Input
Subclass of Fl_Input that displays its input as a string of placeholders . . . 1432
Fl_Select_Browser
The class is a subclass of Fl_Browser which lets the user select a single item, or no items by clicking on the empty space . . . 1443
Fl_Shared_Image
This class supports caching, loading, and drawing of image files . . . 1457
Fl_Shortcut_Button
A button that allows the user to type a key combination to create shortcuts . . . 1467
Fl_Simple_Counter
This widget creates a counter with only 2 arrow buttons . . . 1476
Fl_Single_Window
This is the same as Fl_Window . . . 1485
Fl_Slider
Sliding knob inside a box . . . 1498
Fl_Spinner
This widget is a combination of a numerical input widget and repeat buttons . . . 1511
Fl_Spinner::Fl_Spinner_Input . . . 1524
Fl_Surface_Device
A drawing surface that's susceptible to receive graphical output . . . 1534
Fl_SVG_File_Surface
A drawing surface producing a Scalable Vector Graphics (SVG) file . . . 1536
Fl_SVG_Image
Supports loading, caching and drawing of scalable vector graphics (SVG) images . . . 1540
Fl_Sys_Menu_Bar
A class to create and modify menus that appear on macOS in the menu bar at the top of the screen . . . 1548
Fl_Table
A table of widgets or other content . . . 1564
Fl_Table_Row
A table with row selection capabilities . . . 1590
Fl_Tabs
Container widget that displays a set of tabs, with each tab representing a different child widget 1606
Fl_Terminal
Terminal widget supporting Unicode/utf-8, ANSI/xterm escape codes with full RGB color control 1629
Fl_Text_Buffer
This class manages Unicode text displayed in one or more Fl_Text_Display widgets . . . 1673
Fl_Text_Display
Rich text display widget . . . 1690
Fl_Text_Editor
This is the FLTK text editor widget . . . 1738
Fl_Text_Selection
This is an internal class for Fl_Text_Buffer to manage text selections . . . 1761
Fl_Tile
Lets you resize its children by dragging the border between them . . . 1765

Fl_Tiled_Image
This class supports tiling of images over a specified area 1782
Fl_Timeout
The internal class Fl_Timeout handles all timeout related functions 1787
Fl_Timer
This is provided only to emulate the Forms Timer widget 1794
Fl_Toggle_Button
The toggle button is a push button that needs to be clicked once to toggle on, and one more time to toggle off 1803
Fl_Tooltip
Tooltip support for all FLTK widgets 1811
Fl_Tree
Tree widget 1816
Fl_Tree_Item
Tree widget item 1866
Fl_Tree_Item_Array
Manages an array of Fl_Tree_Item pointers 1887
Fl_Tree_Prefs
Tree widget's preferences 1890
Fl_Valuator
Controls a single floating-point value and provides a consistent interface to set the value, range, and step, and insures that callbacks are done the same for every object 1895
Fl_Value_Input
Displays a numeric value 1906
Fl_Value_Output
Displays a floating point value 1917
Fl_Value_Slider
Fl_Slider widget with a box displaying the current value 1928
Fl_Widget
Fl_Widget is the base class for all widgets in FLTK 1940
Fl_Widget_Surface
A surface on which any FLTK widget can be drawn 1982
Fl_Widget_Tracker
This class should be used to control safe widget deletion 1986
Fl_Window
This widget produces an actual window 1988
Fl_Wizard
This widget is based off the Fl_Tabs widget, but instead of displaying tabs it only changes "tabs" under program control 2021
Fl_XBM_Image
Supports loading, caching, and drawing of X Bitmap (XBM) bitmap files 2030
Fl_XPM_Image
Supports loading, caching, and drawing of X Pixmap (XPM) images, including transparency . . 2033
Fl_GIF_Image::GIF_FRAME 2036
Fl_ICO_Image::IconDirEntry
Windows ICONDIRENTRY structure
2037
Fl_Text_Editor::Key_Binding
Simple linked list item associating a key/state to a function 2037
Fl_Terminal::Margin 2038
Fl_Preferences::Name
'Name' provides a simple method to create numerical or more complex procedural names for entries and groups on the fly 2038
Fl_Preferences::Node 2039
Fl_Paged_Device::page_format
Width, height and name of a page format 2040
Fl_Terminal::PartialUtf8Buf 2040
Fl_Terminal::RingBuffer 2040

Fl_Preferences::RootNode 2041
Fl_Scroll::Scrollbar_Data
A local struct to manage a scrollbar's xywh region and tab values 2041
Fl_Scroll::ScrollInfo
Structure to manage scrollbar and widget interior sizes 2042
Fl_Terminal::Selection 2043
Fl_Tile::Size_Range 2043
Fl_Text_Display::Style_Table_Entry
This structure associates the color, font, and font size of a string to draw with an attribute mask matching attr 2044
Fl::TimeoutData 2044
Fl_Terminal::Utf8Char 2044

Chapter 10

File Index

10.1 File List

Here is a list of all documented files with brief descriptions:

fl_config.h 2047
events.H
Global event handling variables and functions 2048
function_types.H
FLTK typedefs for service and widget callbacks 2054
options.H
Application and system wide options management 2056
pen_events.H
Pen event handling variables and functions 2058
Enumerations.H
This file contains type definitions and general enumerations 2062
filename.H
File names and URI utility functions 2093
Fl.H
Fl namespace 2096
Fl_Adjuster.H 2108
Fl_Anim_GIF_Image.H 2108
fl_ask.H
API for common dialogs 2110
fl_attr.h
This file defines compiler-specific macros 2114
Fl_Bitmap.H 2117
Fl_BMP_Image.H 2118
Fl_Box.H
Fl_Box widget 2118
Fl_Browser.H 2119
Fl_Browser_.H 2121
Fl_Button.H 2122
Fl_Cairo.H
Cairo is currently supported for the following platforms: Windows, macOS, Unix/Linux (X11 + Wayland) 2124
Fl_Cairo_Window.H
Fl_Cairo_Window, an FLTK window incorporating a Cairo draw callback 2125
fl_callback_macros.H
This file provides macros for easy function and method callbacks with multiple type safe arguments 2126
fl_casts.H 2134
Fl_Chart.H
Fl_Chart widget 2135
Fl_Check_Browser.H 2137

Fl_Check_Button.H . . . 2138
Fl_Choice.H . . . 2138
Fl_Clock.H . . . 2139
Fl_Color_Chooser.H
Fl_Color_Chooser widget . . . 2140
Fl_Copy_Surface.H . . . 2142
Fl_Counter.H . . . 2142
Fl_Device.H
Declaration of classes Fl_Surface_Device, Fl_Display_Device, Fl_Device_Plugin . . . 2143
Fl_Dial.H . . . 2145
Fl_Double_Window.H . . . 2145
fl_draw.H
Utility header to pull drawing functions together . . . 2146
Fl_Export.H . . . 2157
Fl_File_Browser.H . . . 2158
Fl_File_Chooser.H . . . 2159
Fl_File_Icon.H . . . 2161
Fl_File_Input.H . . . 2162
Fl_Fill_Dial.H . . . 2163
Fl_Fill_Slider.H . . . 2164
Fl_Flex.H . . . 2164
Fl_Float_Input.H . . . 2166
Fl_FormsBitmap.H . . . 2166
Fl_FormsPixmap.H . . . 2167
Fl_Free.H . . . 2167
Fl_GIF_Image.H . . . 2168
Fl_Gl_Window.H . . . 2169
Fl_Graphics_Driver.H . . . 2170
Fl_Grid.H
Fl_Grid container widget . . . 2173
Fl_Group.H
Fl_Group and Fl_End classes . . . 2177
Fl_Help_Dialog.H . . . 2179
Fl_Help_View.H . . . 2180
Fl_Hold_Browser.H . . . 2181
Fl_Hor_Fill_Slider.H . . . 2182
Fl_Hor_Nice_Slider.H . . . 2182
Fl_Hor_Slider.H . . . 2183
Fl_Hor_Value_Slider.H . . . 2183
Fl_ICO_Image.H . . . 2183
Fl_Image.H
Fl_Image, Fl_RGB_Image classes . . . 2184
Fl_Image_Surface.H . . . 2187
Fl_Input.H . . . 2188
Fl_Input_.H . . . 2189
Fl_Input_Choice.H . . . 2192
Fl_Int_Input.H . . . 2194
Fl_JPEG_Image.H . . . 2194
Fl_Light_Button.H . . . 2195
Fl_Line_Dial.H . . . 2195
Fl_Menu.H . . . 2195
Fl_Menu_.H . . . 2196
Fl_Menu_Bar.H . . . 2197
Fl_Menu_Button.H . . . 2198
Fl_Menu_Item.H . . . 2198
Fl_Menu_Window.H . . . 2202
fl_message.H . . . 2202
Fl_Multi_Browser.H . . . 2202

Fl_Multi_Label.H . . . 2203
Fl_Multiline_Input.H . . . 2203
Fl_Multiline_Output.H . . . 2204
Fl_Native_File_Chooser.H
Fl_Native_File_Chooser widget . . . 2204
Fl_Nice_Slider.H . . . 2207
Fl_Object.H . . . 2207
Fl_Output.H . . . 2207
Fl_Overlay_Window.H . . . 2208
Fl_Pack.H . . . 2208
Fl_Paged_Device.H
Declaration of class Fl_Paged_Device . . . 2209
Fl_PDF_File_Surface.H . . . 2211
Fl_Pixmap.H . . . 2211
Fl_Plugin.H . . . 2212
Fl_PNG_Image.H . . . 2213
Fl_PNM_Image.H . . . 2214
Fl_Positioner.H . . . 2214
Fl_PostScript.H
Declaration of classes Fl_PostScript_File_Device and Fl_EPS_File_Surface . . . 2215
Fl_Preferences.H . . . 2217
Fl_Printer.H
Declaration of class Fl_Printer . . . 2220
Fl_Progress.H . . . 2221
Fl_Radio_Button.H . . . 2221
Fl_Radio_Light_Button.H . . . 2222
Fl_Radio_Round_Button.H . . . 2222
Fl_Rect.H . . . 2223
Fl_Repeat_Button.H . . . 2224
Fl_Return_Button.H . . . 2224
Fl_RGB_Image.H . . . 2225
Fl_Roller.H . . . 2225
Fl_Round_Button.H . . . 2226
Fl_Round_Clock.H . . . 2226
Fl_Scheme.H . . . 2226
Fl_Scheme_Choice.H . . . 2227
Fl_Scroll.H . . . 2228
Fl_Scrollbar.H . . . 2229
Fl_Secret_Input.H . . . 2230
Fl_Select_Browser.H . . . 2230
Fl_Shared_Image.H
Fl_Shared_Image class . . . 2230
Fl_Shortcut_Button.H . . . 2233
fl_show_colormap.H
The fl_show_colormap() function hides the implementation classes used to provide the popup window and color selection mechanism . . . 2234
fl_show_input.H . . . 2234
Fl_Simple_Counter.H . . . 2235
Fl_Single_Window.H . . . 2235
Fl_Slider.H . . . 2236
Fl_Spinner.H . . . 2237
fl_string_functions.h
Public header for FLTK's platform-agnostic string handling . . . 2238
Fl_SVG_File_Surface.H . . . 2239
Fl_SVG_Image.H . . . 2240
Fl_Sys_Menu_Bar.H
Definition of class Fl_Sys_Menu_Bar . . . 2241
Fl_Table.H . . . 2242

Fl_Table_Row.H . . . 2248
Fl_Tabs.H . . . 2249
Fl_Terminal.H
Fl_Terminal widget . . . 2250
Fl_Text_Buffer.H . . . 2260
Fl_Text_Display.H . . . 2264
Fl_Text_Editor.H . . . 2269
Fl_Tile.H . . . 2270
Fl_Tiled_Image.H . . . 2271
Fl_Timer.H . . . 2272
Fl_Toggle_Button.H . . . 2272
Fl_Toggle_Light_Button.H . . . 2273
Fl_Toggle_Round_Button.H . . . 2273
Fl_Tooltip.H . . . 2274
Fl_Tree.H
This file contains the definitions of the Fl_Tree class . . . 2275
Fl_Tree_Item.H
This file contains the definitions for Fl_Tree_Item . . . 2278
Fl_Tree_Item_Array.H
This file defines a class that manages an array of Fl_Tree_Item pointers . . . 2283
Fl_Tree_Prefs.H
This file contains the definitions for Fl_Tree's preferences . . . 2284
fl_types.h
This file contains simple "C"-style type definitions . . . 2289
fl_utf8.h
Header for Unicode and UTF-8 character handling . . . 2290
Fl_Valuator.H . . . 2295
Fl_Value_Input.H . . . 2296
Fl_Value_Output.H . . . 2297
Fl_Value_Slider.H . . . 2297
Fl_Widget.H
Fl_Widget and Fl_Label classes . . . 2298
Fl_Widget_Surface.H . . . 2304
Fl_Widget_Tracker.H
Track widget deletion . . . 2305
Fl_Window.H
Fl_Window widget . . . 2306
Fl_Wizard.H . . . 2309
Fl_XBM_Image.H . . . 2310
Fl_XPM_Image.H . . . 2310
forms.H . . . 2311
gl.h
This file defines wrapper functions for OpenGL in FLTK . . . 2321
gl2opengl.h . . . 2325
gl_draw.H . . . 2326
glu.h . . . 2326
glut.H . . . 2327
mac.H
Mac OS X-specific symbols . . . 2333
math.h . . . 2335
names.h
This file defines arrays of human readable names for FLTK symbolic constants . . . 2336
platform.H . . . 2338
platform_types.h
Definitions of platform-dependent types . . . 2339
rgb_colors.H . . . 2342
wayland.H
Definitions of functions specific to the Wayland platform . . . 2348

win32.H
Definitions of functions specific to the Windows platform . . . 2350
win32_target.h . . . 2351
x.H . . . 2352
x11.H
Definitions of functions specific to the X11 platform . . . 2352
cgdebug.h . . . 2354
fastarrow.h . . . 2357
Fl.cxx
Implementation of the member functions of class Fl . . . 2357
fl_arc.cxx
Utility functions for drawing arcs and circles . . . 2359
fl_ask.cxx
Utility functions for common dialogs . . . 2359
fl_boxtype.cxx
Drawing code for common box types . . . 2361
fl_cmap.h . . . 2364
fl_color.cxx
Color handling . . . 2367
Fl_compose.cxx
Utility functions to support text input . . . 2367
fl_contrast.cxx
Color contrast handling . . . 2368
fl_curve.cxx
Utility for drawing Bézier curves, adding the points to the current fl_begin/fl_vertex/fl_end path . 2368
Fl_Double_Window.cxx
Fl_Double_Window implementation . . . 2369
Fl_get_system_colors.cxx
System color support . . . 2369
Fl_Gl_Choice.H . . . 2370
Fl_Gl_Window_Driver.H . . . 2371
Fl_Graphics_Driver.cxx
Implementation of class Fl_Graphics_Driver . . . 2372
Fl_Grid.cxx
Implements the Fl_Grid container widget . . . 2373
Fl_Image_Reader.h . . . 2373
Fl_Message.h . . . 2374
Fl_Native_File_Chooser_Kdialog.H . . . 2376
Fl_Native_File_Chooser_Zenity.H . . . 2377
fl_oxy.h . . . 2377
Fl_Paged_Device.cxx
Implementation of class Fl_Paged_Device . . . 2378
fl_plastic.cxx
Implementation of the `'plastic'` scheme . . . 2378
Fl_Private.H
Fl::Private namespace . . . 2378
fl_rect.cxx
Drawing and clipping routines for rectangles . . . 2381
Fl_Scalable_Graphics_Driver.H . . . 2381
Fl_Scheme.cxx
Implementation of the Fl_Scheme class . . . 2383
Fl_Screen_Driver.H . . . 2383
Fl_Sys_Menu_Bar_Driver.H . . . 2386
Fl_System_Driver.H . . . 2387
Fl_Timeout.cxx . . . 2389
Fl_Timeout.h
Fl_Timeout handling . . . 2389

fl_vertex.cxx
Portable drawing code for drawing arbitrary shapes with simple 2D transformations 2391
Fl_Window_Driver.H 2392
fl_write_jpeg.cxx
JPEG image support functions 2394
fl_write_png.cxx
PNG image support functions 2396
Fl_XColor.H 2398
flstring.h 2398
freeglut_teapot_data.h 2402
mediumarrow.h 2403
numericsort.c 2404
print_button.h 2405
print_panel.h 2406
slowarrow.h 2406
utf8_internal.h 2406
vsnprintf.c
Portable vsnprintf() implementation 2407
Xutf8.h 2408

Chapter 11

Topic Documentation

11.1 Version Numbers

FLTK defines some constants to help the programmer to find out, for which FLTK version a program is compiled.

Macros

- #define FL_ABI_VERSION 10500

  The FLTK ABI (Application Binary Interface) version number as an int.
- #define FL_API_VERSION (FL_MAJOR_VERSION∗10000 + FL_MINOR_VERSION∗100 + FL_PATCH_VERSION)

  The FLTK API version number as an int.
- #define FL_MAJOR_VERSION 1

  The major release version of this FLTK library.
- #define FL_MINOR_VERSION 5

  The minor release version for this library.
- #define FL_PATCH_VERSION 0

  The patch version for this library.
- #define FL_VERSION

  The FLTK version number as a double.

11.1.1 Detailed Description

FLTK defines some constants to help the programmer to find out, for which FLTK version a program is compiled.
The following constants are defined:

11.1.2 Macro Definition Documentation

11.1.2.1 FL_ABI_VERSION

`#define FL_ABI_VERSION 10500`

The FLTK ABI (Application Binary Interface) version number as an *int*.
FL_ABI_VERSION is an *int* that describes the major, minor, and patch ABI version numbers in the same format as FL_API_VERSION.
The ABI version number `FL_ABI_VERSION` is usually the same as the API version `FL_API_VERSION` with the last two digits set to '00'.
FLTK retains the ABI (Application Binary Interface) during patch releases of the same major and minor versions.
Examples:

```
FLTK Version  FL_API_VERSION  FL_ABI_VERSION  FL_VERSION (deprecated)
  1.3.0          10300           10300           1.0300
  1.3.4          10304           10300           1.0304
```

Version 1.2.3 is actually stored as 10203 to allow for more than 9 minor and patch releases.
The FL_MAJOR_VERSION, FL_MINOR_VERSION, and FL_PATCH_VERSION constants give the integral values for the major, minor, and patch releases respectively.
To enable new ABI-breaking features in patch releases you can configure FLTK to use a higher FL_ABI_VERSION. The highest allowed version is
FL_ABI_VERSION = FL_API_VERSION + 1
to allow for "next version" ABI features when FLTK is built from git or from a snapshot (pre-release version).

See also

README.abi-version.txt

11.1.2.2 FL_API_VERSION

```
#define FL_API_VERSION (FL_MAJOR_VERSION*10000 + FL_MINOR_VERSION*100 + FL_PATCH_VERSION)
```

The FLTK API version number as an *int*.
FL_API_VERSION is an *int* that describes the major, minor, and patch version numbers.
Version 1.2.3 is actually stored as 10203 to allow for more than 9 minor and patch releases.
The FL_MAJOR_VERSION, FL_MINOR_VERSION, and FL_PATCH_VERSION constants give the integral values for the major, minor, and patch releases respectively.

Note

FL_API_VERSION is intended to replace the deprecated *double* FL_VERSION.

See also

Fl::api_version()

11.1.2.3 FL_MAJOR_VERSION

```
#define FL_MAJOR_VERSION 1
```

The major release version of this FLTK library.

See also

FL_VERSION

11.1.2.4 FL_MINOR_VERSION

```
#define FL_MINOR_VERSION 5
```

The minor release version for this library.
FLTK remains mostly source-code compatible between minor version changes.

11.1.2.5 FL_PATCH_VERSION

```
#define FL_PATCH_VERSION 0
```

The patch version for this library.
FLTK remains binary compatible between patch versions.

11.1.2.6 FL_VERSION

```
#define FL_VERSION
```

Value:

```
( (double)FL_MAJOR_VERSION + \
(double)FL_MINOR_VERSION * 0.01 + \
(double)FL_PATCH_VERSION * 0.0001 )
```

The FLTK version number as a *double*.
FL_VERSION is a *double* that describes the major, minor, and patch version numbers.
Version 1.2.3 is actually stored as 1.0203 to allow for more than 9 minor and patch releases.

Deprecated This `double` version number is retained for compatibility with existing program code. New code should use *int* FL_API_VERSION instead. FL_VERSION is deprecated because comparisons of floating point values may fail due to rounding errors. However, there are currently no plans to remove this deprecated constant.

FL_VERSION is equivalent to *(double)FL_API_VERSION / 10000*.

See also

Fl::version() (deprecated as well)

FL_API_VERSION

Fl::api_version()

11.2 Events handling functions

Fl class events handling API declared in <FL/Fl.H>

Modules

- Pen and tablet event handling

  This chapter documents the Fl::Pen namespace API, declared in <FL/core/pen_events.H>

Functions

- void Fl::add_handler (Fl_Event_Handler ha)

  Install a function to parse unrecognized events.
- void Fl::add_handler (Fl_Event_Handler ha, Fl_Event_Handler before)

  Install a function to parse unrecognized events with less priority than another function.
- void Fl::add_system_handler (Fl_System_Handler h, void *data)

  Install a function to intercept system events.
- Fl_Widget * Fl::belowmouse ()

  Gets the widget that is below the mouse.
- void Fl::belowmouse (Fl_Widget *)

  Sets the widget that is below the mouse.
- Fl_Callback_Reason Fl::callback_reason ()

  Give the reason for calling a callback.
- int Fl::compose (int &del)

  Any text editing widget should call this for each FL_KEYBOARD event.
- void Fl::compose_reset ()

  If the user moves the cursor, be sure to call Fl::compose_reset().
- void Fl::disable_im ()

  Disables the system input methods facilities.
- void Fl::enable_im ()

  Enables the system input methods facilities.
- int Fl::event ()

  Returns the last event that was processed.
- int **Fl::event_alt** ()

  Returns non-zero if the Alt key is pressed.
- int Fl::event_button ()

  Gets which particular mouse button caused the current event.
- int Fl::event_button1 ()

  Returns non-zero if mouse button 1 is currently held down.
- int Fl::event_button2 ()

  Returns non-zero if mouse button 2 is currently held down.

- int Fl::event_button3 ()

  Returns non-zero if mouse button 3 is currently held down.
- int Fl::event_button4 ()

  Returns non-zero if mouse button 4 is currently held down.
- int Fl::event_button5 ()

  Returns non-zero if mouse button 5 is currently held down.
- int Fl::event_buttons ()

  Returns the mouse buttons state bits; if non-zero, then at least one button is pressed now.
- int Fl::event_clicks ()

  Returns non zero if we had a double click event.
- void Fl::event_clicks (int i)

  Manually sets the number returned by Fl::event_clicks().
- void * Fl::event_clipboard ()

  During an FL_PASTE event of non-textual data, returns a pointer to the pasted data.
- const char * Fl::event_clipboard_type ()

  Returns the type of the pasted data during an FL_PASTE event.
- int **Fl::event_command** ()

  Returns non-zero if the FL_COMMAND key is pressed, either FL_CTRL or on OSX FL_META.
- int **Fl::event_ctrl** ()

  Returns non-zero if the Control key is pressed.
- Fl_Event_Dispatch **Fl::event_dispatch** ()

  Return the current event dispatch function.
- void Fl::event_dispatch (Fl_Event_Dispatch d)

  Set a new event dispatch function.
- int Fl::event_dx ()

  Returns the current horizontal mouse scrolling associated with the FL_MOUSEWHEEL event.
- float Fl::event_dx_f ()

  Returns the current horizontal mouse scrolling associated with the FL_MOUSEWHEEL event.
- int Fl::event_dy ()

  Returns the current vertical mouse scrolling associated with the FL_MOUSEWHEEL event.
- float Fl::event_dy_f ()

  Returns the current vertical mouse scrolling associated with the FL_MOUSEWHEEL event.
- int Fl::event_inside (const Fl_Widget *)

  Returns whether or not the mouse event is inside a given child widget.
- int Fl::event_inside (int, int, int, int)

  Returns whether or not the mouse event is inside the given rectangle.
- int Fl::event_is_click ()

  Returns non-zero if the mouse has not moved far enough and not enough time has passed since the last FL_PUSH or FL_KEYBOARD event for it to be considered a "drag" rather than a "click".
- void Fl::event_is_click (int i)

  Clears the value returned by Fl::event_is_click().
- int Fl::event_key ()

  Gets which key on the keyboard was last pushed.
- int Fl::event_key (int key)

  Returns true if the given `key` was held down (or pressed) during the last event.
- int Fl::event_length ()

  Returns the length of the text in Fl::event_text().
- int Fl::event_original_key ()

  Returns the keycode of the last key event, regardless of the NumLock state.
- int **Fl::event_shift** ()

  Returns non-zero if the Shift key is pressed.

- int Fl::event_state ()

  Returns the keyboard and mouse button states of the last event.
- int Fl::event_state (int mask)

  Returns non-zero if any of the passed event state bits are turned on.
- const char * Fl::event_text ()

  Returns the text associated with the current event, including FL_PASTE or FL_DND_RELEASE events.
- int **Fl::event_x** ()

  Returns the mouse position of the event relative to the Fl_Window it was passed to.
- int Fl::event_x_root ()

  Returns the mouse position on the screen of the event.
- int **Fl::event_y** ()

  Returns the mouse position of the event relative to the Fl_Window it was passed to.
- int Fl::event_y_root ()

  Returns the mouse position on the screen of the event.
- std::string fl_callback_reason_str (int reason)

  Return the C++ symbol of an Fl_Callback_Reason as a string.
- std::string fl_eventname_str (int event)

  Return the C++ symbol of an Fl_Event as a string.
- std::string fl_fontname_str (int font)

  Return the C++ symbol of an Fl_Font as a string.
- Fl_Widget * Fl::focus ()

  Gets the current Fl::focus() widget.
- void Fl::focus (Fl_Widget *)

  Sets the widget that will receive FL_KEYBOARD events.
- int Fl::get_key (int key)

  Returns true if the given `key` is held down now.
- int Fl::get_mouse (int &X, int &Y)

  Return the number of the mouse-containing screen and computes the mouse screen coordinates.
- int Fl::handle (int, Fl_Window *)

  Handle events from the window system.
- int Fl::handle_ (int, Fl_Window *)

  Handle events from the window system.
- Fl_Event_Handler Fl::last_handler ()

  Returns the last function installed by a call to Fl::add_handler().
- Fl_Widget * Fl::pushed ()

  Gets the widget that is being pushed.
- void Fl::pushed (Fl_Widget *)

  Sets the widget that is being pushed.
- void Fl::remove_handler (Fl_Event_Handler h)

  Removes a previously added event handler.
- void Fl::remove_system_handler (Fl_System_Handler h)

  Removes a previously added system event handler.
- int Fl::test_shortcut (Fl_Shortcut)

  Tests the current event, which must be an FL_KEYBOARD or FL_SHORTCUT, against a shortcut value (described in Fl_Button).

Variables

- char const ∗const **Fl::clipboard_image** = "image"

  Denotes image data.
- char const ∗const **Fl::clipboard_plain_text** = "text/plain"

  Denotes plain textual data.
- const char ∗const fl_callback_reason_names [ ]

  This is an array of callback reason names you can use to convert callback reasons into names.
- const char ∗const fl_fontnames [ ]

  This is an array of font names you can use to convert font numbers into names.

11.2.1 Detailed Description

Fl class events handling API declared in <FL/Fl.H>

11.2.2 Function Documentation

11.2.2.1 add_handler() [1/2]

```
void Fl::add_handler (
            Fl_Event_Handler ha )  [extern]
```

Install a function to parse unrecognized events.
If FLTK cannot figure out what to do with an event, it calls each of these functions (most recent first) until one of them returns non-zero. If none of them returns non-zero then the event is ignored. Events that cause this to be called are:

- FL_SHORTCUT events that are not recognized by any widget. This lets you provide global shortcut keys.
- FL_SCREEN_CONFIGURATION_CHANGED events. Under X11, this event requires the libXrandr.so shared library to be loadable at run-time and the X server to implement the RandR extension.
- FL_ZOOM_EVENT events.
- FL_APP_ACTIVATE and FL_APP_DEACTIVATE events.
- System events that FLTK does not recognize. See fl_xevent.
- *Some* other events when the widget FLTK selected returns zero from its handle() method. Exactly which ones may change in future versions, however.

See also

Fl::remove_handler(Fl_Event_Handler)

Fl::event_dispatch(Fl_Event_Dispatch d)

Fl::handle(int, Fl_Window∗)

11.2.2.2 add_handler() [2/2]

```
void Fl::add_handler (
            Fl_Event_Handler ha,
            Fl_Event_Handler before )  [extern]
```

Install a function to parse unrecognized events with less priority than another function.
Install function `ha` to handle unrecognized events giving it the priority just lower than that of function `before` which was previously installed.

See also

Fl::add_handler(Fl_Event_Handler)

Fl::last_handler()

Since

1.4.0

11.2.2.3 add_system_handler()

```
void Fl::add_system_handler (
            Fl_System_Handler ha,
            void * data )  [extern]
```

Install a function to intercept system events.
FLTK calls each of these functions as soon as a new system event is received. The processing will stop at the first function to return non-zero. If all functions return zero then the event is passed on for normal handling by FLTK.
Each function will be called with a pointer to the system event as the first argument and `data` as the second argument. The system event pointer will always be void *, but will point to different objects depending on the platform:

- X11: XEvent
- Windows: MSG
- OS X: NSEvent
- Wayland: N/A (Fl::add_system_handler has no effect under Wayland)

Parameters

ha	The event handler function to register
data	User data to include on each call

See also

Fl::remove_system_handler(Fl_System_Handler)

11.2.2.4 belowmouse() [1/2]

```
Fl_Widget * Fl::belowmouse ( )  [inline]
```

Gets the widget that is below the mouse.

See also

belowmouse(Fl_Widget*)

11.2.2.5 belowmouse() [2/2]

```
void Fl::belowmouse (
            Fl_Widget * o )  [extern]
```

Sets the widget that is below the mouse.
This is for highlighting buttons. It is not used to send FL_PUSH or FL_MOVE directly, for several obscure reasons, but those events typically go to this widget. This is also the first widget tried for FL_SHORTCUT events.
If you change the belowmouse widget, the previous one and all parents (that don't contain the new widget) are sent FL_LEAVE events. Changing this does *not* send FL_ENTER to this or any widget, because sending FL_ENTER is supposed to *test* if the widget wants the mouse (by it returning non-zero from handle()).

11.2.2.6 callback_reason()

```
Fl_Callback_Reason Fl::callback_reason ( )  [extern]
```

Give the reason for calling a callback.

Returns

the reason for the current callback

See also

Fl_Widget::when(), Fl_Widget::do_callback(), Fl_Widget::callback()

Since

1.4.0

11.2.2.7 compose()

```
int Fl::compose (
            int & del ) [extern]
```

Any text editing widget should call this for each FL_KEYBOARD event.
Use of this function is very simple.
If *true* is returned, then it has modified the Fl::event_text() and Fl::event_length() to a set of *bytes* to insert (it may be of zero length!). It will also set the "del" parameter to the number of *bytes* to the left of the cursor to delete, this is used to delete the results of the previous call to Fl::compose().
If *false* is returned, the keys should be treated as function keys, and del is set to zero. You could insert the text anyways, if you don't know what else to do.
Text editing widgets can preferentially call fl_set_spot() to indicate the window coordinates of the bottom of the current insertion point and the line height. This way, auxiliary windows that help choosing among alternative characters with some text input methods appear just below or above the insertion point. If widgets don't do that, such auxiliary windows appear at the widget's bottom.
On some platforms, text input can involve marked text, that is, temporary text replaced by other text during the input process. This occurs, e.g., under Wayland or macOS when using dead keys or when entering CJK characters. Text editing widgets should preferentially signal marked text, usually underlining it. Widgets can use `int Fl::compose_state` *after* having called Fl::compose(int&) to obtain the length in bytes of marked text that always finishes at the current insertion point. Widgets should also call void fl_reset_spot() when processing FL_UNFOCUS events. The Fl_Input and Fl_Text_Editor widgets underline marked text. If none of this is done by a user-defined text editing widget, text input will work, but will not signal to the user what text is marked.
Finally, text editing widgets should call `set_flag(MAC_USE_ACCENTS_MENU);` in their constructor if they want to use, on the macOS platform, the feature introduced with Mac OS 10.7 "Lion" where pressing and holding certain keys on the keyboard opens a diacritic marks popup window.

Note

For compatibility with FLTK 1.3, text editing widgets can call `Fl::insertion_point_location(int x, int y, int height)` and `Fl::reset_marked_text()` only under the macOS platform to indicate/reset the coordinates of the current insertion point. This is deprecated in version 1.4 because redundant with the platform-independent fl_set_spot() and fl_reset_spot() functions.

11.2.2.8 compose_reset()

```
void Fl::compose_reset ( ) [extern]
```

If the user moves the cursor, be sure to call Fl::compose_reset().
The next call to Fl::compose() will start out in an initial state. In particular it will not set "del" to non-zero. This call is very fast so it is ok to call it many times and in many places.

11.2.2.9 disable_im()

```
void Fl::disable_im ( ) [extern]
```

Disables the system input methods facilities.

See also

enable_im()

11.2.2.10 enable_im()

`void Fl::enable_im ( ) [extern]`

Enables the system input methods facilities.
This is the default.

See also

disable_im()

11.2.2.11 event()

`int Fl::event ( ) [inline]`

Returns the last event that was processed.
This can be used to determine if a callback is being done in response to a keypress, mouse click, etc.

11.2.2.12 event_button()

`int Fl::event_button ( ) [inline]`

Gets which particular mouse button caused the current event.
This returns garbage if the most recent event was not a FL_PUSH or FL_RELEASE event.

Return values

FL_LEFT_MOUSE	
FL_MIDDLE_MOUSE	
FL_RIGHT_MOUSE	
FL_BACK_MOUSE	
FL_FORWARD_MOUSE	

See also

Fl::event_buttons(), Fl::event_state()

11.2.2.13 event_button1()

`int Fl::event_button1 ( ) [inline]`

Returns non-zero if mouse button 1 is currently held down.
For more details, see Fl::event_buttons().

11.2.2.14 event_button2()

`int Fl::event_button2 ( ) [inline]`

Returns non-zero if mouse button 2 is currently held down.
For more details, see Fl::event_buttons().

11.2.2.15 event_button3()

`int Fl::event_button3 ( ) [inline]`

Returns non-zero if mouse button 3 is currently held down.
For more details, see Fl::event_buttons().

11.2.2.16 event_button4()

`int Fl::event_button4 ( ) [inline]`

Returns non-zero if mouse button 4 is currently held down.
For more details, see Fl::event_buttons().

11.2.2.17 event_button5()

```
int Fl::event_button5 ( ) [inline]
```

Returns non-zero if mouse button 5 is currently held down.
For more details, see Fl::event_buttons().

11.2.2.18 event_buttons()

```
int Fl::event_buttons ( ) [inline]
```

Returns the mouse buttons state bits; if non-zero, then at least one button is pressed now.
This function returns the button state at the time of the event. During an FL_RELEASE event, the state of the released button will be 0. To find out, which button caused an FL_RELEASE event, you can use Fl::event_button() instead.

Returns

a bit mask value like { [FL_BUTTON1] | [FL_BUTTON2] | ... | [FL_BUTTON5] }

11.2.2.19 event_clicks() [1/2]

```
int Fl::event_clicks ( ) [inline]
```

Returns non zero if we had a double click event.

Return values

Non-zero	if the most recent FL_PUSH or FL_KEYBOARD was a "double click".
N-1	for N clicks. A double click is counted if the same button is pressed again while event_is_click() is true.

11.2.2.20 event_clicks() [2/2]

```
void Fl::event_clicks (
            int i ) [inline]
```

Manually sets the number returned by Fl::event_clicks().
This can be used to set it to zero so that later code does not think an item was double-clicked.

Parameters

in	*i*	corresponds to no double-click if 0, i+1 mouse clicks otherwise

See also

int event_clicks()

11.2.2.21 event_clipboard()

```
void * Fl::event_clipboard ( ) [inline]
```

During an FL_PASTE event of non-textual data, returns a pointer to the pasted data.
The returned data is an Fl_RGB_Image * when the result of Fl::event_clipboard_type() is Fl::clipboard_image.

11.2.2.22 event_clipboard_type()

```
const char * Fl::event_clipboard_type ( ) [inline]
```

Returns the type of the pasted data during an FL_PASTE event.
This type can be Fl::clipboard_plain_text or Fl::clipboard_image.

11.2.2.23 event_dispatch()

```
void Fl::event_dispatch (
            Fl_Event_Dispatch d ) [extern]
```

Set a new event dispatch function.
The event dispatch function is called after native events are converted to FLTK events, but before they are handled by FLTK. If the dispatch function Fl_Event_Dispatch d is set, it is up to the dispatch function to call Fl::handle_(int, Fl_Window*) or to ignore the event.
The dispatch function itself must return 0 if it ignored the event, or non-zero if it used the event. If you call Fl::handle_(), then this will return the correct value.
The event dispatch can be used to handle exceptions in FLTK events and callbacks before they reach the native event handler:

```
int myHandler(int e, Fl_Window *w) {
  try {
    return Fl::handle_(e, w);
  } catch () {
    ...
  }
}

main() {
  Fl::event_dispatch(myHandler);
  ...
  Fl::run();
}
```

Parameters

d	new dispatch function, or NULL

See also

Fl::add_handler(Fl_Event_Handler)

Fl::handle(int, Fl_Window*)

Fl::handle_(int, Fl_Window*)

11.2.2.24 event_dx()

```
int Fl::event_dx ( ) [inline]
```

Returns the current horizontal mouse scrolling associated with the FL_MOUSEWHEEL event.
Right is positive.

11.2.2.25 event_dx_f()

```
float Fl::event_dx_f ( ) [inline]
```

Returns the current horizontal mouse scrolling associated with the FL_MOUSEWHEEL event.
Right is positive.

11.2.2.26 event_dy()

```
int Fl::event_dy ( ) [inline]
```

Returns the current vertical mouse scrolling associated with the FL_MOUSEWHEEL event.
Down is positive.

11.2.2.27 event_dy_f()

```
float Fl::event_dy_f ( ) [inline]
```

Returns the current vertical mouse scrolling associated with the FL_MOUSEWHEEL event.
Down is positive.

11.2.2.28 event_inside() [1/2]

```
int Fl::event_inside (
            const Fl_Widget * o ) [extern]
```

Returns whether or not the mouse event is inside a given child widget.
Returns non-zero if the current Fl::event_x() and Fl::event_y() put it inside the given child widget's bounding box.

This method can only be used to check whether the mouse event is inside a **child** widget of the window that handles the event, and there must not be an intermediate subwindow (i.e. the widget must not be inside a subwindow of the current window). However, it is valid if the widget is inside a nested Fl_Group.
You must not use it with the window itself as the o argument in a window's handle() method.

Note

The mentioned restrictions are necessary, because this method does not transform coordinates of child widgets, and thus the given widget o must be within the *same* window that is handling the current event. Otherwise the results are undefined.

You should always call this rather than doing your own comparison so you are consistent about edge effects.

See also

Fl::event_inside(int, int, int, int)

Parameters

in	*o*	child widget to be tested

Returns

non-zero, if mouse event is inside the widget

11.2.2.29 event_inside() [2/2]

```
int Fl::event_inside (
            int xx,
            int yy,
            int ww,
            int hh )  [extern]
```

Returns whether or not the mouse event is inside the given rectangle.
Returns non-zero if the current Fl::event_x() and Fl::event_y() put it inside the given arbitrary bounding box.
You should always call this rather than doing your own comparison so you are consistent about edge effects.
To find out, whether the event is inside a child widget of the current window, you can use Fl::event_inside(const Fl_Widget *).

Parameters

in	*xx,yy,ww,hh*	bounding box

Returns

non-zero, if mouse event is inside

11.2.2.30 event_is_click() [1/2]

```
int Fl::event_is_click ( )  [inline]
```

Returns non-zero if the mouse has not moved far enough and not enough time has passed since the last FL_PUSH or FL_KEYBOARD event for it to be considered a "drag" rather than a "click".
You can test this on FL_DRAG, FL_RELEASE, and FL_MOVE events.

11.2.2.31 event_is_click() [2/2]

```
void Fl::event_is_click (
            int i )  [inline]
```

Clears the value returned by Fl::event_is_click().
Useful to prevent the *next* click from being counted as a double-click or to make a popup menu pick an item with a single click. Don't pass non-zero to this.

11.2.2.32 event_key() [1/2]

```
int Fl::event_key ( ) [inline]
```

Gets which key on the keyboard was last pushed.
The returned integer 'key code' is not necessarily a text equivalent for the keystroke. For instance: if someone presses '5' on the numeric keypad with numlock on, Fl::event_key() may return the 'key code' for this key, and NOT the character '5'. To always get the '5', use Fl::event_text() instead.

Returns

an integer 'key code', or 0 if the last event was not a key press or release.

See also

int event_key(int), event_text(), compose(int&).

11.2.2.33 event_key() [2/2]

```
int Fl::event_key (
            int key ) [extern]
```

Returns true if the given `key` was held down (or pressed) *during* the last event.
This is constant until the next event is read from the server.
Fl::get_key(int) returns true if the given key is held down *now*. Under X this requires a round-trip to the server and is *much* slower than Fl::event_key(int).
Keys are identified by the *unshifted* values. FLTK defines a set of symbols that should work on most modern machines for every key on the keyboard:

- All keys on the main keyboard producing a printable ASCII character use the value of that ASCII character (as though shift, ctrl, and caps lock were not on). The space bar is 32.
- All keys on the numeric keypad producing a printable ASCII character use the value of that ASCII character plus FL_KP (e.g., FL_KP + '4', FL_KP + '/'). The highest possible value is FL_KP_Last so you can range-check to see if something is on the keypad.
- All numbered function keys use the number on the function key plus FL_F. The highest possible number is FL_F_Last, so you can range-check a value.
- Buttons on the mouse are considered keys, and use the button number (where the left button is 1) plus FL_Button.
- All other keys on the keypad have a symbol: FL_Escape, FL_BackSpace, FL_Tab, FL_Enter, FL_Print, FL↩_Scroll_Lock, FL_Pause, FL_Insert, FL_Home, FL_Page_Up, FL_Delete, FL_End, FL_Page_Down, FL_Left, FL_Up, FL_Right, FL_Down, FL_Iso_Key, FL_Shift_L, FL_Shift_R, FL_Control_L, FL_Control_R, FL_Caps↩_Lock, FL_Alt_L, FL_Alt_R, FL_Meta_L, FL_Meta_R, FL_Menu, FL_Num_Lock, FL_KP_Enter. Be careful not to confuse these with the very similar, but all-caps, symbols used by Fl::event_state().

On X Fl::get_key(FL_Button+n) does not work.
On Windows Fl::get_key(FL_KP_Enter) and Fl::event_key(FL_KP_Enter) do not work.

11.2.2.34 event_length()

```
int Fl::event_length ( ) [inline]
```

Returns the length of the text in Fl::event_text().
There will always be a nul at this position in the text. However there may be a nul before that if the keystroke translates to a nul character or you paste a nul character.

11.2.2.35 event_original_key()

```
int Fl::event_original_key ( ) [inline]
```

Returns the keycode of the last key event, regardless of the NumLock state.
If NumLock is deactivated, FLTK translates events from the numeric keypad into the corresponding arrow key events. event_key() returns the translated key code, whereas event_original_key() returns the keycode before NumLock translation.

11.2.2.36 event_state() [1/2]

```
int Fl::event_state ( ) [inline]
```

Returns the keyboard and mouse button states of the last event.
This is a bitfield of what shift states were on and what mouse buttons were held down during the most recent event.

Note

FLTK platforms differ in what Fl::event_state() returns when it is called while a modifier key or mouse button is being pressed or released.

- Under X11 and Wayland, Fl::event_state() indicates the state of the modifier keys and mouse buttons just **prior** to the event. Thus, during the `FL_KEYDOWN` event generated when pressing the shift key, for example, the `FL_SHIFT` bit of event_state() is 0 and becomes 1 only at the next event which can be any other event, including e.g. `FL_MOVE`.
- Under other platforms the reported state of modifier keys or mouse buttons includes that of the key or button being pressed or released.
- Fl::event_state() returns the same value under all platforms when it's called while a non-modifier key (e.g. a letter or function key) is being pressed or released.
- X servers do not agree on shift states, and `FL_NUM_LOCK`, `FL_META`, and `FL_SCROLL_LOCK` may not work.
- The values were selected to match the XFree86 server on Linux.

Note

This inconsistency **may** be fixed (on X11 and Wayland) in a later release.

The legal event state bits are:

Device	State Bit	Key or Button	Since
Keyboard	FL_SHIFT	Shift	
Keyboard	FL_CAPS_LOCK	Caps Lock	
Keyboard	FL_CTRL	Ctrl	
Keyboard	FL_ALT	Alt	
Keyboard	FL_NUM_LOCK	Num Lock	
Keyboard	FL_META	Meta, e.g. "Windows"	
Keyboard	FL_SCROLL_LOCK	Scroll Lock	
Mouse	FL_BUTTON1	left button	
Mouse	FL_BUTTON2	middle button	
Mouse	FL_BUTTON3	right button	
Mouse	FL_BUTTON4	side button 1 (back)	1.3.10
Mouse	FL_BUTTON5	side button 2 (forward)	1.3.10

11.2.2.37 event_state() [2/2]

```
int Fl::event_state (
            int mask ) [inline]
```

Returns non-zero if any of the passed event state bits are turned on.
Use `mask` to pass the event states you're interested in. The legal event state bits are defined in Fl::event_state().

11.2.2.38 event_text()

```
const char * Fl::event_text ( ) [inline]
```

Returns the text associated with the current event, including FL_PASTE or FL_DND_RELEASE events.

This can be used in response to FL_KEYUP, FL_KEYDOWN, FL_PASTE, and FL_DND_RELEASE.
When responding to FL_KEYUP/FL_KEYDOWN, use this function instead of Fl::event_key() to get the text equivalent of keystrokes suitable for inserting into strings and text widgets.
The returned string is guaranteed to be NULL terminated. However, see Fl::event_length() for the actual length of the string, in case the string itself contains NULLs that are part of the text data.

Returns

A NULL terminated text string equivalent of the last keystroke.

11.2.2.39 event_x_root()

```
int Fl::event_x_root ( ) [inline]
```

Returns the mouse position on the screen of the event.
To find the absolute position of an Fl_Window on the screen, use the difference between event_x_root(),event_y_root() and event_x(),event_y().

11.2.2.40 event_y_root()

```
int Fl::event_y_root ( ) [inline]
```

Returns the mouse position on the screen of the event.
To find the absolute position of an Fl_Window on the screen, use the difference between event_x_root(),event_y_root() and event_x(),event_y().

11.2.2.41 fl_callback_reason_str()

```
std::string fl_callback_reason_str (
            int reason ) [inline]
```

Return the C++ symbol of an Fl_Callback_Reason as a string.

Parameters

in	*reason*	as an integer

Returns

C++ symbol of reason as a string

11.2.2.42 fl_eventname_str()

```
std::string fl_eventname_str (
            int event ) [inline]
```

Return the C++ symbol of an Fl_Event as a string.

Parameters

in	*event*	index as an integer

Returns

C++ symbol of event index as a string

11.2.2.43 fl_fontname_str()

```
std::string fl_fontname_str (
            int font ) [inline]
```

Return the C++ symbol of an Fl_Font as a string.

Parameters

in	*font*	index as an integer

Returns

C++ symbol of font index as a string

11.2.2.44 focus() [1/2]

```
Fl_Widget * Fl::focus ( ) [inline]
```

Gets the current Fl::focus() widget.

See also

Fl::focus(Fl_Widget*)

11.2.2.45 focus() [2/2]

```
void Fl::focus (
            Fl_Widget * o ) [extern]
```

Sets the widget that will receive FL_KEYBOARD events.
Use this function inside the `handle(int)` member function of a widget of yours to give focus to the widget, for example when it receives the FL_FOCUS or the FL_PUSH event. Otherwise, use Fl_Widget::take_focus() to give focus to a widget;
If you change Fl::focus(), the previous widget and all parents (that don't contain the new widget) are sent FL_↩UNFOCUS events. Changing the focus does *not* send FL_FOCUS to this or any widget, because sending FL_↩FOCUS is supposed to *test* if the widget wants the focus (by it returning non-zero from handle()).
Since FLTK 1.4.0 widgets can set the NEEDS_KEYBOARD flag to indicate that a keyboard is essential for the widget to function. Touchscreen devices will be sent a request to show an on-screen keyboard if no hardware keyboard is connected.

See also

Fl_Widget::take_focus()

Fl_Widget::needs_keyboard() const

Fl_Widget::needs_keyboard(bool)

11.2.2.46 get_key()

```
int Fl::get_key (
            int key ) [extern]
```

Returns true if the given `key` is held down *now*.
Under X this requires a round-trip to the server and is *much* slower than Fl::event_key(int).

See also

event_key(int)

11.2.2.47 get_mouse()

```
int Fl::get_mouse (
            int & X,
            int & Y ) [extern]
```

Return the number of the mouse-containing screen and computes the mouse screen coordinates.
This requires a round-trip query to the server. You should use Fl::event_x_root() and Fl::event_y_root() if possible, but this is necessary if you are not sure if a mouse event has been processed recently (such as to position your first window). If the display is not open, this will open it.

Parameters

X,Y	assigned with the mouse screen coordinates upon return

Returns

number of the mouse-containing screen

Since

1.5 added a function return value

11.2.2.48 handle()

```
int Fl::handle (
            int e,
            Fl_Window * window )  [extern]
```

Handle events from the window system.
This is called from the native event dispatch after native events have been converted to FLTK notation. This function calls Fl::handle_(int, Fl_Window*) unless the user sets a dispatch function. If a user dispatch function is set, the user must make sure that Fl::handle_() is called, or the event will be ignored.

Parameters

e	the event type (Fl::event_number() is not yet set)
window	the window that caused this event

Returns

0 if the event was not handled

See also

Fl::add_handler(Fl_Event_Handler)

Fl::event_dispatch(Fl_Event_Dispatch)

11.2.2.49 handle_()

```
int Fl::handle_ (
            int e,
            Fl_Window * window )  [extern]
```

Handle events from the window system.
This function is called from the native event dispatch, unless the user sets another dispatch function. In that case, the user dispatch function must decide when to call Fl::handle_(int, Fl_Window*)
Callbacks can set `FL_REASON_CLOSED` and `FL_REASON_CANCELLED`.

Parameters

e	the event type (Fl::event_number() is not yet set)
window	the window that caused this event

Returns

0 if the event was not handled

See also

Fl::event_dispatch(Fl_Event_Dispatch)

11.2.2.50 last_handler()

```
Fl_Event_Handler Fl::last_handler ( ) [extern]
```

Returns the last function installed by a call to Fl::add_handler().

Since

1.4.0

11.2.2.51 pushed() [1/2]

```
Fl_Widget * Fl::pushed ( ) [inline]
```

Gets the widget that is being pushed.

See also

void pushed(Fl_Widget*)

11.2.2.52 pushed() [2/2]

```
void Fl::pushed (
            Fl_Widget * o ) [extern]
```

Sets the widget that is being pushed.
FL_DRAG or FL_RELEASE (and any more FL_PUSH) events will be sent to this widget.
If you change the pushed widget, the previous one and all parents (that don't contain the new widget) are sent FL_RELEASE events. Changing this does *not* send FL_PUSH to this or any widget, because sending FL_PUSH is supposed to *test* if the widget wants the mouse (by it returning non-zero from handle()).

11.2.2.53 remove_handler()

```
void Fl::remove_handler (
            Fl_Event_Handler ha ) [extern]
```

Removes a previously added event handler.

See also

Fl::handle(int, Fl_Window*)

11.2.2.54 remove_system_handler()

```
void Fl::remove_system_handler (
            Fl_System_Handler ha ) [extern]
```

Removes a previously added system event handler.

Parameters

ha	The event handler function to remove

See also

Fl::add_system_handler(Fl_System_Handler)

11.2.2.55 test_shortcut()

```
int Fl::test_shortcut (
```

```
Fl_Shortcut shortcut )  [extern]
```

Tests the current event, which must be an FL_KEYBOARD or FL_SHORTCUT, against a shortcut value (described in Fl_Button).
Not to be confused with Fl_Widget::test_shortcut().

Returns

non-zero if there is a match.

11.2.3 Variable Documentation

11.2.3.1 fl_callback_reason_names

```
const char* const fl_callback_reason_names[]
```

Initial value:

```
=
{
  "FL_REASON_UNKNOWN",
  "FL_REASON_SELECTED",
  "FL_REASON_DESELECTED",
  "FL_REASON_RESELECTED",
  "FL_REASON_OPENED",
  "FL_REASON_CLOSED",
  "FL_REASON_DRAGGED",
  "FL_REASON_CANCELLED",
  "FL_REASON_CHANGED",
  "FL_REASON_GOT_FOCUS",
  "FL_REASON_LOST_FOCUS",
  "FL_REASON_RELEASED",
  "FL_REASON_ENTER_KEY",
  NULL, NULL, NULL,
  NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
  NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
  "FL_REASON_USER", "FL_REASON_USER+1", "FL_REASON_USER+2", "FL_REASON_USER+3",
}
```

This is an array of callback reason names you can use to convert callback reasons into names.
The array gets defined inline wherever your '#include <FL/names.h>' appears.

11.2.3.2 fl_fontnames

```
const char* const fl_fontnames[]
```

Initial value:

```
=
{
  "FL_HELVETICA",
  "FL_HELVETICA_BOLD",
  "FL_HELVETICA_ITALIC",
  "FL_HELVETICA_BOLD_ITALIC",
  "FL_COURIER",
  "FL_COURIER_BOLD",
  "FL_COURIER_ITALIC",
  "FL_COURIER_BOLD_ITALIC",
  "FL_TIMES",
  "FL_TIMES_BOLD",
  "FL_TIMES_ITALIC",
  "FL_TIMES_BOLD_ITALIC",
  "FL_SYMBOL",
  "FL_SCREEN",
  "FL_SCREEN_BOLD",
  "FL_ZAPF_DINGBATS",
}
```

This is an array of font names you can use to convert font numbers into names.
The array gets defined inline wherever your '#include <FL/names.h>' appears.

Example:

```
#include <FL/names.h>          // array will be defined here
int MyClass::my_callback(Fl_Widget *w, void*) {
    int fnum = w->labelfont();
    // Resulting output might be e.g. "Label's font is FL_HELVETICA (0)"
    printf("Label's font is %s (%d)\n", fl_fontnames[fnum], fnum);
    // ..resulting output might be e.g. "Label's font is FL_HELVETICA (0)"..
    [..]
}
```

11.2.4 Pen and tablet event handling

This chapter documents the Fl::Pen namespace API, declared in <FL/core/pen_events.H>

Enumerations

- enum Fl::Pen::Event {
  Fl::Pen::DETECTED = 0x1000 , Fl::Pen::CHANGED , Fl::Pen::IN_RANGE , Fl::Pen::OUT_OF_RANGE , Fl::Pen::ENTER , Fl::Pen::LEAVE , Fl::Pen::TOUCH , Fl::Pen::LIFT , Fl::Pen::HOVER , Fl::Pen::DRAW , Fl::Pen::BUTTON_PUSH , Fl::Pen::BUTTON_RELEASE }

  List of pen events.
- enum class Fl::Pen::State : uint32_t {
  Fl::Pen::State::NONE = 0x0000 , Fl::Pen::State::TIP_HOVERS = 0x0001 , Fl::Pen::State::TIP_DOWN = 0x0002 , Fl::Pen::State::ERASER_HOVERS = 0x0004 , Fl::Pen::State::ERASER_DOWN = 0x0008 , Fl::Pen::State::BUTTON0 = 0x0100 , Fl::Pen::State::BUTTON1 = 0x0200 , Fl::Pen::State::BUTTON2 = 0x0400 , Fl::Pen::State::BUTTON3 = 0x0800 , Fl::Pen::State::ANY_DOWN = BUTTON0 | BUTTON1 | BUTTON2 | BUTTON3 | TIP_DOWN | ERASER_DOWN }

  Bitfield of pen state flags.
- enum class Fl::Pen::Trait : uint32_t {
  Fl::Pen::Trait::NONE = 0x0000 , Fl::Pen::Trait::DRIVER_AVAILABLE = 0x0001 , Fl::Pen::Trait::DETECTED = 0x0002 , Fl::Pen::Trait::DISPLAY = 0x0004 , Fl::Pen::Trait::PEN_ID = 0x0008 , Fl::Pen::Trait::ERASER = 0x0010 , Fl::Pen::Trait::PRESSURE = 0x0020 , Fl::Pen::Trait::BARREL_PRESSURE = 0x0040 , Fl::Pen::Trait::TILT_X = 0x0080 , Fl::Pen::Trait::TILT_Y = 0x0100 , Fl::Pen::Trait::TWIST = 0x0200 , Fl::Pen::Trait::PROXIMITY = 0x0400 }

  Bitfield of traits.

Functions

- Trait Fl::Pen::driver_traits ()

  Query the traits supported by the pen/tablet driver.
- bool Fl::Pen::driver_traits (Trait bits)

  Return true if the corresponding bit is set in the driver traits.
- constexpr State Fl::Pen::operator& (State lhs, State rhs)

  Bitwise AND operator for State enum.
- constexpr Trait Fl::Pen::operator& (Trait lhs, Trait rhs)

  Bitwise AND operator for Trait enum.
- constexpr State Fl::Pen::operator| (State lhs, State rhs)

  Bitwise OR operator for State enum.
- constexpr Trait Fl::Pen::operator| (Trait lhs, Trait rhs)

  Bitwise OR operator for Trait enum.
- State & Fl::Pen::operator|= (State &lhs, State rhs)

  Bitwise OR assignment operator for State enum.
- Trait & Fl::Pen::operator|= (Trait &lhs, Trait rhs)

  Bitwise OR assignment operator for Trait enum.
- Trait Fl::Pen::pen_traits (int pen_id=0)

  Query traits of the current pen or stylus.
- bool Fl::Pen::pen_traits (Trait bits, int pen_id=0)

  Return true if the corresponding bit is set in the pen traits.
- void Fl::Pen::release ()

  Clear the "pushed" state and forward pen events as mouse events.
- void Fl::Pen::subscribe (Fl_Widget *widget)

  Receives a Pen::ENTER event when the pen enters this widget.
- void Fl::Pen::unsubscribe (Fl_Widget *widget)

  Stop receiving Pen::ENTER for this widget.

Query values during event handling

- double Fl::Pen::event_x ()

  Returns the pen x and y position inside the handling widget as doubles.
- double **Fl::Pen::event_y** ()

  Returns pen Y coordinate in widget space, see event_x().
- double Fl::Pen::event_x_root ()

  Returns the pen x and y position in global coordinates as doubles.
- double **Fl::Pen::event_y_root** ()

  Returns pen Y coordinate in screen space, see event_x_root().
- int Fl::Pen::event_pen_id ()

  Returns the ID of the pen used in the last event.
- double Fl::Pen::event_pressure ()

  Returns the pressure between the tip or eraser and the surface.
- double Fl::Pen::event_barrel_pressure ()

  Returns barrel pressure or tangential pressure.
- double Fl::Pen::event_tilt_x ()

  Returns the tilt of the pen in the x and y directions between -1 and 1.
- double **Fl::Pen::event_tilt_y** ()

  Returns pen Y-axis tilt, see event_tilt_x()
- double Fl::Pen::event_twist ()

  Returns the pen's axial rotation in degrees.
- double Fl::Pen::event_proximity ()

  Returns the proximity of the pen to the surface between 0 and 1.
- State Fl::Pen::event_state ()

  Returns the state of the various buttons and tips.
- bool Fl::Pen::event_state (State bits)

  Return true if any of the bits are set in the event state.
- State Fl::Pen::event_trigger ()

  Returns the state change that triggered the event.

11.2.4.1 Detailed Description

This chapter documents the Fl::Pen namespace API, declared in <FL/core/pen_events.H>
The FL::Pen namespace contains everything needed to work with a pen type input device, either in connection with an external tablet, or as a stylus for drawing directly onto a screen.
To receive pen input, call Fl::Pen::subscribe() for one or more widgets. The widget will receive a Fl::Pen::ENTER event when the stylus enters the widget area. By returning 1 to Fl::Pen::ENTER, all further pen events are sent to this widget, and no mouse events are generated until Fl::Pen::LEAVE.
Returning 0 Fl::Pen::ENTER tells FLTK to suppress further pen events until Fl::Pen::LEAVE, and convert them into mouse events instead.
Pen events also set Fl::event_x(), Fl::event_y(), Fl::event_x_root(), Fl::event_y_root(), Fl::event_is_click(), and Fl::event_clicks().

11.2.4.2 Enumeration Type Documentation

11.2.4.2.1 Event

`enum Fl::Pen::Event`
List of pen events.
These events extend the standard Fl_Event enumeration.

See also

enum Fl_Event

Enumerator

DETECTED	Pen entered the proximity of the tablet with a new pen.
CHANGED	Pen entered the proximity of the tablet with a known, but changed pen. User changed to a different pen (event_id() > 0) or the pen or tablet was disconnected (event_id() == -1). Pen IDs, if supported, are assigned by the tablet manufacturer.
IN_RANGE	Pen entered the proximity of the tablet with a known pen.
OUT_OF_RANGE	Pen left the proximity of the tablet.
ENTER	Pen entered the widget area, either by moving in x/y, or by a proximity change (pen gets closer to the surface). event_trigger() returns 0, TIP_HOVERS, or ERASER_HOVERS.
LEAVE	If no button is pressed, indicates that the pen left the widget area. While any pen button is held down, or the pen touches the surface, Fl::pushed() is set, and the pushed widgets receives DRAG events, even if the pen leaves the widget area. If all buttons are released outside the widget area, a LEAVE event is sent as well as LIFT or BUTTON_RELEASE.
TOUCH	Pen went from hovering to touching the surface. If the event handler returns 1, the widget will be marked Fl::pushed() and will receive DRAW events until the tip or eraser is released. event_trigger() returns TIP_DOWN or ERASER_DOWN.
LIFT	Pen went from touching to hovering over the surface. event_trigger() returns TIP_HOVERS or ERASER_HOVERS.
HOVER	Pen moved without touching the surface and no button is pressed.
DRAW	Pen moved while touching the surface, or any button is pressed.
BUTTON_PUSH	A pen button was pushed. If the event handler returns 1, the widget will be marked Fl::pushed() and will receive DRAW events until the button is released. event_trigger() returns BUTTON0, BUTTON1, BUTTON2, or BUTTON3.
BUTTON_RELEASE	A pen button was released. event_trigger() returns BUTTON0, BUTTON1, BUTTON2, or BUTTON3.

11.2.4.2.2 State

`enum class Fl::Pen::State : uint32_t [strong]`

Bitfield of pen state flags.

See also

event_state(), event_trigger()

Enumerator

NONE	No button pressed.
TIP_HOVERS	The tip hovers over the surface but does not touch it.
TIP_DOWN	The tip touches the surface.
ERASER_HOVERS	The eraser hovers over the surface but does not touch it.
ERASER_DOWN	The eraser touches the surface.
BUTTON0	Barrel button 0, usually the lower button on a pen, is pressed.
BUTTON1	Barrel button 1, usually the upper button on a pen, is pressed.
BUTTON2	Barrel button 2 is pressed.
BUTTON3	Barrel button 3 is pressed.
ANY_DOWN	Mask for all buttons, tip, and eraser down.

11.2.4.2.3 Trait

`enum class Fl::Pen::Trait : uint32_t [strong]`

Bitfield of traits.

This is used in Fl::Pen::driver_traits() and Fl::Pen::pen_traits().

Enumerator

NONE	No bits set.
DRIVER_AVAILABLE	Set if FLTK supports tablets and pens on this platform.
DETECTED	Set after the system detected a pen, stylus, or tablet. This bit may not be set until a pen is brought into proximity of the tablet.
DISPLAY	If set, this is a digitizer for a display; if clear, this is a standalone tablet.
PEN_ID	Driver provides different device IDs for different pens.
ERASER	Pen may have an eraser tip.
PRESSURE	Pen returns a pressure value.
BARREL_PRESSURE	Pen returns a barrel pressure value (tangential pressure)
TILT_X	Pen returns tilt in X direction.
TILT_Y	Pen returns tilt in Y direction.
TWIST	Pen returns a twist value.
PROXIMITY	Pen returns a proximity value.

11.2.4.3 Function Documentation

11.2.4.3.1 driver_traits() [1/2]

`Trait Fl::Pen::driver_traits ( ) [extern]`

Query the traits supported by the pen/tablet driver.

This function returns a bitfield of traits that are supported by the FLTK driver for this platform. If a trait is not supported, the corresponding event value will not return a useful value. Note that even if the FLTK driver support a trait, the underlying pen device or driver may not. Fl::Pen will return a known default for those event values.

The bitfield returned is static.

Returns

a bitfield of supported traits

See also

pen_traits()

11.2.4.3.2 driver_traits() [2/2]

```
bool Fl::Pen::driver_traits (
            Trait bits ) [inline]
```

Return true if the corresponding bit is set in the driver traits.

Parameters

in	*bits*	check for one or more trait bits

Returns

true if any bit is set

11.2.4.3.3 event_barrel_pressure()

`double Fl::Pen::event_barrel_pressure ( ) [extern]`
Returns barrel pressure or tangential pressure.

Returns

Pressure value from -1.0 to 1.0 , defaults to 0.0 .

See also

Trait::BARREL_PRESSURE

11.2.4.3.4 event_pen_id()

`int Fl::Pen::event_pen_id ( ) [extern]`
Returns the ID of the pen used in the last event.

Returns

Unique pen identifier, or -1 if pen was removed, defaults to 0

See also

Trait::PEN_ID

11.2.4.3.5 event_pressure()

`double Fl::Pen::event_pressure ( ) [extern]`
Returns the pressure between the tip or eraser and the surface.

Returns

pressure value from 0.0 (no pressure) to 1.0 (maximum pressure), defaults to 1.0.

See also

Trait::PRESSURE

11.2.4.3.6 event_proximity()

`double Fl::Pen::event_proximity ( ) [extern]`
Returns the proximity of the pen to the surface between 0 and 1.
A proximity of 0 is closest to the surface, 1 is farthest away.

Returns

Proximity value from 0.0 (touching) to 1.0 (far away), defaults to 0.0 .

See also

Trait::PROXIMITY

11.2.4.3.7 event_state() [1/2]

```
State Fl::Pen::event_state ( ) [extern]
```

Returns the state of the various buttons and tips.

Returns

Current state flags (combination of State values)

11.2.4.3.8 event_state() [2/2]

```
bool Fl::Pen::event_state (
            State bits ) [inline]
```

Return true if any of the bits are set in the event state.

Parameters

in	*bits*	check for one or more event state bits

Returns

true if any bit is set

11.2.4.3.9 event_tilt_x()

```
double Fl::Pen::event_tilt_x ( ) [extern]
```

Returns the tilt of the pen in the x and y directions between -1 and 1.
X-axis tilt returns -1.0 when the pen tilts all the way to the left, 0.0 when it is perfectly vertical, and 1.0 all the way to the right. Most pens seem to return a maximum range of -0.7 to 0.7.
Y-axis tilt returns -1.0 when the pen tilts away from the user, and 1.0 when it tilts toward the user.

Returns

Tilt value from -1.0 to 1.0, defaults to 0.0

See also

Trait::TILT_X, Trait::TILT_Y

11.2.4.3.10 event_trigger()

```
State Fl::Pen::event_trigger ( ) [extern]
```

Returns the state change that triggered the event.

Returns

a state with one bit set for the action that triggered this event

11.2.4.3.11 event_twist()

```
double Fl::Pen::event_twist ( ) [extern]
```

Returns the pen's axial rotation in degrees.
The twist angle is reported in degrees, with:

- 0° : pen's "top" (the button side) facing upward
- +180° : twisted halfway clockwise, looking at the end of the pen toward the tip
- –180° → twisted half way counter clockwise So the full range is usually –180° to +180°.

Returns

Twist angle in degrees, defaults to 0.0.

See also

Trait::TWIST

11.2.4.3.12 event_x()

```
double Fl::Pen::event_x ( ) [extern]
```

Returns the pen x and y position inside the handling widget as doubles.
These functions provide high-precision pen coordinates relative to the widget that received the pen event. For integer coordinates, use Fl::event_x() and Fl::event_y() instead.

Returns

Pen position as floating-point coordinate, defaults to 0.0

See also

Fl::event_x(), Fl::event_y()

11.2.4.3.13 event_x_root()

```
double Fl::Pen::event_x_root ( ) [extern]
```

Returns the pen x and y position in global coordinates as doubles.
For integer coordinates, use Fl::event_x_root() and Fl::event_y_root().

Returns

Pen position as floating-point coordinate in screen space, defaults to 0.0

See also

Fl::event_x_root(), Fl::event_y_root()

11.2.4.3.14 operator&() [1/2]

```
constexpr State Fl::Pen::operator& (
            State lhs,
            State rhs ) [inline], [constexpr]
```

Bitwise AND operator for State enum.

Parameters

lhs	Left-hand side state flags
rhs	Right-hand side state flags

Returns

Intersection of state flags

11.2.4.3.15 operator&() [2/2]

```
constexpr Trait Fl::Pen::operator& (
            Trait lhs,
            Trait rhs ) [inline], [constexpr]
```

Bitwise AND operator for Trait enum.

Parameters

lhs	Left-hand side trait flags
rhs	Right-hand side trait flags

Returns

Intersection of trait flags

11.2.4.3.16 operator"|() [1/2]

```
constexpr State Fl::Pen::operator| (
            State lhs,
            State rhs )  [inline], [constexpr]
```

Bitwise OR operator for State enum.

Parameters

lhs	Left-hand side state flags
rhs	Right-hand side state flags

Returns

Combined state flags

11.2.4.3.17 operator"|() [2/2]

```
constexpr Trait Fl::Pen::operator| (
            Trait lhs,
            Trait rhs )  [inline], [constexpr]
```

Bitwise OR operator for Trait enum.

Parameters

lhs	Left-hand side trait flags
rhs	Right-hand side trait flags

Returns

Combined trait flags

11.2.4.3.18 operator"|=() [1/2]

```
State & Fl::Pen::operator|= (
            State & lhs,
            State rhs )  [inline]
```

Bitwise OR assignment operator for State enum.

Parameters

lhs	Left-hand side state flags (modified in place)
rhs	Right-hand side state flags

Returns

Reference to modified lhs

11.2.4.3.19 operator"|=() [2/2]

```
Trait & Fl::Pen::operator|= (
            Trait & lhs,
            Trait rhs )  [inline]
```

Bitwise OR assignment operator for Trait enum.

Parameters

lhs	Left-hand side trait flags (modified in place)
rhs	Right-hand side trait flags

Returns

Reference to modified lhs

11.2.4.3.20 pen_traits() [1/2]

```
Trait Fl::Pen::pen_traits (
            int pen_id = 0 )  [extern]
```

Query traits of the current pen or stylus.
The value returned by this function may change when pens change or when more information becomes known about the currently used pen.

Parameters

in	*pen↩ _id*	a know pen ID as returned from event_pen_id(), or 0 for the current pen

Returns

a bitfield of supported traits

11.2.4.3.21 pen_traits() [2/2]

```
bool Fl::Pen::pen_traits (
            Trait bits,
            int pen_id = 0 )  [inline]
```

Return true if the corresponding bit is set in the pen traits.

Parameters

in	*bits*	check for one or more trait bits
in	*pen↩ _id*	a know pen ID as returned from event_pen_id(), or 0 for the current pen

Returns

true if any bit is set

11.2.4.3.22 release()

```
void Fl::Pen::release ( ) [extern]
```

Clear the "pushed" state and forward pen events as mouse events.
Call this if another window is popped up during pen event handling, so mouse event handling can resume normal.

11.2.4.3.23 subscribe()

```
void Fl::Pen::subscribe (
            Fl_Widget * widget ) [extern]
```

Receives a Pen::ENTER event when the pen enters this widget.
Multiple widgets may subscribe to pen events. Additional subscription requests for the same widget are ignored.

Parameters

widget	The widget subscribing to pen events.

11.2.4.3.24 unsubscribe()

```
void Fl::Pen::unsubscribe (
            Fl_Widget * widget ) [extern]
```

Stop receiving Pen::ENTER for this widget.
Deleting a widget will automatically unsubscribe it.

Parameters

widget	Widget to unsubscribe from pen events

11.3 Callback Function Typedefs

Typedefs defined in <FL/core/function_types.H> for callback or handler functions passed as function parameters.

Typedefs

- typedef void(∗ **Fl_Abort_Handler**) (const char ∗format,...)

  Signature of set_abort functions passed as parameters.
- typedef int(∗ **Fl_Args_Handler**) (int argc, char ∗∗argv, int &i)

  Signature of args functions passed as parameters.
- typedef void(∗ **Fl_Atclose_Handler**) (Fl_Window ∗window, void ∗data)

  Signature of set_atclose functions passed as parameters.
- typedef void(∗ **Fl_Awake_Handler**) (void ∗data)

  Signature of some wakeup callback functions passed as parameters.
- typedef void() **Fl_Box_Draw_F**(int x, int y, int w, int h, Fl_Color color)

  Signature of some box drawing functions passed as parameters.
- typedef void() **Fl_Box_Draw_Focus_F**(Fl_Boxtype bt, int x, int y, int w, int h, Fl_Color fg, Fl_Color bg)

  Signature of box focus frame drawing functions.
- typedef void(∗ **Fl_Clipboard_Notify_Handler**) (int source, void ∗data)

  Signature of add_clipboard_notify functions passed as parameters.
- typedef int(∗ Fl_Event_Dispatch) (int event, Fl_Window ∗w)

Signature of event_dispatch functions passed as parameters.

- typedef int(∗ **Fl_Event_Handler**) (int event)

  Signature of add_handler functions passed as parameters.
- typedef void(∗ **Fl_FD_Handler**) (FL_SOCKET fd, void ∗data)

  Signature of add_fd functions passed as parameters.
- typedef void(∗ Fl_Idle_Handler) (void ∗data)

  Signature of add_idle callback function passed as parameter.
- typedef void() **Fl_Label_Draw_F**(const Fl_Label ∗label, int x, int y, int w, int h, Fl_Align align)

  Signature of some label drawing functions passed as parameters.
- typedef void() **Fl_Label_Measure_F**(const Fl_Label ∗label, int &width, int &height)

  Signature of some label measurement functions passed as parameters.
- typedef void(∗ Fl_Old_Idle_Handler) ()

  Signature of add_idle callback function passed as parameter.
- typedef int(∗ **Fl_System_Handler**) (void ∗event, void ∗data)

  Signature of add_system_handler functions passed as parameters.
- typedef void(∗ Fl_Timeout_Handler) (void ∗data)

  Signature of timeout callback functions passed as parameters.

11.3.1 Detailed Description

Typedefs defined in <FL/core/function_types.H> for callback or handler functions passed as function parameters.
FLTK uses callback functions as parameters for some function calls, e.g. to set up global event handlers (Fl::add_handler()), to add a timeout handler (Fl::add_timeout()), and many more.
The typedefs defined in this group describe the function parameters used to set up or clear the callback functions and should also be referenced to define the callback function to handle such events in the user's code.

See also

Fl::add_handler(), Fl::add_timeout(), Fl::repeat_timeout(), Fl::remove_timeout() and others

11.3.2 Typedef Documentation

11.3.2.1 Fl_Event_Dispatch

`typedef int(* Fl_Event_Dispatch) (int event, Fl_Window *w)`
Signature of event_dispatch functions passed as parameters.

See also

Fl::event_dispatch(Fl_Event_Dispatch)

11.3.2.2 Fl_Idle_Handler

`typedef void(* Fl_Idle_Handler) (void *data)`
Signature of add_idle callback function passed as parameter.
This signature allows an idle callback to use one parameter as an arbitrary `data` value.

11.3.2.3 Fl_Old_Idle_Handler

`typedef void(* Fl_Old_Idle_Handler) ()`
Signature of add_idle callback function passed as parameter.
This signature allows an idle callback without parameters. When the callback is called it is called with an additional parameter (set to nullptr) which is not used by the callback.

11.3.2.4 Fl_Timeout_Handler

`typedef void(* Fl_Timeout_Handler) (void *data)`
Signature of timeout callback functions passed as parameters.
Please see Fl::add_timeout() for details.

11.4 Windows handling functions

Windows and standard dialogs handling declared in <FL/Fl.H>

Functions

- void Fl::default_atclose (Fl_Window *, void *)

  Default callback for window widgets.
- Fl_Window * Fl::first_window ()

  Returns the first top-level window in the list of shown() windows.
- void Fl::first_window (Fl_Window *)

  Sets the window that is returned by first_window().
- Fl_Window * Fl::grab ()

  Returns the window that currently receives all events.
- void Fl::grab (Fl_Window *)

  Selects the window to grab.
- Fl_Window * Fl::modal ()

  Returns the top-most modal() window currently shown.
- Fl_Window * Fl::next_window (const Fl_Window *)

  Returns the next top-level window in the list of shown() windows.
- void **Fl::set_abort** (Fl_Abort_Handler f)

  For back compatibility, sets the void Fl::fatal handler callback.
- void Fl::set_atclose (Fl_Atclose_Handler f)

  For back compatibility, sets the Fl::atclose handler callback.

Variables

- void(* Fl::atclose)(Fl_Window *, void *) = Fl::default_atclose

  Back compatibility: default window callback handler.

11.4.1 Detailed Description

Windows and standard dialogs handling declared in <FL/Fl.H>

11.4.2 Function Documentation

11.4.2.1 default_atclose()

```
void Fl::default_atclose (
            Fl_Window * window,
            void * v )  [extern]
```

Default callback for window widgets.
It hides the window and then calls the default widget callback.

11.4.2.2 first_window() [1/2]

```
Fl_Window * Fl::first_window ( )  [extern]
```

Returns the first top-level window in the list of shown() windows.
If a modal() window is shown this is the top-most modal window, otherwise it is the most recent window to get an event.

11.4.2.3 first_window() [2/2]

```
void Fl::first_window (
            Fl_Window * window )  [extern]
```

Sets the window that is returned by first_window().
The window is removed from wherever it is in the list and inserted at the top. This is not done if Fl::modal() is on or if the window is not shown(). Because the first window is used to set the "parent" of modal windows, this is often useful.

11.4.2.4 grab() [1/2]

```
Fl_Window * Fl::grab ( )  [inline]
```

Returns the window that currently receives all events.

Returns

The window that currently receives all events, or NULL if event grabbing is currently OFF.

11.4.2.5 grab() [2/2]

```
void Fl::grab (
            Fl_Window * win )  [extern]
```

Selects the window to grab.
This is used when pop-up menu systems are active.
Send all events to the passed window no matter where the pointer or focus is (including in other programs). The window *does not have to be shown()* , this lets the handle() method of a "dummy" window override all event handling and allows you to map and unmap a complex set of windows (under both X and Windows *some* window must be mapped because the system interface needs a window id).
If grab() is on it will also affect show() of windows by doing system-specific operations (on X it turns on override-redirect). These are designed to make menus popup reliably and faster on the system.
To turn off grabbing do Fl::grab(0).
Be careful that your program does not enter an infinite loop while grab() is on. On X this will lock up your screen!
To avoid this potential lockup, all newer operating systems seem to limit mouse pointer grabbing to the time during which a mouse button is held down. Some OS's may not support grabbing at all.

11.4.2.6 modal()

```
Fl_Window * Fl::modal ( )  [inline]
```

Returns the top-most modal() window currently shown.
This is the most recently shown() window with modal() true, or NULL if there are no modal() windows shown(). The modal() window has its handle() method called for all events, and no other windows will have handle() called (grab() overrides this).

11.4.2.7 next_window()

```
Fl_Window * Fl::next_window (
            const Fl_Window * window )  [extern]
```

Returns the next top-level window in the list of shown() windows.
You can use this call to iterate through all the windows that are shown().

Parameters

in	*window*	must be shown and not NULL

11.4.2.8 set_atclose()

```
void Fl::set_atclose (
            Fl_Atclose_Handler f )  [inline]
```

For back compatibility, sets the Fl::atclose handler callback.
You can now simply change the callback for the window instead.

See also

Fl_Window::callback(Fl_Callback∗)

11.4.3 Variable Documentation

11.4.3.1 atclose

```
void(* Fl::atclose)(Fl_Window *, void *) (
            Fl_Window * ,
            void *  ) = Fl::default_atclose  [extern]
```

Back compatibility: default window callback handler.

See also

Fl::set_atclose()

11.5 Selection & Clipboard functions

FLTK global copy/cut/paste functions declared in <FL/Fl.H>

Functions

- void Fl::add_clipboard_notify (Fl_Clipboard_Notify_Handler h, void ∗data=0)

  FLTK will call the registered callback whenever there is a change to the selection buffer or the clipboard.
- int Fl::clipboard_contains (const char ∗type)

  Returns non 0 if the clipboard contains data matching `type`.
- void Fl::copy (const char ∗stuff, int len, int destination=0, const char ∗type=Fl::clipboard_plain_text)

  Copies data to the selection buffer, the clipboard, or both.
- int Fl::dnd ()

  Initiate a Drag And Drop operation.
- void Fl::paste (Fl_Widget &receiver)

  Backward compatibility only.
- void Fl::paste (Fl_Widget &receiver, int source, const char ∗type=Fl::clipboard_plain_text)

  Pastes the data from the selection buffer (`source` is 0) or the clipboard (`source` is 1) into `receiver`.
- void **Fl::remove_clipboard_notify** (Fl_Clipboard_Notify_Handler h)

  Stop calling the specified callback when there are changes to the selection buffer or the clipboard.
- void Fl::selection (Fl_Widget &owner, const char ∗, int len)

  Changes the current selection.
- Fl_Widget ∗ Fl::selection_owner ()

  back-compatibility only: Gets the widget owning the current selection
- void Fl::selection_owner (Fl_Widget ∗)

  Back-compatibility only: The single-argument call can be used to move the selection to another widget or to set the owner to NULL, without changing the actual text of the selection.
- int Fl::selection_to_clipboard ()

  Returns the current selection_to_clipboard mode.
- void Fl::selection_to_clipboard (int mode)

  Copies selections on X11 directly to the clipboard if enabled.

11.5.1 Detailed Description

FLTK global copy/cut/paste functions declared in <FL/Fl.H>

11.5.2 Function Documentation

11.5.2.1 add_clipboard_notify()

```
void Fl::add_clipboard_notify (
            Fl_Clipboard_Notify_Handler h,
            void * data = 0 )  [extern]
```

FLTK will call the registered callback whenever there is a change to the selection buffer or the clipboard.
The source argument indicates which of the two has changed. Only changes by other applications are reported.
Example:

```
void clip_callback(int source, void *data) {
    if ( source == 0 ) printf("CLIP CALLBACK: selection buffer changed\n");
    if ( source == 1 ) printf("CLIP CALLBACK: clipboard changed\n");
}
[..]
int main() {
    [..]
    Fl::add_clipboard_notify(clip_callback);
    [..]
}
```

Note

Some systems require polling to monitor the clipboard and may therefore have some delay in detecting changes.

11.5.2.2 clipboard_contains()

```
int Fl::clipboard_contains (
            const char * type )  [extern]
```

Returns non 0 if the clipboard contains data matching `type`.
The clipboard can contain both text and image data; in that situation this function returns non 0 to both requests. This function is *not* meant to check whether the clipboard is empty. This function does not allow to query the selection buffer because FLTK allows to copy/paste non-textual data only from/to the clipboard.

Parameters

type	can be Fl::clipboard_plain_text or Fl::clipboard_image.

11.5.2.3 copy()

```
void Fl::copy (
            const char * stuff,
            int len,
            int destination = 0,
            const char * type = Fl::clipboard_plain_text )  [extern]
```

Copies data to the selection buffer, the clipboard, or both.
The `destination` can be:

- 0: selection buffer (see note below)
- 1: clipboard
- 2: both

The selection buffer exists only on the X11 platform and is used for middle-mouse pastes and for drag-and-drop selections. The clipboard is used for traditional copy/cut/paste operations. On all other platforms the selection buffer (`destination` = 0) is mapped to the clipboard, i.e. on platforms other than X11 all `destinations` are equivalent and the data is always copied to the clipboard.

Note

Please see Fl::selection_to_clipboard() to enable duplication of the selection buffer to the clipboard on X11, i.e. if `destination` = 0 (selection buffer) **and** Fl::selection_to_clipboard() is enabled, then the data is copied to both the selection buffer and the clipboard. This makes the X11 behavior similar to other platforms but keeps the selection buffer for X11 specific inter process communication.

`type` should always be `Fl::clipboard_plain_text` which is the default. Other values are ignored and reserved for future extensions.

Note

This function is, at present, intended only to copy UTF-8 encoded textual data. To copy graphical data, use the Fl_Copy_Surface class. The `type` argument may allow to copy other kinds of data in the future.

Parameters

in	*stuff*	text data to be copied
in	*len*	the number of relevant bytes in `stuff`
in	*destination*	0 = selection, 1 = clipboard, 2 = both (see description)
in	*type*	usually plain text (see description)

11.5.2.4 dnd()

```
int Fl::dnd ( ) [extern]
```

Initiate a Drag And Drop operation.
The selection buffer should be filled with relevant data before calling this method. FLTK will then initiate the system wide drag and drop handling. Dropped data will be marked as *text*.
Create a selection first using: Fl::copy(const char ∗stuff, int len, 0)

11.5.2.5 paste() [1/2]

```
void Fl::paste (
            Fl_Widget & receiver ) [extern]
```

Backward compatibility only.
This calls Fl::paste(receiver, 0);

See also

Fl::paste(Fl_Widget &receiver, int clipboard, const char∗ type)

11.5.2.6 paste() [2/2]

```
void Fl::paste (
            Fl_Widget & receiver,
            int source,
            const char * type = Fl::clipboard_plain_text ) [extern]
```

Pastes the data from the selection buffer (`source` is 0) or the clipboard (`source` is 1) into `receiver`.
The selection buffer (`source` is 0) is used for middle-mouse pastes and for drag-and-drop selections. The clipboard (`source` is 1) is used for copy/cut/paste operations.
If `source` is 1, the optional `type` argument indicates what type of data is requested from the clipboard. At present, Fl::clipboard_plain_text (requesting text data) and Fl::clipboard_image (requesting image data) are possible. Set things up so the handle function of the `receiver` widget will be called with an FL_PASTE event some time in the future if the clipboard does contain data of the requested type.
The handle function of `receiver` can process the FL_PASTE event as follows:

- If the `receiver` widget is known to only receive text data, the text string from the specified `source` is in Fl::event_text() with UTF-8 encoding, and the number of bytes is in Fl::event_length(). If Fl::paste() gets called during the drop step of a files-drag-and-drop operation, Fl::event_text() contains a list of filenames (see Drag and Drop Events).

- If the `receiver` widget can potentially receive non-text data, use Fl::event_clipboard_type() to determine what sort of data is being sent. If Fl::event_clipboard_type() returns Fl::clipboard_plain_text, proceed as above. It it returns Fl::clipboard_image, the pointer returned by Fl::event_clipboard() can be safely cast to type Fl_RGB_Image * to obtain a pointer to the pasted image. If `receiver` accepts the clipboard image, receiver.handle() should return 1 and the application should take ownership of this image (that is, delete it after use). Conversely, if receiver.handle() returns 0, the application must not use the image.

The receiver should be prepared to be called *directly* by this, or for it to happen *later*, or possibly *not at all*. This allows the window system to take as long as necessary to retrieve the paste buffer (or even to screw up completely) without complex and error-prone synchronization code in FLTK.

Platform details for image data:

- Unix/Linux platform: Clipboard images in PNG or BMP formats are recognized. Requires linking with the fltk_images library.
- Windows platform: Both bitmap and vectorial (Enhanced metafile) data from clipboard can be pasted as image data.
- Mac OS X platform: Both bitmap (TIFF) and vectorial (PDF) data from clipboard can be pasted as image data.

11.5.2.7 selection()

```
void Fl::selection (
            Fl_Widget & owner,
            const char * text,
            int len )  [extern]
```

Changes the current selection.
The block of text is copied to an internal buffer by FLTK (be careful if doing this in response to an FL_PASTE as this *may* be the same buffer returned by event_text()). The selection_owner() widget is set to the passed owner.

11.5.2.8 selection_owner() [1/2]

```
Fl_Widget * Fl::selection_owner ( )  [inline]
```

back-compatibility only: Gets the widget owning the current selection

See also

Fl_Widget* selection_owner(Fl_Widget*)

11.5.2.9 selection_owner() [2/2]

```
void Fl::selection_owner (
            Fl_Widget * owner )  [extern]
```

Back-compatibility only: The single-argument call can be used to move the selection to another widget or to set the owner to NULL, without changing the actual text of the selection.
FL_SELECTIONCLEAR is sent to the previous selection owner, if any.
Copying the buffer every time the selection is changed is obviously wasteful, especially for large selections. An interface will probably be added in a future version to allow the selection to be made by a callback function. The current interface will be emulated on top of this.

11.5.2.10 selection_to_clipboard() [1/2]

```
int Fl::selection_to_clipboard ( )  [extern]
```

Returns the current selection_to_clipboard mode.

See also

void selection_to_clipboard(int)

11.5.2.11 selection_to_clipboard() [2/2]

```
void Fl::selection_to_clipboard (
            int mode )  [extern]
```

Copies selections on X11 directly to the clipboard if enabled.
This method can be called on all platforms. Other platforms than X11 are not affected by this feature.
If this is switched on (`mode` = 1), Fl::copy() copies all data to the clipboard regardless of its `destination` argument. If the destination is 0 (selection buffer) data is copied to both the selection buffer and the clipboard.
Drag and drop is also affected since drag-and-drop data is copied to the selection buffer.
You can use this to make the experience of data selection and copying more like that on other platforms (Windows, macOS, and even Wayland).
The default operation mode is the standard X11 behavior (disabled).

Note

This feature is experimental and enabling it may have unexpected side effects. It is your own responsibility if you enable it.

Since

1.4.0

Parameters

in	*mode*	1 = enable selection_to_clipboard, 0 = disable selection_to_clipboard

See also

copy(const char *, int, int, const char *)

11.6 Screen functions

Fl global screen functions declared in <FL/Fl.H>.

Functions

- int **Fl::h** ()

  Returns the height in pixels of the main screen work area.
- void Fl::keyboard_screen_scaling (int value)

  Controls the possibility to scale all windows by ctrl/+/-/0/ or cmd/+/-/0/.
- float Fl::normalized_screen_scale (int n)

  Gets the normalized GUI scaling factor of screen number `n`.
- void Fl::normalized_screen_scale (int n, float factor)

  Sets the normalized GUI scaling factor of screen number `n`.
- int Fl::screen_count ()

  Gets the total count of available screens.
- void Fl::screen_dpi (float &h, float &v, int n=0)

  Gets the screen resolution in dots-per-inch for the given screen.
- int Fl::screen_num (int x, int y)

  Gets the screen number of a screen that contains the specified screen position `x`, `y`.
- int Fl::screen_num (int x, int y, int w, int h)

  Gets the screen number of the screen which intersects the most with the rectangle defined by `x`, `y`, `w`, `h`.
- float Fl::screen_scale (int n)

  Gets the GUI scaling factor of screen number `n`.
- void Fl::screen_scale (int n, float factor)

Sets the GUI scaling factor of screen number n.

- int Fl::screen_scaling_supported ()

  Returns whether scaling factors are supported by this platform.
- void Fl::screen_work_area (int &X, int &Y, int &W, int &H)

  Gets the bounding box of the work area of the screen that contains the mouse pointer.
- void Fl::screen_work_area (int &X, int &Y, int &W, int &H, int mx, int my)

  Gets the bounding box of the work area of a screen that contains the specified screen position mx, my.
- void Fl::screen_work_area (int &X, int &Y, int &W, int &H, int n)

  Gets the bounding box of the work area of the given screen.
- void Fl::screen_xywh (int &X, int &Y, int &W, int &H)

  Gets the bounding box of a screen that contains the mouse pointer.
- void Fl::screen_xywh (int &X, int &Y, int &W, int &H, int mx, int my)

  Gets the bounding box of a screen that contains the specified screen position mx, my.
- void Fl::screen_xywh (int &X, int &Y, int &W, int &H, int mx, int my, int mw, int mh)

  Gets the screen bounding rect for the screen which intersects the most with the rectangle defined by mx, my, mw, mh.
- void Fl::screen_xywh (int &X, int &Y, int &W, int &H, int n)

  Gets the screen bounding rect for the given screen.
- int **Fl::w** ()

  Returns the width in pixels of the main screen work area.
- int **Fl::x** ()

  Returns the leftmost x coordinate of the main screen work area.
- int **Fl::y** ()

  Returns the topmost y coordinate of the main screen work area.

11.6.1 Detailed Description

Fl global screen functions declared in <FL/Fl.H>.
FLTK supports high-DPI screens using a screen scaling factor. The scaling factor is initialized by the library to a value based on information obtained from the OS. If this initial value is not satisfactory, the FLTK_SCALING_↩FACTOR environment variable can be set to a value FLTK will multiply to the OS-given value. The 2 variants of functions Fl::screen_scale() allow to programmatically get and set scaling factor values. The scaling factor value can be further changed at runtime by typing `Ctrl/+/-/0/` (`Cmd/+/-/0/` under macOS). See FL_SHORTCUT for more details about these shortcuts.

11.6.2 Function Documentation

11.6.2.1 keyboard_screen_scaling()

```
void Fl::keyboard_screen_scaling (
            int value )  [extern]
```

Controls the possibility to scale all windows by ctrl/+/-/0/ or cmd/+/-/0/.
This function **should** be called before fl_open_display() runs. If it is not called, the default is to handle these keys for window scaling.

Note

This function can currently only be used to switch the internal handler **off**, i.e. `value` must be 0 (zero) - all other values result in undefined behavior and are reserved for future extension.

Parameters

value	0 to stop recognition of ctrl/+/-/0/ (or cmd/+/-/0/ under macOS) keys as window scaling.

Since

1.4.0

11.6.2.2 normalized_screen_scale() [1/2]

```
float Fl::normalized_screen_scale (
            int n )  [extern]
```

Gets the normalized GUI scaling factor of screen number `n`.
Calculate the scale of the windows and their graphics on a screen in relation to the initial scale of the screen. The initial scale is the scale of the screen when the application was started. This is the samle value that is shown in the transient popup windows during interactive app scaling.

Parameters

in	*n*	screen index between 0 and Fl::screen_count() - 1

Returns

screen scaling factor in relation to the initial scale

See also

Fl::screen_scale(int)

Since

1.5.0

11.6.2.3 normalized_screen_scale() [2/2]

```
void Fl::normalized_screen_scale (
            int n,
            float factor )  [extern]
```

Sets the normalized GUI scaling factor of screen number `n`.
Set the scale of the windows and their graphics on a screen in relation to the initial scale of the screen.

Parameters

in	*n*	Screen index between 0 and Fl::screen_count() - 1. Set this to -1 to scale all screens.
in	*factor*	Scaling factor for the given screen.

See also

Fl::screen_scale(int, float)

Since

1.5.0

11.6.2.4 screen_count()

```
int Fl::screen_count ( )  [extern]
```

Gets the total count of available screens.

Note

Screen numbers range from 0 to Fl::screen_count() - 1 in the FLTK API.

11.6.2.5 screen_dpi()

```
void Fl::screen_dpi (
            float & h,
            float & v,
            int n = 0 )  [extern]
```

Gets the screen resolution in dots-per-inch for the given screen.

Parameters

out	*h,v*	horizontal and vertical resolution
in	*n*	the screen number (0 to Fl::screen_count() - 1)

See also

void screen_xywh(int &x, int &y, int &w, int &h, int mx, int my)

11.6.2.6 screen_num() [1/2]

```
int Fl::screen_num (
            int x,
            int y )  [extern]
```

Gets the screen number of a screen that contains the specified screen position `x`, `y`.

Parameters

in	*x,y*	the absolute screen position

Returns

a screen number [0 , Fl::screen_count()-1]

Attention

When the running system contains screens with different scaling factors, this API may become ambiguous because a given value pair (`x`, `y`) may belong to distinct screens. In that situation other APIs should be preferred, e.g. Fl_Window::screen_num() and Fl::screen_scale(int).

11.6.2.7 screen_num() [2/2]

```
int Fl::screen_num (
            int x,
            int y,
            int w,
            int h )  [extern]
```

Gets the screen number of the screen which intersects the most with the rectangle defined by `x`, `y`, `w`, `h`.

Parameters

in	*x,y,w,h*	the rectangle to search for intersection with

Returns

a screen number [0 , Fl::screen_count()-1]

11.6.2.8 screen_scale() [1/2]

```
float Fl::screen_scale (
            int n )  [extern]
```

Gets the GUI scaling factor of screen number `n`.
The valid range of `n` is 0 .. Fl::screen_count() - 1.
The return value is `1.0` if screen scaling is not supported or `n` is outside the valid range.

Returns

Current screen scaling factor (default: `1.0`)

See also

Fl::screen_count()

Fl::screen_scaling_supported()

Since

1.4.0

11.6.2.9 screen_scale() [2/2]

```
void Fl::screen_scale (
            int n,
            float factor )  [extern]
```

Sets the GUI scaling factor of screen number `n`.
The valid range of `n` is 0 .. Fl::screen_count() - 1.
This method does nothing if `n` is out of range or screen scaling is not supported by this platform.
Otherwise it also sets the scaling factor of all windows mapped to screen number `n` or all screens, depending on the type of screen scaling support on the platform.

Parameters

in	*n*	screen number
in	*factor*	scaling factor of screen `n`

See also

Fl::screen_scaling_supported()

Since

1.4.0

11.6.2.10 screen_scaling_supported()

```
int Fl::screen_scaling_supported ( )  [extern]
```

Returns whether scaling factors are supported by this platform.

Return values

0	scaling factors are not supported by this platform
1	a single scaling factor is shared by all screens
2	each screen can have its own scaling factor

See also

Fl::screen_scale(int)

Since

1.4.0

11.6.2.11 screen_work_area() [1/3]

```
void Fl::screen_work_area (
            int & X,
            int & Y,
            int & W,
            int & H )  [extern]
```

Gets the bounding box of the work area of the screen that contains the mouse pointer.

Parameters

out	*X,Y,W,H*	the work area bounding box

See also

void screen_work_area(int &x, int &y, int &w, int &h, int n)

11.6.2.12 screen_work_area() [2/3]

```
void Fl::screen_work_area (
            int & X,
            int & Y,
            int & W,
            int & H,
            int mx,
            int my )  [extern]
```

Gets the bounding box of the work area of a screen that contains the specified screen position `mx`, `my`.

Parameters

out	*X,Y,W,H*	the work area bounding box
in	*mx,my*	the absolute screen position

See also

void screen_work_area(int &x, int &y, int &w, int &h, int n)

11.6.2.13 screen_work_area() [3/3]

```
void Fl::screen_work_area (
            int & X,
            int & Y,
            int & W,
            int & H,
            int n )  [extern]
```

Gets the bounding box of the work area of the given screen.

Parameters

out	*X,Y,W,H*	the work area bounding box
in	*n*	the screen number (0 to Fl::screen_count() - 1)

See also

void screen_xywh(int &x, int &y, int &w, int &h, int mx, int my)

Note

Under X11 and Wayland, the screen work area is given values that differ from that screen's bounding box only if the system contains a single screen.

Like all quantities accessible via public APIs of FLTK, values of `X`,Y,W,H are given in FLTK units, that is, in drawing units divided by the scaling factor of screen `n`.

11.6.2.14 screen_xywh() [1/4]

```
void Fl::screen_xywh (
            int & X,
            int & Y,
            int & W,
            int & H )  [extern]
```

Gets the bounding box of a screen that contains the mouse pointer.

Parameters

out	*X,Y,W,H*	the corresponding screen bounding box

See also

void screen_xywh(int &x, int &y, int &w, int &h, int mx, int my)

11.6.2.15 screen_xywh() [2/4]

```
void Fl::screen_xywh (
            int & X,
            int & Y,
            int & W,
            int & H,
            int mx,
            int my )  [extern]
```

Gets the bounding box of a screen that contains the specified screen position `mx`, `my`.

Parameters

out	*X,Y,W,H*	the corresponding screen bounding box
in	*mx,my*	the absolute screen position

11.6.2.16 screen_xywh() [3/4]

```
void Fl::screen_xywh (
            int & X,
            int & Y,
            int & W,
```

```
            int & H,
            int mx,
            int my,
            int mw,
            int mh )  [extern]
```

Gets the screen bounding rect for the screen which intersects the most with the rectangle defined by `mx`, `my`, `mw`, `mh`.

Parameters

out	*X,Y,W,H*	the corresponding screen bounding box
in	*mx,my,mw,mh*	the rectangle to search for intersection with

See also

void screen_xywh(int &X, int &Y, int &W, int &H, int n)

11.6.2.17 screen_xywh() [4/4]

```
void Fl::screen_xywh (
            int & X,
            int & Y,
            int & W,
            int & H,
            int n )  [extern]
```

Gets the screen bounding rect for the given screen.
Under Windows, Mac OS X, and X11 + the Gnome desktop, screen #0 contains the menubar/taskbar

Parameters

out	*X,Y,W,H*	the corresponding screen bounding box
in	*n*	the screen number (0 to Fl::screen_count() - 1)

Note

Like all quantities accessible via public APIs of FLTK, values of `X`,Y,W,H are given in FLTK units, that is, in drawing units divided by the scaling factor of screen `n`.

See also

void screen_xywh(int &x, int &y, int &w, int &h, int mx, int my)

11.7 Color & Font functions

fl global color, font functions.

Functions

- Fl_Color fl_color ()

  Return the last fl_color() that was set.
- void fl_color (Fl_Color c)

  Set the color for all subsequent drawing operations.
- void **fl_color** (int c)

  for back compatibility - use fl_color(Fl_Color c) instead
- void fl_color (uchar r, uchar g, uchar b)

Set the color for all subsequent drawing operations.

- Fl_Color fl_color_average (Fl_Color color1, Fl_Color color2, float weight)

  Returns the weighted average color between the two given colors.
- Fl_Color fl_contrast (Fl_Color fg, Fl_Color bg, int context, int size)

  Returns a color that contrasts with the background color.
- void fl_contrast_function (Fl_Contrast_Function *f)

  Register a custom contrast function.
- int fl_contrast_level ()

  Get the contrast level (sensitivity) of the fl_contrast() method.
- void fl_contrast_level (int level)

  Set the contrast level (sensitivity) of the fl_contrast() method.
- int fl_contrast_mode ()

  Return the current contrast algorithm (mode).
- void fl_contrast_mode (int mode)

  Set the contrast algorithm (mode).
- int fl_descent ()

  Return the recommended distance above the bottom of a fl_height() tall box to draw the text at so it looks centered vertically in that box.
- Fl_Font fl_font ()

  Return the `face` set by the most recent call to fl_font().
- void fl_font (Fl_Font face, Fl_Fontsize fsize)

  Sets the current font, which is then used in various drawing routines.
- int fl_height ()

  Return the recommended minimum line spacing for the current font.
- int fl_height (int font, int size)

  This function returns the actual height of the specified `font` and `size`.
- Fl_Color **fl_inactive** (Fl_Color c)

  Returns the inactive, dimmed version of the given color.
- const char * fl_latin1_to_local (const char *t, int n=-1)

  Convert text from Windows/X11 latin1 character set to local encoding.
- double fl_lightness (Fl_Color color)

  Return the perceived lightness of a color.
- const char * fl_local_to_latin1 (const char *t, int n=-1)

  Convert text from local encoding to Windows/X11 latin1 character set.
- const char * fl_local_to_mac_roman (const char *t, int n=-1)

  Convert text from local encoding to Mac Roman character set.
- double fl_luminance (Fl_Color color)

  Return the raw / physical luminance of a color.
- const char * fl_mac_roman_to_local (const char *t, int n=-1)

  Convert text from Mac Roman character set to local encoding.
- Fl_Color fl_show_colormap (Fl_Color oldcol)

  Pops up a window to let the user pick a colormap entry.
- Fl_Fontsize fl_size ()

  Return the `size` set by the most recent call to fl_font().
- void fl_text_extents (const char *t, int &dx, int &dy, int &w, int &h)

  Determine the minimum pixel dimensions of a nul-terminated string using the current fl_font().
- void fl_text_extents (const char *t, int n, int &dx, int &dy, int &w, int &h)

  Determine the minimum pixel dimensions of a sequence of `n` bytes of text using the current font face and size.
- double fl_width (const char *txt)

  Return the typographical width of a nul-terminated string using the current font face and size.
- double fl_width (const char *txt, int n)

Return the typographical width of a sequence of `n` bytes of text in UTF-8 encoding, using the current font face and size.

- double fl_width (unsigned int c)

  Return the typographical width of a single character using the current font face and size.
- void Fl::free_color (Fl_Color i, int overlay=0)

  Frees the specified color from the colormap, if applicable.
- unsigned Fl::get_color (Fl_Color i)

  Returns the RGB value(s) for the given FLTK color index.
- void Fl::get_color (Fl_Color i, uchar &red, uchar &green, uchar &blue)

  Returns the RGB value(s) for the given FLTK color index.
- void Fl::get_color (Fl_Color i, uchar &red, uchar &green, uchar &blue, uchar &alpha)

  Returns the RGBA value(s) for the given FLTK color index.
- const char * Fl::get_font (Fl_Font)

  Gets the string for this face.
- const char * Fl::get_font_name (Fl_Font, int *attributes=0)

  Get a human-readable string describing the family of this face.
- int Fl::get_font_sizes (Fl_Font, int *&sizep)

  Return an array of sizes in `sizep`.
- void Fl::set_color (Fl_Color i, unsigned c)

  Sets an entry in the fl_color index table.
- void Fl::set_color (Fl_Color, uchar, uchar, uchar)

  Sets an entry in the fl_color index table.
- void Fl::set_color (Fl_Color, uchar, uchar, uchar, uchar)

  Sets an entry in the fl_color index table.
- void Fl::set_font (Fl_Font, const char *)

  Changes a face.
- void **Fl::set_font** (Fl_Font, Fl_Font)

  Copies one face to another.
- Fl_Font Fl::set_fonts (const char *=0)

  FLTK will open the display, and add every fonts on the server to the face table.

11.7.1 Detailed Description

fl global color, font functions.
These functions are declared in <FL/Fl.H> or <FL/fl_draw.H>.

11.7.2 Function Documentation

11.7.2.1 fl_color() [1/3]

```
Fl_Color fl_color ( ) [inline]
```

Return the last fl_color() that was set.
This can be used for state save/restore.

11.7.2.2 fl_color() [2/3]

```
void fl_color (
            Fl_Color c ) [inline]
```

Set the color for all subsequent drawing operations.
For color-mapped displays, a color cell will be allocated out of `fl_colormap` the first time you use a color. If the colormap fills up then a least-squares algorithm is used to find the closest color. If no valid graphical context (fl_gc) is available, the foreground is not set for the current window.

Parameters

in	*c*	color

11.7.2.3 fl_color() [3/3]

```
void fl_color (
            uchar r,
            uchar g,
            uchar b )  [inline]
```

Set the color for all subsequent drawing operations.
The closest possible match to the RGB color is used. The RGB color is used directly on TrueColor displays. For colormap visuals the nearest index in the gray ramp or color cube is used. If no valid graphical context (fl_gc) is available, the foreground is not set for the current window.

Parameters

in	*r,g,b*	color components

11.7.2.4 fl_color_average()

```
Fl_Color fl_color_average (
            Fl_Color color1,
            Fl_Color color2,
            float weight )
```

Returns the weighted average color between the two given colors.
The red, green and blue values are averages using the following formula:
`color = color1 * weight  + color2 * (1 - weight)`
Thus, a `weight` value of 1.0 will return the first color, while a value of 0.0 will return the second color.

Parameters

in	*color1,color2*	boundary colors
in	*weight*	weighting factor

11.7.2.5 fl_contrast()

```
Fl_Color fl_contrast (
            Fl_Color fg,
            Fl_Color bg,
            int context,
            int size )
```

Returns a color that contrasts with the background color.
This will be the foreground color if it contrasts sufficiently with the background color. Otherwise, returns `FL_WHITE` or `FL_BLACK` depending on which color provides the best contrast.
FLTK 1.4.0 uses a different default contrast function than earlier releases (1.3.x) but you can use the old "legacy" contrast function by calling
`fl_contrast_mode(FL_CONTRAST_LEGACY);`
early in your main program.

Note

> It is a known issue that static initialization using fl_contrast() may already have been executed before you call this function in main(). You should be aware of this and, if necessary, write your own (static) contrast initialization function. This should rarely be necessary.

You can change the behavior of fl_contrast() in several ways:

- Change the "level" (sensitivity) for contrast calculation, see fl_contrast_level(). Valid levels are 0 - 100, the default "medium" value depends on the contrast mode. If you raise the level above the default value the overall contrast will generally be higher, i.e. the required contrast to return the foreground color is raised and therefore the calculated color switches "earlier" to either black or white. In other words, using the following values:
  - 0 always uses the foreground color
  - the default, unmodified algorithm allows a sufficient contrast such that the text is readable
  - 100 will always use black or white

  Changing the `level` is particularly useful and intended for the "legacy mode" to improve the results partially. Values slightly above 50 (50 - 70) will likely return the best results (50 is the default, as used in FLTK 1.3.x).

  Note

  Different contrast modes (algorithms) can use their own values and defaults of fl_contrast_level().

- Change the used contrast calculation function. You can either use the old (FLTK 1.3.x) function or use the better but slower function based on the CIELAB (L∗a∗b∗) color model, or you can define your own custom contrast function if you need even better contrast results.

The following contrast functions are available:

- FL_CONTRAST_LEGACY, the old FLTK 1.3.x compatible function. This is the fastest function (using integer arithmetic) but it provides worse results in border cases. You may want to use this on embedded or otherwise CPU constrained systems or if you need strict backwards compatibility. For slightly better results you may utilize the new fl_contrast_level(int) function (since 1.4.0) to increase the contrast sensitivity. This will provide slightly better results than FLTK 1.3.x and earlier but we recommend to use the new default function:
- FL_CONTRAST_CIELAB, based on the CIELAB (L∗a∗b∗) color model. This function is superior regarding the human contrast perception but may be slightly slower - which should not matter on a modern CPU. The default contrast level in this mode is 39 which results in a very similar experience as the old contrast function but avoids unreadable border cases. **This is the default since FLTK 1.4.0.**
- FL_CONTRAST_CUSTOM, your own contrast calculation function.

In the future we **may** provide even more (and superior) contrast algorithms.
The new parameters `context` and `size` (since 1.4.0) are defined for future extensions and are currently not used. Default values are 0.

- The `context` is intended to differentiate text and other kinds of objects, e.g. radio buttons, check marks, or icon types.
- The `size` parameter is an unspecified (object) size that may be used to calculate the required contrast. In text mode this must be the font size. Rule: the larger the object (font), the lower the required contrast.

Note

These new optional parameters must be provided in the custom contrast function which is the reason why they are added now. In the future we may use the (font) size to adjust the calculated contrast, and users defining their own contrast functions may use them in their functions.

Parameters

in	*fg*	foreground (text) color
in	*bg*	background color
in	*context*	graphical context (optional, default = 0 == text)
in	*size*	unspecified size (optional, default = 0 == undefined)

Returns

contrasting color: `fg`, `FL_BLACK`, or `FL_WHITE`

See also

fl_contrast_level(int)

fl_contrast_mode(int)

fl_contrast_function()

11.7.2.6 fl_contrast_function()

```
void fl_contrast_function (
            Fl_Contrast_Function * f )
```

Register a custom contrast function.
Your custom contrast function will be called when fl_contrast() is called if and only if you previously registered your function and called fl_contrast_mode(FL_CONTRAST_CUSTOM) .
Your custom contrast function must provide the signature
`Fl_Color my_contrast_function(Fl_Color fg, Fl_Color bg, int context, int size)`
The arguments are the same as for the full fl_contrast() function since FLTK 1.4. You can use the supplied `size` to modify the result. Depending on the caller the `size` parameter can be 0 (default) or a valid size. In the context of text, i.e. `context` == 0, the `size` parameter is the fontsize.
The `context` parameter is not yet used and will always be 0 unless included in a call to fl_contrast(). The value 0 must be interpreted as text. In the future the `context` argument will be used to supply a different context than text (small icons, large icons, etc.). The exact usage is not yet specified.
Your function may also use fl_contrast_level() to modify the result accordingly.

Since

1.4.0

See also

fl_contrast_mode(int)

fl_contrast_level(int)

fl_contrast()

11.7.2.7 fl_contrast_level() [1/2]

```
int fl_contrast_level ( )
```

Get the contrast level (sensitivity) of the fl_contrast() method.
This returns the level of the currently selected contrast mode.

Returns

The current contrast level.

See also

fl_contrast_level(int level)

fl_contrast_mode(int mode)

Since

1.4.0

11.7.2.8 fl_contrast_level() [2/2]

```
void fl_contrast_level (
            int level )
```

Set the contrast level (sensitivity) of the fl_contrast() method.
This can be used to tune the legacy fl_contrast() function to achieve slightly better results. The default value is defined per contrast mode (see below). Values between 50 and 70 may be useful for the legacy contrast mode but you can raise it up to 100. Lower values than 50 are probably not useful.
The contrast `level` affects not only the legacy (1.3.x) fl_contrast() function but also the new CIELAB contrast mode which is the default since FLTK 1.4.0. See default value below.
Other contrast modes are currently not affected by the contrast level.
You may use the contrast level if you define your own custom contrast function in mode FL_CONTRAST_CUSTOM.

Note

All contrast modes store their own contrast level because the behavior is slightly different. You must change the contrast mode fl_contrast_mode() **before** you set or get the contrast level.

The default contrast level is

- 50 in mode FL_CONTRAST_LEGACY (compatible with FLTK 1.3.x)
- 39 in mode FL_CONTRAST_CIELAB (similar threshold as in FLTK 1.3.x)
- 0 (undefined) for all other modes

See the description of fl_contrast_mode(int mode) for more information about the contrast level per mode.
Example:

```
fl_contrast_mode(FL_CONTRAST_LEGACY);
fl_contrast_level(60);
```

A `level` greater than 50 (probably best in the range 50 to 70) may achieve better results of the legacy fl_contrast() function in some border cases of low contrast between foreground and background colors but we recommend to use the new default algorithm `FL_CONTRAST_CIELAB` unless you need strict backwards compatibility or use a CPU constrained embedded system.

Parameters

in	*level*	valid range is 0 to 100

Since

1.4.0

11.7.2.9 fl_contrast_mode() [1/2]

```
int fl_contrast_mode ( )
```

Return the current contrast algorithm (mode).

Returns

Contrast algorithm (mode).

Since

1.4.0

See also

fl_contrast_mode(int)

11.7.2.10 fl_contrast_mode() [2/2]

```
void fl_contrast_mode (
            int mode )
```

Set the contrast algorithm (mode).
You can use one of

- FL_CONTRAST_NONE (not recommended: returns the foreground color)
- FL_CONTRAST_LEGACY (same as in FLTK 1.3.x)
- FL_CONTRAST_CIELAB (better, this is the default since FLTK 1.4.0)
- FL_CONTRAST_CUSTOM (you must define your own contrast algorithm)

If you set FL_CONTRAST_CUSTOM you must also register your custom contrast function by calling fl_contrast_function().
You may set the contrast level fl_contrast_level(int) after setting the contrast mode. This affects the contrast algorithm as described below:

- FL_CONTRAST_LEGACY: default level is 50 which is compatible with FLTK 1.3.x and older. This mode is no longer the default and is not recommended because it doesn't take human contrast perception into account and doesn't properly handle sRGB color values. You may get better contrasts if you set the level higher than 50. Values in the range 50 to 70 may be useful. Higher values result in higher contrast, i.e. the algorithm switches "earlier" to black or white mode.
- FL_CONTRAST_CIELAB: defaut level is 39 which appears to be a good value. The higher the level is, the more contrast is to be expected. Values in the range below 39 accept lower contrast and values above 39 switch "earlier" to black or white. Values between 36 and 46 may yield usable contrast experience.

Note

The goal of fl_contrast() is to achieve a "sufficient" contrast between text and background. Level 39 in CIELAB mode means that the accepted contrast is about 39% of the lightness difference between both colors. This can be perceived as very low contrast in some cases, but the text should at least be readable. Note that the highest possible contrast value on a medium gray background is 50% (either black or white). Bill Spitzak wrote on May 16, 2024 in fltk.general in thread "FLTK 1.4 Menu Bar Style": *"I would certainly aim for a function that does not alter color combinations where it is physically possible to read the text, even if squinting is needed."* See https://groups.google.com/g/fltkgeneral/c/EkWI4HTHSLA/m/rsZunZ1vAwAJ

Parameters

in	*mode*	if invalid, FL_CONTRAST_CIELAB will be selected

Since

1.4.0

See also

fl_contrast_function(Fl_Contrast_Function *)

fl_contrast_level(int)

11.7.2.11 fl_descent()

```
int fl_descent ( ) [inline]
```

Return the recommended distance above the bottom of a fl_height() tall box to draw the text at so it looks centered vertically in that box.

See also

fl_font(Fl_Font face, Fl_Fontsize fsize) to load the current font before this method is called.

11.7.2.12 fl_font() [1/2]

```
Fl_Font fl_font ( ) [inline]
```

Return the `face` set by the most recent call to fl_font().
This can be used to save/restore the font.

11.7.2.13 fl_font() [2/2]

```
void fl_font (
            Fl_Font face,
            Fl_Fontsize fsize )
```

Sets the current font, which is then used in various drawing routines.
You may call this outside a draw context if necessary to measure text, for instance by calling fl_width(), fl_measure(), or fl_text_extents(), but on X this will open the display.
The font is identified by a `face` and a `size`. The size of the font is measured in pixels and not "points". Lines should be spaced `size` pixels apart or more.

11.7.2.14 fl_height() [1/2]

```
int fl_height ( ) [inline]
```

Return the recommended minimum line spacing for the current font.
You can also use the value of `size` passed to fl_font().

See also

fl_font(Fl_Font face, Fl_Fontsize fsize) to load the current font before this method is called.

11.7.2.15 fl_height() [2/2]

```
int fl_height (
            int font,
            int size )
```

This function returns the actual height of the specified `font` and `size`.
Normally the font height should always be 'size', but with the advent of XFT, there are (currently) complexities that seem to only be solved by asking the font what its actual font height is. (See STR#2115)
This function was originally undocumented in 1.1.x, and was used only by Fl_Text_Display. We're now documenting it in 1.3.x so that apps that need precise height info can get it with this function.

Returns

the height of the font in pixels.

Todo In the future, when the XFT issues are resolved, this function should simply return the 'size' value.

11.7.2.16 fl_latin1_to_local()

```
const char * fl_latin1_to_local (
            const char * t,
            int n = -1 )
```

Convert text from Windows/X11 latin1 character set to local encoding.

Parameters

in	*t*	character string (latin1 encoding)
in	*n*	optional number of characters (bytes) to convert (default is all)

Returns

pointer to internal buffer containing converted characters

11.7.2.17 fl_lightness()

```
double fl_lightness (
            Fl_Color color )
```

Return the perceived lightness of a color.
This function calculates the perceived lightness of Fl_Color `color`.
The returned lightness value `Lstar` according to the CIELAB (L∗a∗b∗) color model is almost linear with respect to human perception. It is in the range 0 (black) to 100 (white).
The result values of two colors can be compared directly and the difference is their perceived contrast.

Parameters

in	*color*	Fl_Color value

Returns

perceived lightness (0 .. 100)

Since

1.4.0

11.7.2.18 fl_local_to_latin1()

```
const char * fl_local_to_latin1 (
            const char * t,
            int n = -1 )
```

Convert text from local encoding to Windows/X11 latin1 character set.

Parameters

in	*t*	character string (local encoding)
in	*n*	optional number of characters (bytes) to convert (default is all)

Returns

pointer to internal buffer containing converted characters

11.7.2.19 fl_local_to_mac_roman()

```
const char * fl_local_to_mac_roman (
            const char * t,
            int n = -1 )
```

Convert text from local encoding to Mac Roman character set.

Parameters

in	*t*	character string (local encoding)
in	*n*	optional number of characters to convert (default is all)

Returns

pointer to internal buffer containing converted characters

11.7.2.20 fl_luminance()

```
double fl_luminance (
            Fl_Color color )
```

Return the raw / physical luminance of a color.
This function calculates the physical luminance of Fl_Color `color`.
The returned luminance value (aka `Y`) is the physical luminance of the Fl_Color `color`.
The result is in the range 0.0 (black) to 1.0 (white).

Note

This is probably not what you want if you are interested in perceived contrast or lightness calculation because the luminance `Y` is **not** linear with respect to human perception.

See fl_lightness(Fl_Color) for a function that returns the perceived lightness of a color which can be used directly for contrast calculation.

Parameters

in	*color*	Fl_Color value

Returns

Raw (physical) luminance (0.0 .. 1.0)

Since

1.4.0

See also

fl_lightness(Fl_Color)

11.7.2.21 fl_mac_roman_to_local()

```
const char * fl_mac_roman_to_local (
            const char * t,
            int n = -1 )
```

Convert text from Mac Roman character set to local encoding.

Parameters

in	*t*	character string (Mac Roman encoding)
in	*n*	optional number of characters to convert (default is all)

Returns

pointer to internal buffer containing converted characters

11.7.2.22 fl_show_colormap()

```
Fl_Color fl_show_colormap (
            Fl_Color oldcol )
```

Pops up a window to let the user pick a colormap entry.

Figure 11.1 fl_show_colormap

Parameters

in	*oldcol*	color to be highlighted when grid is shown.

Return values

Fl_Color	value of the chosen colormap entry.

See also

Fl_Color_Chooser

11.7.2.23 fl_size()

```
Fl_Fontsize fl_size ( ) [inline]
```

Return the `size` set by the most recent call to fl_font().
This can be used to save/restore the font.

11.7.2.24 fl_text_extents() [1/2]

```
void fl_text_extents (
            const char * t,
            int & dx,
            int & dy,
            int & w,
            int & h )
```

Determine the minimum pixel dimensions of a nul-terminated string using the current fl_font().
Usage: given a string "txt" drawn using fl_draw(txt, x, y) you would determine its pixel extents on the display using fl_text_extents(txt, dx, dy, wo, ho) such that a bounding box that exactly fits around the text could be drawn with

fl_rect(x+dx, y+dy, wo, ho). Note the dx, dy values hold the offset of the first "colored in" pixel of the string, from the draw origin. dx and dy can be negative.
Note the desired font and font size must be set with fl_font() before calling this function.
This differs slightly from fl_measure() in that the dx/dy values are also returned.
No FLTK symbol expansion will be performed.
Example use:

```
int dx,dy,W,H;
fl_font(FL_HELVETICA, 12);                    // set font face and size first
fl_text_extents("Some text", dx, dy, W, H);   // get width and height of string
printf("text's width=%d, height=%d\n", W, H);
```

Parameters

in	*t*	text to be measured
out	*dx,dy*	offsets from x,y position of text (see description)
out	*w,h*	width and height of bounding box

11.7.2.25 fl_text_extents() [2/2]

```
void fl_text_extents (
            const char * t,
            int n,
            int & dx,
            int & dy,
            int & w,
            int & h )  [inline]
```

Determine the minimum pixel dimensions of a sequence of `n` bytes of text using the current font face and size.
This is the same as fl_text_extents(const char* t, int& dx, int& dy, int& w, int& h) except that the string doesn't need to be nul-terminated. The length in bytes is given in the second parameter `n`.

Parameters

in	*t*	text to be measured
in	*n*	length of text in bytes
out	*dx,dy*	offsets from x,y position of text (see description)
out	*w,h*	width and height of bounding box

See also

fl_text_extents(const char* t, int& dx, int& dy, int& w, int& h)

11.7.2.26 fl_width() [1/3]

```
double fl_width (
            const char * txt )
```

Return the typographical width of a nul-terminated string using the current font face and size.

See also

fl_font(Fl_Font face, Fl_Fontsize fsize) to load the current font before this method is called.

11.7.2.27 fl_width() [2/3]

```
double fl_width (
            const char * txt,
            int n )  [inline]
```

Return the typographical width of a sequence of `n` bytes of text in UTF-8 encoding, using the current font face and size.

n must be chosen such that only complete UTF-8 characters are included in the given text txt. Otherwise the result is undefined.

Parameters

in	*txt*	The UTF-8 encoded text to be measured.
in	*n*	Length of the string in bytes.

See also

fl_font(Fl_Font face, Fl_Fontsize fsize) to load the current font before this method is called.

11.7.2.28 fl_width() [3/3]

```
double fl_width (
            unsigned int c ) [inline]
```

Return the typographical width of a single character using the current font face and size.

Parameters

in	*c*	The Unicode code point of the character to be measured.

See also

fl_font(Fl_Font face, Fl_Fontsize fsize) to load the current font before this method is called.

Example:

```
auto width = fl_width(0x20ac);  // measures the width of the `€' sign.
```

11.7.2.29 free_color()

```
void Fl::free_color (
            Fl_Color i,
            int overlay = 0 ) [extern]
```

Frees the specified color from the colormap, if applicable.
If overlay is non-zero then the color is freed from the overlay colormap.

11.7.2.30 get_color() [1/3]

```
unsigned Fl::get_color (
            Fl_Color i ) [extern]
```

Returns the RGB value(s) for the given FLTK color index.
This form returns the RGB values packed in a 32-bit unsigned integer with the red value in the upper 8 bits, the green value in the next 8 bits, and the blue value in bits 8-15. The lower 8 bits will always be 0.

11.7.2.31 get_color() [2/3]

```
void Fl::get_color (
            Fl_Color i,
            uchar & red,
            uchar & green,
            uchar & blue ) [extern]
```

Returns the RGB value(s) for the given FLTK color index.
This form returns the red, green, and blue values separately in referenced variables.

See also

unsigned get_color(Fl_Color c)

11.7.2.32 get_color() [3/3]

```
void Fl::get_color (
            Fl_Color i,
            uchar & red,
            uchar & green,
            uchar & blue,
            uchar & alpha )  [extern]
```

Returns the RGBA value(s) for the given FLTK color index.
This form returns the red, green, blue, and alpha values separately in referenced variables.

See also

unsigned get_color(Fl_Color c)

11.7.2.33 get_font()

```
const char * Fl::get_font (
            Fl_Font fnum )  [extern]
```

Gets the string for this face.
This string is different for each face. Under X this value is passed to XListFonts to get all the sizes of this face.

11.7.2.34 get_font_name()

```
const char * Fl::get_font_name (
            Fl_Font fnum,
            int * attributes = 0 )  [extern]
```

Get a human-readable string describing the family of this face.
This is useful if you are presenting a choice to the user. There is no guarantee that each face has a different name. The return value points to a static buffer that is overwritten each call.
The integer pointed to by `attributes` (if the pointer is not zero) is set to zero, FL_BOLD or FL_ITALIC or FL↩_BOLD | FL_ITALIC. To locate a "family" of fonts, search forward and back for a set with non-zero attributes, these faces along with the face with a zero attribute before them constitute a family.

11.7.2.35 get_font_sizes()

```
int Fl::get_font_sizes (
            Fl_Font fnum,
            int *& sizep )  [extern]
```

Return an array of sizes in `sizep`.
The return value is the length of this array. The sizes are sorted from smallest to largest and indicate what sizes can be given to fl_font() that will be matched exactly (fl_font() will pick the closest size for other sizes). A zero in the first location of the array indicates a scalable font, where any size works, although the array may list sizes that work "better" than others. Warning: the returned array points at a static buffer that is overwritten each call. Under X this will open the display.

11.7.2.36 set_color() [1/3]

```
void Fl::set_color (
            Fl_Color i,
            unsigned c )  [extern]
```

Sets an entry in the fl_color index table.
You can set it to any 8-bit RGB color. The color is not allocated until fl_color(i) is used.

11.7.2.37 set_color() [2/3]

```
void Fl::set_color (
            Fl_Color i,
            uchar red,
```

```
            uchar green,
            uchar blue )  [extern]
```

Sets an entry in the fl_color index table.
You can set it to any 8-bit RGB color. The color is not allocated until fl_color(i) is used.

11.7.2.38 set_color() [3/3]

```
void Fl::set_color (
            Fl_Color i,
            uchar red,
            uchar green,
            uchar blue,
            uchar alpha )  [extern]
```

Sets an entry in the fl_color index table.
You can set it to any 8-bit RGBA color.

Note

The color transparency is effective under the Wayland, hybrid Wayland/X11 and macOS platforms, whereas it has no effect under the X11 and Windows platforms. It's also effective for widgets added to an Fl_Gl_Window.

Version

1.4

11.7.2.39 set_font()

```
void Fl::set_font (
            Fl_Font fnum,
            const char * name )  [extern]
```

Changes a face.

Parameters

fnum	The font number to be assigned a new face
name	Name of the font to assign. The string pointer is simply stored, the string is not copied, so the string must be in static memory. The exact name to be used depends on the platform :

- Windows, X11, Xft: use the family name prefixed by one character to indicate the desired font variant. Characters `' '`, `'I'`, `'B'`, `'P'` denote plain, italic, bold, and bold-italic variants, respectively. For example, string `"IGabriola"` is to be used to denote the `"Gabriola italic"` font. The `"Oblique"` suffix, in whatever case, is to be treated as `"italic"`, that is, prefix the family name with `'I'`.
- Other platforms, i.e., X11 + Pango, Wayland, macOS: use the full font name as returned by function Fl::get_font_name() or as listed by applications test/fonts or test/utf8. No prefix is to be added.

11.7.2.40 set_fonts()

```
Fl_Font Fl::set_fonts (
            const char * xstarname = 0 )  [extern]
```

FLTK will open the display, and add every fonts on the server to the face table.
It will attempt to put "families" of faces together, so that the normal one is first, followed by bold, italic, and bold italic.
The only argument to this function is somewhat obsolete since FLTK and most underlying operating systems move to support Unicode. For completeness, following is the original documentation and a few updates:
On X11, the optional argument is a string to describe the set of fonts to add. Passing NULL will select only fonts that have the ISO8859-1 character set (and are thus usable by normal text). Passing "-*" will select all fonts with any encoding as long as they have normal X font names with dashes in them. Passing "*" will list every font that exists (on X this may produce some strange output). Other values may be useful but are system dependent.

With the Xft option on Linux, this parameter is ignored.
With Windows, `NULL` selects fonts with ANSI_CHARSET encoding and non-NULL selects all fonts.
On macOS, this parameter is ignored.
The return value is how many faces are in the table after this is done.

11.8 Drawing functions

FLTK global graphics and GUI drawing functions.

Enumerations

- enum {
  FL_SOLID = 0 , FL_DASH = 1 , FL_DOT = 2 , FL_DASHDOT = 3 ,
  FL_DASHDOTDOT = 4 , FL_CAP_FLAT = 0x100 , FL_CAP_ROUND = 0x200 , FL_CAP_SQUARE = 0x300 ,
  FL_JOIN_MITER = 0x1000 , FL_JOIN_ROUND = 0x2000 , FL_JOIN_BEVEL = 0x3000 , FL_UNIFORM_WIDTH = 0x10000 }

Functions

- int fl_add_symbol (const char ∗name, void(∗draw_in_rect)(int x, int y, int w, int h, Fl_Color c), int scalable)

  Registers (or replaces) a named symbol drawn into an explicit pixel rectangle.
- int fl_add_symbol (const char ∗name, void(∗drawit)(Fl_Color c), int scalable)

  Registers (or replaces) a named symbol drawn using complex vector drawing.
- int **fl_antialias** ()

  Return whether line drawings are currently antialiased.
- void fl_antialias (int state)

  Turn antialiased line drawings ON or OFF, if supported by platform.
- void fl_arc (double x, double y, double r, double start, double end)

  Add a series of points to the current path on the arc of a circle.
- void fl_arc (int x, int y, int w, int h, double a1, double a2)

  Draw ellipse sections using integer coordinates.
- void fl_begin_complex_polygon ()

  Start drawing a complex filled polygon.
- void **fl_begin_line** ()

  Start drawing a list of lines.
- void **fl_begin_loop** ()

  Start drawing a closed sequence of lines.
- void fl_begin_offscreen (Fl_Offscreen ctx)

  Send all subsequent drawing commands to this offscreen buffer.
- void fl_begin_points ()

  Start drawing a list of points.
- void **fl_begin_polygon** ()

  Start drawing a convex filled polygon.
- char fl_can_do_alpha_blending ()

  Check whether platform supports true alpha blending for RGBA images.
- Fl_RGB_Image ∗ fl_capture_window (Fl_Window ∗win, int x, int y, int w, int h)

  Captures the content of a rectangular zone of a mapped window.
- void **fl_chord** (int x, int y, int w, int h, double a1, double a2)

  fl_chord declaration is a place holder - the function does not yet exist
- void fl_circle (double x, double y, double r)

  fl_circle(x,y,r) is equivalent to fl_arc(x,y,r,0,360), but may be faster.
- int fl_clip_box (int x, int y, int w, int h, int &X, int &Y, int &W, int &H)

Intersect a rectangle with the current clip region and return the bounding box of the result.

- Fl_Region fl_clip_region ()

  Return the current clipping region.
- void fl_clip_region (Fl_Region r)

  Replace the top of the clipping stack with a clipping region of any shape.
- void fl_copy_offscreen (int x, int y, int w, int h, Fl_Offscreen pixmap, int srcx, int srcy)

  Copy a rectangular area of the given offscreen buffer into the current drawing destination.
- Fl_Offscreen fl_create_offscreen (int w, int h)

  Creation of an offscreen graphics buffer.
- void fl_cursor (Fl_Cursor)

  Sets the cursor for the current window to the specified shape and colors.
- void **fl_cursor** (Fl_Cursor, Fl_Color fg, Fl_Color bg=FL_WHITE)
- void fl_curve (double X0, double Y0, double X1, double Y1, double X2, double Y2, double X3, double Y3)

  Add a series of points on a Bézier curve to the path.
- void fl_delete_offscreen (Fl_Offscreen ctx)

  Deletion of an offscreen graphics buffer.
- void **fl_draw** (const char ∗str, int n, int x, int y)

  Draws starting at the given `x`, `y` location a UTF-8 string of length `n` bytes.
- void fl_draw (const char ∗str, int x, int y)

  Draw a nul-terminated UTF-8 string starting at the given `x`, `y` location.
- void fl_draw (const char ∗str, int x, int y, int w, int h, Fl_Align align, Fl_Image ∗img=0, int draw_symbols=1, int spacing=0)

  Fancy string drawing function which is used to draw all the labels.
- void fl_draw (const char ∗str, int x, int y, int w, int h, Fl_Align align, void(∗callthis)(const char ∗, int, int, int), Fl_Image ∗img=0, int draw_symbols=1, int spacing=0)

  The same as fl_draw(const char∗,int,int,int,int,Fl_Align,Fl_Image∗,int) with the addition of the `callthis` parameter, which is a pointer to a text drawing function such as fl_draw(const char∗, int, int, int) to do the real work.
- void fl_draw (int angle, const char ∗str, int n, int x, int y)

  Draw at the given `x`, `y` location a UTF-8 string of length `n` bytes rotating `angle` degrees counter-clockwise.
- void fl_draw (int angle, const char ∗str, int x, int y)

  Draw a nul-terminated UTF-8 string starting at the given `x`, `y` location and rotating `angle` degrees counter-clockwise.
- void fl_draw_arrow (Fl_Rect bb, Fl_Arrow_Type t, Fl_Orientation o, Fl_Color color)

  Draw an "arrow like" GUI element for the selected scheme.
- void fl_draw_box (Fl_Boxtype, int x, int y, int w, int h, Fl_Color)

  Draws a box using given type, position, size and color.
- void fl_draw_box_focus (Fl_Boxtype, int x, int y, int w, int h, Fl_Color, Fl_Color)

  Draws the focus rectangle inside a box using given type, position, size and color.
- void fl_draw_check (Fl_Rect bb, Fl_Color col)

  Draw a check mark inside the given bounding box.
- void fl_draw_circle (int x, int y, int d, Fl_Color color)

  Draw a potentially small, filled circle using a given color.
- void fl_draw_image (const uchar ∗buf, int X, int Y, int W, int H, int D=3, int L=0)

  Draw an 8-bit per color RGB or luminance image.
- void fl_draw_image (Fl_Draw_Image_Cb cb, void ∗data, int X, int Y, int W, int H, int D=3)

  Draw an image using a callback function to generate image data.
- void fl_draw_image_mono (const uchar ∗buf, int X, int Y, int W, int H, int D=1, int L=0)

  Draw a gray-scale (1 channel) image.
- void fl_draw_image_mono (Fl_Draw_Image_Cb cb, void ∗data, int X, int Y, int W, int H, int D=1)

  Draw a gray-scale image using a callback function to generate image data.
- int fl_draw_pixmap (char ∗const ∗data, int x, int y, Fl_Color bg=FL_GRAY)

  Draw XPM image data, with the top-left corner at the given position.

- int fl_draw_pixmap (const char ∗const ∗cdata, int x, int y, Fl_Color bg=FL_GRAY)

  Draw XPM image data, with the top-left corner at the given position.
- void fl_draw_radio (int x, int y, int d, Fl_Color color)

  Draw a round check mark (circle) of a radio button.
- int fl_draw_symbol (const char ∗label, int x, int y, int w, int h, Fl_Color)

  Draw the named symbol in the given rectangle using the given color.
- void **fl_end_complex_polygon** ()

  End complex filled polygon, and draw.
- void **fl_end_line** ()

  End list of lines, and draw.
- void **fl_end_loop** ()

  End closed sequence of lines, and draw.
- void **fl_end_offscreen** ()

  Quit sending drawing commands to the current offscreen buffer.
- void **fl_end_points** ()

  End list of points, and draw.
- void **fl_end_polygon** ()

  End convex filled polygon, and draw.
- const char ∗ fl_expand_text (const char ∗from, char ∗buf, int maxbuf, double maxw, int &n, double &width, int wrap, int draw_symbols=0)

  Copy `from` to `buf`, replacing control characters with ^X.
- void fl_focus_rect (int x, int y, int w, int h)

  Draw a dotted rectangle, used to indicate keyboard focus on a widget.
- void fl_frame (const char ∗s, int x, int y, int w, int h)

  Draws a series of line segments around the given box.
- void fl_frame2 (const char ∗s, int x, int y, int w, int h)

  Draws a series of line segments around the given box.
- void fl_gap ()

  Separate loops of the path.
- void **fl_line** (int x, int y, int x1, int y1)

  Draw a line from (x,y) to (x1,y1)
- void **fl_line** (int x, int y, int x1, int y1, int x2, int y2)

  Draw a line from (x,y) to (x1,y1) and another from (x1,y1) to (x2,y2)
- void fl_line_style (int style, int width=0, char ∗dashes=0)

  Set how to draw lines (the "pen").
- void **fl_load_identity** ()

  Set the transformation matrix to identity.
- void fl_load_matrix (double a, double b, double c, double d, double x, double y)

  Set the current transformation matrix.
- void **fl_loop** (int x, int y, int x1, int y1, int x2, int y2)

  Outline a 3-sided polygon with lines.
- void **fl_loop** (int x, int y, int x1, int y1, int x2, int y2, int x3, int y3)

  Outline a 4-sided polygon with lines.
- void fl_measure (const char ∗str, int &x, int &y, int draw_symbols=1)

  Measure how wide and tall the string will be when printed by the fl_draw() function with `align` parameter.
- int fl_measure_pixmap (char ∗const ∗data, int &w, int &h)

  Get the dimensions of a pixmap.
- int fl_measure_pixmap (const char ∗const ∗cdata, int &w, int &h)

  Get the dimensions of a pixmap.
- void fl_mult_matrix (double a, double b, double c, double d, double x, double y)

  Concatenate another transformation onto the current one.

- int fl_not_clipped (int x, int y, int w, int h)

  Does the rectangle intersect the current clip region?
- unsigned int fl_old_shortcut (const char *s)

  Emulation of XForms named shortcuts.
- void fl_overlay_clear ()

  Erase a selection rectangle without drawing a new one.
- void fl_overlay_rect (int x, int y, int w, int h)

  Draw a transient dotted selection rectangle.
- float fl_override_scale ()

  Removes any GUI scaling factor in subsequent drawing operations.
- void fl_pie (int x, int y, int w, int h, double a1, double a2)

  Draw filled ellipse sections using integer coordinates.
- void **fl_point** (int x, int y)

  Draw a single pixel at the given coordinates.
- void **fl_polygon** (int x, int y, int x1, int y1, int x2, int y2)

  Fill a 3-sided polygon.
- void fl_polygon (int x, int y, int x1, int y1, int x2, int y2, int x3, int y3)

  Fill a 4-sided polygon.
- void fl_pop_clip ()

  Restore the previous clip region.
- void **fl_pop_matrix** ()

  Restore the current transformation matrix from the stack.
- void fl_push_clip (int x, int y, int w, int h)

  Intersect the current clip region with a rectangle and push this new region onto the stack.
- void fl_push_matrix ()

  Save the current transformation matrix on the stack.
- void **fl_push_no_clip** ()

  Push an empty clip region onto the stack so nothing will be clipped.
- uchar * fl_read_image (uchar *p, int X, int Y, int W, int H, int alpha=0)

  Reads an RGB(A) image from the current window or off-screen buffer.
- void fl_rect (Fl_Rect r)

  Draw a border inside the given bounding box.
- void fl_rect (int x, int y, int w, int h)

  Draw a border inside the given bounding box.
- void fl_rect (int x, int y, int w, int h, Fl_Color c)

  Draw with passed color a border inside the given bounding box.
- void fl_rectf (Fl_Rect bb, uchar r, uchar g, uchar b)

  Color a rectangle with "exactly" the passed `r,g,b` color.
- void **fl_rectf** (Fl_Rect r)

  Color with current color a rectangle that exactly fills the given bounding box.
- void fl_rectf (Fl_Rect r, Fl_Color c)

  Color with passed color a rectangle that exactly fills the given bounding box.
- void **fl_rectf** (int x, int y, int w, int h)

  Color with current color a rectangle that exactly fills the given bounding box.
- void fl_rectf (int x, int y, int w, int h, Fl_Color c)

  Color with passed color a rectangle that exactly fills the given bounding box.
- void fl_rectf (int x, int y, int w, int h, uchar r, uchar g, uchar b)

  Color a rectangle with "exactly" the passed `r,g,b` color.
- int fl_remove_symbol (const char *name)

  Removes the symbol with name `name` from the system.
- void fl_rescale_offscreen (Fl_Offscreen &ctx)

Adapts an offscreen buffer to a changed value of the scale factor.

- void fl_reset_spot (void)

  Resets marked text.
- void **fl_restore_clip** ()

  Undo any clobbering of the clip region done by your program.
- void fl_restore_scale (float s)

  Restores the GUI scaling factor in subsequent drawing operations.
- void fl_rotate (double d)

  Concatenate rotation transformation onto the current one.
- void fl_rounded_rect (int x, int y, int w, int h, int r)

  Draw a rounded border inside the given bounding box.
- void fl_rounded_rectf (int x, int y, int w, int h, int r)

  Color with current color a rounded rectangle that exactly fills the given bounding box.
- void **fl_rtl_draw** (const char ∗str, int n, int x, int y)

  Draw a UTF-8 string of length `n` *bytes right to left starting at the given* `x, y` *location.*
- void fl_scale (double x)

  Concatenate scaling transformation onto the current one.
- void fl_scale (double x, double y)

  Concatenate scaling transformation onto the current one.
- void fl_scroll (int X, int Y, int W, int H, int dx, int dy, void(∗draw_area)(void ∗, int, int, int, int), void ∗data)

  Scroll a rectangle and draw the newly exposed portions.
- void fl_set_spot (int font, int size, int X, int Y, int W, int H, Fl_Window ∗win=0)

  Inform text input methods about the current text insertion cursor.
- void fl_set_status (int X, int Y, int W, int H)

  Related to text input methods under X11.
- const char ∗ fl_shortcut_label (unsigned int shortcut)

  Get a human-readable string from a shortcut value.
- const char ∗ fl_shortcut_label (unsigned int shortcut, const char ∗∗eom)

  Get a human-readable string from a shortcut value.
- double fl_transform_dx (double x, double y)

  Transform distance using current transformation matrix.
- double fl_transform_dy (double x, double y)

  Transform distance using current transformation matrix.
- double fl_transform_x (double x, double y)

  Transform coordinate using the current transformation matrix.
- double fl_transform_y (double x, double y)

  Transform coordinate using the current transformation matrix.
- void fl_transformed_vertex (double xf, double yf)

  Add coordinate pair to the vertex list without further transformations.
- void fl_translate (double x, double y)

  Concatenate translation transformation onto the current one.
- void fl_vertex (double x, double y)

  Add a single vertex to the current path.
- void **fl_xyline** (int x, int y, int x1)

  Draw a horizontal line from (x,y) to (x1,y).
- void **fl_xyline** (int x, int y, int x1, int y2)

  Draw a horizontal line from (x,y) to (x1,y), then vertical from (x1,y) to (x1,y2).
- void **fl_xyline** (int x, int y, int x1, int y2, int x3)

  Draw a horizontal line from (x,y) to (x1,y), then a vertical from (x1,y) to (x1,y2) and then another horizontal from (x1,y2) to (x3,y2).
- void **fl_yxline** (int x, int y, int y1)

Draw a vertical line from (x,y) to (x,y1)

- void **fl_yxline** (int x, int y, int y1, int x2)

  Draw a vertical line from (x,y) to (x,y1), then a horizontal from (x,y1) to (x2,y1).

- void **fl_yxline** (int x, int y, int y1, int x2, int y3)

  Draw a vertical line from (x,y) to (x,y1), then a horizontal from (x,y1) to (x2,y1), then another vertical from (x2,y1) to (x2,y3).

11.8.1 Detailed Description

FLTK global graphics and GUI drawing functions.
These functions are declared in <FL/fl_draw.H>, and in <FL/platform.H> for offscreen buffer-related ones.

11.8.2 Enumeration Type Documentation

11.8.2.1 anonymous enum

```
anonymous enum
```

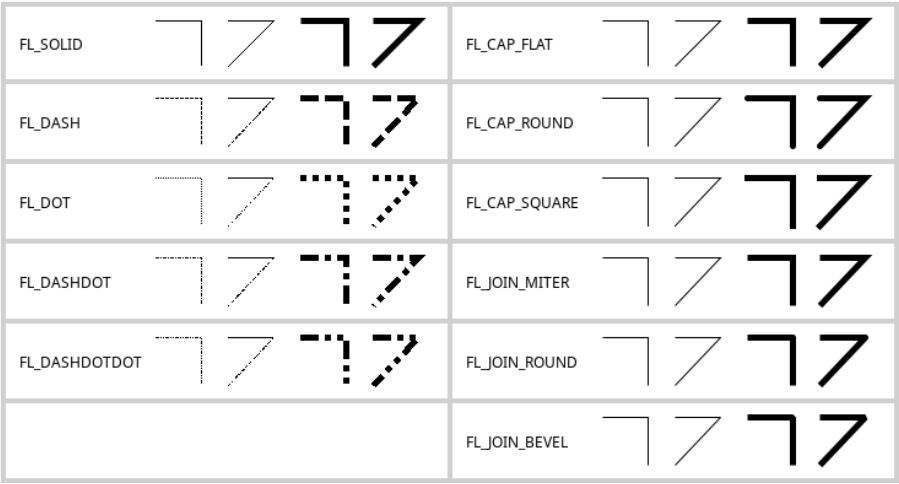


Figure 11.2 fl_line_style() styles

Enumerator

FL_SOLID	line style: solid line
FL_DASH	line style: 75% dashed line
FL_DOT	line style: 50% pixel dotted
FL_DASHDOT	line style: dash / dot pattern
FL_DASHDOTDOT	line style: dash / two dot pattern
FL_CAP_FLAT	cap style: end is flat
FL_CAP_ROUND	cap style: end is round
FL_CAP_SQUARE	cap style: end wraps end point
FL_JOIN_MITER	join style: line join extends to a point
FL_JOIN_ROUND	join style: line join is rounded
FL_JOIN_BEVEL	join style: line join is tidied
FL_UNIFORM_WIDTH	use uniform line width, even in scaled contexts

11.8.3 Function Documentation

11.8.3.1 fl_add_symbol() [1/2]

```
int fl_add_symbol (
            const char * name,
            void(*)(int x, int y, int w, int h, Fl_Color c) draw_in_rect,
            int scalable )
```

Registers (or replaces) a named symbol drawn into an explicit pixel rectangle.
Use this variant for symbols drawn with fast lines, text, or bitmaps, where working directly in pixel coordinates is more natural. If `scalable` is set, the matrix is pushed and pre-scaled as with the vector variant, so the symbol can also use complex-shape drawing calls in the (-1,-1) to (1,1) space.
See Drawing Complex Shapes for the FLTK drawing calls that work with the preset transform.

Parameters

in	*name*	symbol name without the leading "@"
in	*draw_in_rect*	drawing function; receives color and target rectangle; must not be null
in	*scalable*	1 (default) to allow scaling and rotation, 0 to keep fixed size

Returns

1 if the operation was successful (always).

See also

`fl_add_symbol(const char* name, void (* drawit)(Fl_Color c), int scalable)`

11.8.3.2 fl_add_symbol() [2/2]

```
int fl_add_symbol (
            const char * name,
            void(*)(Fl_Color c) drawit,
            int scalable )
```

Registers (or replaces) a named symbol drawn using complex vector drawing.
`drawit` is called with the matrix pushed and the origin translated to the center of the drawing rectangle. If `scalable` is set, the matrix is also pre-scaled so the unit square (-1,-1) to (1,1) maps onto the rectangle; use FLTK's complex-shape drawing calls in that coordinate space.
See Drawing Complex Shapes for the FLTK drawing calls that work with the preset transform.

Parameters

in	*name*	symbol name without the leading "@"
in	*drawit*	drawing function; must not be null
in	*scalable*	1 (default) to allow scaling and rotation, 0 to keep fixed size

Returns

1 if the operation was successful (always).

See also

`fl_add_symbol(const char* name, void (* draw_in_rect)(int x, int y, int w, int h, Fl_Color c), int scalable)`

11.8.3.3 fl_antialias()

```
void fl_antialias (
            int state )  [inline]
```

Turn antialiased line drawings ON or OFF, if supported by platform.
Currently, only the Windows platform allows to change whether line drawings are antialiased. Turning it OFF may accelerate heavy drawing operations.

11.8.3.4 fl_arc() [1/2]

```
void fl_arc (
            double x,
            double y,
            double r,
            double start,
            double end )  [inline]
```

Add a series of points to the current path on the arc of a circle.
The arc is drawn counter-clockwise from 3 o'clock. If `end` is less than `start` then it draws the arc in a clockwise direction. To draw an arc across the 3 o'clock line, `start` and `end` can be greater than 360 or less than 0. For example, to draw a counter-clockwise arc from 6 to 12 o'clock, `start` would be -90 deg, and `end` would be at +90 deg.
You can get elliptical paths by using scale and rotate before calling fl_arc().

Parameters

in	*x,y,r*	center and radius of circular arc
in	*start,end*	angles of start and end of arc measured in degrees

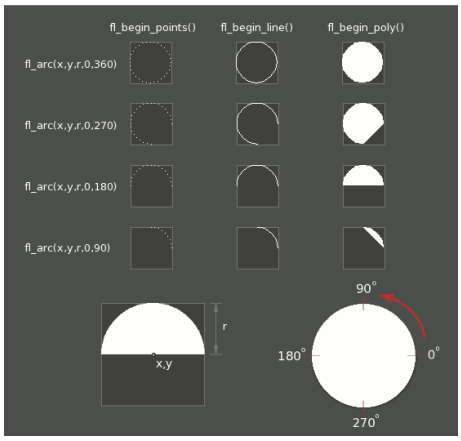


Figure 11.3 fl_arc(x,y,r,a1,a2)

Examples:

```
// Draw an arc of points
fl_begin_points();
fl_arc(100.0, 100.0, 50.0, 0.0, 180.0);
fl_end_points();

// Draw arc with a line
fl_begin_line();
fl_arc(200.0, 100.0, 50.0, 0.0, 180.0);
fl_end_line();

// Draw filled arc
```

```
fl_begin_polygon();
fl_arc(300.0, 100.0, 50.0, 0.0, 180.0);
fl_end_polygon();
```

11.8.3.5 fl_arc() [2/2]

```
void fl_arc (
            int x,
            int y,
            int w,
            int h,
            double a1,
            double a2 )  [inline]
```

Draw ellipse sections using integer coordinates.

These functions match the rather limited circle drawing code provided by X and Windows. The advantage over using fl_arc with floating point coordinates is that they are faster because they often use the hardware, and they draw much nicer small circles, since the small sizes are often hard-coded bitmaps.

If a complete circle is drawn it will fit inside the passed bounding box. The two angles are measured in degrees counter-clockwise from 3 o'clock and are the starting and ending angle of the arc, `a2` must be greater or equal to `a1`.

fl_arc() draws a series of lines to approximate the arc. Notice that the integer version of fl_arc() has a different number of arguments than the double version fl_arc(double x, double y, double r, double start, double end)

Parameters

in	*x,y,w,h*	bounding box of complete circle
in	*a1,a2*	start and end angles of arc measured in degrees counter-clockwise from 3 o'clock. `a2` must be greater than or equal to `a1`.

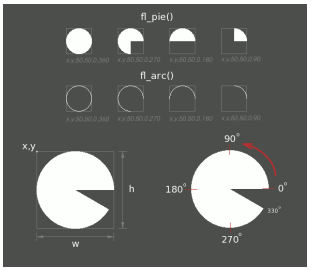


Figure 11.4 fl_pie() and fl_arc()

11.8.3.6 fl_begin_complex_polygon()

```
void fl_begin_complex_polygon ( )  [inline]
```

Start drawing a complex filled polygon.

The polygon may be concave, may have holes in it, or may be several disconnected pieces. Call fl_gap() to separate loops of the path.

To outline the polygon, use fl_begin_loop() and replace each fl_gap() with fl_end_loop();fl_begin_loop() pairs.

Note

For portability, you should only draw polygons that appear the same whether "even/odd" or "non-zero" winding rules are used to fill them. Holes should be drawn in the opposite direction to the outside loop.

11.8.3.7 fl_begin_offscreen()

```
void fl_begin_offscreen (
            Fl_Offscreen ctx )
```

Send all subsequent drawing commands to this offscreen buffer.

Parameters

ctx	the offscreen buffer.

Note

The `ctx` argument must have been created by fl_create_offscreen().

11.8.3.8 fl_begin_points()

```
void fl_begin_points ( ) [inline]
```

Start drawing a list of points.
Points are added to the list with fl_vertex().

11.8.3.9 fl_can_do_alpha_blending()

```
char fl_can_do_alpha_blending ( ) [inline]
```

Check whether platform supports true alpha blending for RGBA images.

Returns

1 if true alpha blending supported by platform

0 not supported so FLTK will use screen door transparency

11.8.3.10 fl_capture_window()

```
Fl_RGB_Image * fl_capture_window (
            Fl_Window * win,
            int x,
            int y,
            int w,
            int h )
```

Captures the content of a rectangular zone of a mapped window.

Parameters

win	a mapped Fl_Window (derived types including Fl_Gl_Window are also possible)
x,y,w,h	window area to be captured. Intersecting sub-windows are captured too.

Returns

The captured pixels as an Fl_RGB_Image. The raw and drawing sizes of the image can differ. Returns NULL when capture was not successful. The image depth may differ between platforms.

Version

1.4

11.8.3.11 fl_circle()

```
void fl_circle (
            double x,
            double y,
            double r ) [inline]
```

fl_circle(x,y,r) is equivalent to fl_arc(x,y,r,0,360), but may be faster.

Parameters

in	*x,y,r*	center and radius of circle

Note

fl_circle() is best used as part of the Drawing Complex Shapes API, that is, flanked by fl_begin_XXX() and fl_end_XXX() calls where XXX can be 'loop' or 'polygon' to draw, respectively a circle or a disk. Transformation functions (e.g., fl_scale(double, double)) can be also used for fl_circle() to draw empty of filled ellipses. It must be the *only* thing in the path: if you want a circle as part of a complex polygon you must use fl_arc(). Nevertheless, fl_circle() can also be used by itself to draw circles.

11.8.3.12 fl_clip_box()

```
int fl_clip_box (
            int x,
            int y,
            int w,
            int h,
            int & X,
            int & Y,
            int & W,
            int & H )  [inline]
```

Intersect a rectangle with the current clip region and return the bounding box of the result.
Returns non-zero if the resulting rectangle is different to the original. The given rectangle `(x, y, w, h)` *should* be entirely inside its window, otherwise the result may be unexpected, i.e. this function *may* not clip the rectangle to the window coordinates and size. In particular `x` and `y` *should* not be negative.
The resulting bounding box can be used to limit the necessary drawing to this rectangle.
Example:

```
void MyGroup::draw() {
  int X = 0, Y = 0, W = 0, H = 0;
  int ret = fl_clip_box(x(), y(), w(), h(), X, Y, W, H);
  if (ret == 0) { // entire group is visible (not clipped)
    // full drawing code here
  } else { // parts of this group are clipped
    // partial drawing code here (uses X, Y, W, and H to test)
  }
}
```

`W` and `H` are set to zero if the rectangle is completely outside the clipping region. In this case `X` and `Y` are undefined and should not be used. Possible values are `(0, 0)`, `(x, y)`, or anything else (platform dependent).

Note

This function is platform-dependent. If the given rectangle is not entirely inside the window, the results are not guaranteed to be the same on all platforms.

Parameters

in	*x,y,w,h*	position and size of rectangle
out	*X,Y,W,H*	position and size of resulting bounding box.

Returns

Non-zero if the resulting rectangle is different to the original.

See also

fl_not_clipped()

11.8.3.13 fl_clip_region() [1/2]

```
Fl_Region fl_clip_region ( ) [inline]
```

Return the current clipping region.

Note

This function is mostly intended for internal use by the FLTK library when drawing to the display. Its return value can be always NULL if the current drawing surface is not the display.

11.8.3.14 fl_clip_region() [2/2]

```
void fl_clip_region (
            Fl_Region r ) [inline]
```

Replace the top of the clipping stack with a clipping region of any shape.
Fl_Region is an operating system specific type.

Note

This function is mostly intended for internal use by the FLTK library when drawing to the display. Its effect can be null if the current drawing surface is not the display.

Parameters

in	*r*	clipping region

11.8.3.15 fl_copy_offscreen()

```
void fl_copy_offscreen (
            int x,
            int y,
            int w,
            int h,
            Fl_Offscreen pixmap,
            int srcx,
            int srcy ) [inline]
```

Copy a rectangular area of the given offscreen buffer into the current drawing destination.
This function is typically used when part of the GUI is drawn in an offscreen buffer created by fl_create_offscreen and copied, possibly in part, to the screen. A key condition for this to work independently of the screen scaling and resolution is to call fl_create_offscreen when the destination window of the offscreen graphics is show()'n and current. Another condition is to resize the offscreen when the display containing this window is rescaled. This can be done calling fl_rescale_offscreen which resizes the offscreen while preserving its graphical content. Another approach is to delete and re-create the offscreen, and redraw its content. FLTK test program offscreen gives an example of proper use of these functions.

Parameters

x,y	position where to draw the copied rectangle in destination window
w,h	size of the copied rectangle in FLTK units
pixmap	offscreen buffer containing the rectangle to copy
srcx,srcy	origin in offscreen buffer of rectangle to copy

See also

fl_create_offscreen, fl_rescale_offscreen.

11.8.3.16 fl_create_offscreen()

```
Fl_Offscreen fl_create_offscreen (
            int w,
            int h )
```

Creation of an offscreen graphics buffer.

Parameters

w,h	width and height in FLTK units of the buffer.

Returns

the created graphics buffer.

The pixel size of the created graphics buffer is equal to the number of pixels in an area of the screen containing the current window sized at w,h FLTK units. This pixel size varies with the value of the scale factor of this screen.

Note

Work with the fl_XXX_offscreen() functions is equivalent to work with an Fl_Image_Surface object, as follows :

Fl_Offscreen-based approach	Fl_Image_Surface-based approach
Fl_Offscreen off = fl_create_offscreen(w, h)	Fl_Image_Surface *surface = new Fl_Image_Surface(w, h, 1)
fl_begin_offscreen(off)	Fl_Surface_Device::push_current(surface)
fl_end_offscreen()	Fl_Surface_Device::pop_current()
fl_copy_offscreen(x,y,w,h, off, sx,sy)	fl_copy_offscreen(x,y,w,h, surface->offscreen(), sx,sy)
fl_rescale_offscreen(off)	surface->rescale()
fl_delete_offscreen(off)	delete surface

11.8.3.17 fl_cursor()

```
void fl_cursor (
            Fl_Cursor c )
```

Sets the cursor for the current window to the specified shape and colors.
The cursors are defined in the <FL/Enumerations.H> header file.

11.8.3.18 fl_curve()

```
void fl_curve (
            double X0,
            double Y0,
            double X1,
            double Y1,
            double X2,
            double Y2,
            double X3,
            double Y3 )  [inline]
```

Add a series of points on a Bézier curve to the path.
The curve ends (and two of the points) are at X0,Y0 and X3,Y3.

Parameters

in	*X0,Y0*	curve start point
in	*X1,Y1*	curve control point
in	*X2,Y2*	curve control point

Parameters

in	*X3,Y3*	curve end point

11.8.3.19 fl_delete_offscreen()

```
void fl_delete_offscreen (
            Fl_Offscreen ctx )
```

Deletion of an offscreen graphics buffer.

Parameters

ctx	the buffer to be deleted.

Note

The `ctx` argument must have been created by fl_create_offscreen().

11.8.3.20 fl_draw() [1/5]

```
void fl_draw (
            const char * str,
            int x,
            int y )
```

Draw a nul-terminated UTF-8 string starting at the given `x`, `y` location.
Text is aligned to the left and to the baseline of the font. To align to the bottom, subtract fl_descent() from `y`. To align to the top, subtract fl_descent() and add fl_height(). This version of fl_draw provides direct access to the text drawing function of the underlying OS. It does not apply any special handling to control characters.

11.8.3.21 fl_draw() [2/5]

```
void fl_draw (
            const char * str,
            int x,
            int y,
            int w,
            int h,
            Fl_Align align,
            Fl_Image * img = 0,
            int draw_symbols = 1,
            int spacing = 0 )
```

Fancy string drawing function which is used to draw all the labels.
The string is formatted and aligned inside the passed box. Handles '\t' and '\n', expands all other control characters to '^X', and aligns inside or against the edges of the box. See Fl_Widget::align() for values of `align`. The value FL_ALIGN_INSIDE is ignored, as this function always prints inside the box. If `img` is provided and is not `NULL`, the image is drawn above or below the text as specified by the `align` value. The `draw_symbols` argument specifies whether or not to look for symbol names starting with the '@' character'

Parameters

in	*str*	UTF-8 string, can start and end with an '@sym' symbol, can contain '\n'
in	*x,y,w,h*	bounding box
in	*align*	label and image alignment in bounding box
in	*img*	pointer to image
in	*draw_symbols*	if true, interprete leading and trailing '@sym' as graphical symbols
in	*spacing*	spacing between text and image

11.8.3.22 fl_draw() [3/5]

```
void fl_draw (
            const char * str,
            int x,
            int y,
            int w,
            int h,
            Fl_Align align,
            void(*)(const char *, int, int, int) callthis,
            Fl_Image * img = 0,
            int draw_symbols = 1,
            int spacing = 0 )
```

The same as fl_draw(const char∗,int,int,int,int,Fl_Align,Fl_Image∗,int) with the addition of the `callthis` parameter, which is a pointer to a text drawing function such as fl_draw(const char∗, int, int, int) to do the real work.

Parameters

in	*str*	UTF-8 string, can start and end with an '@sym' symbol, can contain '\n'
in	*x,y,w,h*	bounding box
in	*align*	label and image alignment in bounding box
in	*callthis*	pointer to text drawing function
in	*img*	pointer to image
in	*draw_symbols*	if true, interprete leading and trailing '@sym' as graphical symbols
in	*spacing*	spacing between text and image

11.8.3.23 fl_draw() [4/5]

```
void fl_draw (
            int angle,
            const char * str,
            int n,
            int x,
            int y )  [inline]
```

Draw at the given `x`, `y` location a UTF-8 string of length `n` bytes rotating `angle` degrees counter-clockwise.

Note

When using X11 (Unix, Linux, Cygwin et al.) this needs Xft to work. Under plain X11 (w/o Xft) rotated text is not supported by FLTK. A warning will be issued to stderr at runtime (only once) if you use this method with an angle other than 0.

11.8.3.24 fl_draw() [5/5]

```
void fl_draw (
            int angle,
            const char * str,
            int x,
            int y )
```

Draw a nul-terminated UTF-8 string starting at the given `x`, `y` location and rotating `angle` degrees counter-clockwise.
This version of fl_draw provides direct access to the text drawing function of the underlying OS and is supported by all fltk platforms except X11 without Xft.

11.8.3.25 fl_draw_arrow()

```
void fl_draw_arrow (
            Fl_Rect r,
```

```
            Fl_Arrow_Type t,
            Fl_Orientation o,
            Fl_Color col )
```

Draw an "arrow like" GUI element for the selected scheme.
In the future this function should be integrated in Fl_Scheme as a virtual method, i.e. it would call a method like ...
`Fl_Scheme::current()->draw_arrow(r, t, o, col);`

Parameters

in	*r*	bounding box
in	*t*	arrow type
in	*o*	orientation
in	*col*	arrow color

Since

1.4.0

11.8.3.26 fl_draw_box()

```
void fl_draw_box (
            Fl_Boxtype t,
            int x,
            int y,
            int w,
            int h,
            Fl_Color c )
```

Draws a box using given type, position, size and color.

Parameters

in	*t*	box type
in	*x,y,w,h*	position and size
in	*c*	color

11.8.3.27 fl_draw_box_focus()

```
void fl_draw_box_focus (
            Fl_Boxtype bt,
            int x,
            int y,
            int w,
            int h,
            Fl_Color fg,
            Fl_Color bg )
```

Draws the focus rectangle inside a box using given type, position, size and color.
Boxes can set their own focus drawing callback. The focus frame does not need to be a rectangle at all, but should fit inside the shape of the box.

Parameters

in	*bt*	box type
in	*x,y,w,h*	position and size
in	*fg,bg*	foreground and background color

11.8.3.28 fl_draw_check()

```
void fl_draw_check (
            Fl_Rect bb,
            Fl_Color col )
```

Draw a check mark inside the given bounding box.
The check mark is allowed to fill the entire box but the algorithm used makes sure that a 1-pixel border is kept free if the box is large enough. You need to calculate margins for box borders etc. yourself.
The check mark size is limited (minimum and maximum size) and the check mark is always centered in the given box.

Note

If the box is too small (bad GUI design) the check mark will be drawn over the box borders. This is intentional for better user experience. Otherwise users might not be able to recognize if a box is checked.

The size limits are implementation details and may be changed at any time.

Parameters

in	*bb*	rectangle that defines the bounding box
in	*col*	Fl_Color to draw the check mark

Since

1.4.0

11.8.3.29 fl_draw_circle()

```
void fl_draw_circle (
            int x,
            int y,
            int d,
            Fl_Color color )
```

Draw a potentially small, filled circle using a given color.
This function draws a filled circle bounded by rectangle `(x, y, d, d)` using color `color`
This function is the same as `fl_pie(x, y, d, d, 0, 360)` except with some systems that don't draw small circles well. In that situation, the circle diameter `d` is converted from FLTK units to pixels and this function approximates a filled circle by drawing several filled rectangles if the converted diameter is 6 pixels.
The current drawing color fl_color() is preserved across the call.

Parameters

in	*x,y*	coordinates of top left of the bounding box
in	*d*	diameter == width and height of the bounding box in FLTK units
in	*color*	the color used to draw the circle

Since

1.4.0

11.8.3.30 fl_draw_image() [1/2]

```
void fl_draw_image (
            const uchar * buf,
            int X,
            int Y,
            int W,
```

```
int H,
int D = 3,
int L = 0 )  [inline]
```

Draw an 8-bit per color RGB or luminance image.

Parameters

in	*buf*	points at the "r" data of the top-left pixel. Color data must be in `r,g,b` order. Luminance data is only one `gray` byte.
in	*X,Y*	position where to put top-left corner of image
in	*W,H*	size of the image
in	*D*	delta to add to the pointer between pixels. It may be any value greater than or equal to 1, or it can be negative to flip the image horizontally
in	*L*	delta to add to the pointer between lines (if 0 is passed it uses `W * D`), and may be larger than `W * D` to crop data, or negative to flip the image vertically

It is highly recommended that you put the following code before the first `show()` of *any* window in your program to get rid of the dithering if possible:

```
Fl::visual(FL_RGB);
```

Gray scale (1-channel) images may be drawn. This is done if `abs(D)` is less than 3, or by calling fl_draw_image_mono(). Only one 8-bit sample is used for each pixel, and on screens with different numbers of bits for red, green, and blue only gray colors are used. Setting `D` greater than 1 will let you display one channel of a color image.

Note:

The X version does not support all possible visuals. If FLTK cannot draw the image in the current visual it will abort. FLTK supports any visual of 8 bits or less, and all common TrueColor visuals up to 32 bits.

11.8.3.31 fl_draw_image() [2/2]

```
void fl_draw_image (
            Fl_Draw_Image_Cb cb,
            void * data,
            int X,
            int Y,
            int W,
            int H,
            int D = 3 )  [inline]
```

Draw an image using a callback function to generate image data.
You can generate the image as it is being drawn, or do arbitrary decompression of stored data, provided it can be decompressed to individual scan lines.

Parameters

in	*cb*	callback function to generate scan line data
in	*data*	user data passed to callback function
in	*X,Y*	screen position of top left pixel
in	*W,H*	image width and height
in	*D*	data size per pixel in bytes (must be greater than 0)

See also

fl_draw_image(const uchar* buf, int X, int Y, int W, int H, int D, int L)

The callback function `cb` is called with the `void* data` user data pointer to allow access to a structure of information about the image, and the `x`, `y`, and `w` of the scan line desired from the image. 0,0 is the upper-left corner

of the image, not x, y. A pointer to a buffer to put the data into is passed. You must copy w pixels from scanline y, starting at pixel x, to this buffer.
Due to cropping, less than the whole image may be requested. So x may be greater than zero, the first y may be greater than zero, and w may be less than W. The buffer is long enough to store the entire W * D pixels, this is for convenience with some decompression schemes where you must decompress the entire line at once: decompress it into the buffer, and then if x is not zero, copy the data over so the x'th pixel is at the start of the buffer.
You can assume the y's will be consecutive, except the first one may be greater than zero.
If D is 4 or more, you must fill in the unused bytes with zero.

11.8.3.32 fl_draw_image_mono() [1/2]

```
void fl_draw_image_mono (
            const uchar * buf,
            int X,
            int Y,
            int W,
            int H,
            int D = 1,
            int L = 0 )  [inline]
```

Draw a gray-scale (1 channel) image.

See also

fl_draw_image(const uchar* buf, int X,int Y,int W,int H, int D, int L)

11.8.3.33 fl_draw_image_mono() [2/2]

```
void fl_draw_image_mono (
            Fl_Draw_Image_Cb cb,
            void * data,
            int X,
            int Y,
            int W,
            int H,
            int D = 1 )  [inline]
```

Draw a gray-scale image using a callback function to generate image data.

See also

fl_draw_image(Fl_Draw_Image_Cb cb, void* data, int X,int Y,int W,int H, int D)

11.8.3.34 fl_draw_pixmap() [1/2]

```
int fl_draw_pixmap (
            char *const * data,
            int x,
            int y,
            Fl_Color bg = FL_GRAY )  [inline]
```

Draw XPM image data, with the top-left corner at the given position.

See also

fl_draw_pixmap(const char* const* data, int x, int y, Fl_Color bg)

11.8.3.35 fl_draw_pixmap() [2/2]

```
int fl_draw_pixmap (
            const char *const * cdata,
            int x,
```

```
            int y,
            Fl_Color bg = FL_GRAY )
```

Draw XPM image data, with the top-left corner at the given position.
The image is dithered on 8-bit displays so you won't lose color space for programs displaying both images and pixmaps.

Parameters

in	*cdata*	pointer to XPM image data
in	*x,y*	position of top-left corner
in	*bg*	background color

Returns

0 if there was any error decoding the XPM data.

11.8.3.36 fl_draw_radio()

```
void fl_draw_radio (
            int x,
            int y,
            int d,
            Fl_Color color )
```

Draw a round check mark (circle) of a radio button.
This draws only the round "radio button mark", it does not draw the (also typically round) box of the radio button.
Call this only if the radio button is `ON`.
This method draws a scheme specific "circle" with a particular light effect if the scheme is gtk+. For all other schemes this function draws a simple, small circle.
The `color` must be chosen by the caller so it has enough contrast with the background.
The bounding box of the circle is the rectangle `(x, y, d, d)`.
The current drawing color fl_color() is preserved across the call.

Parameters

in	*x,y*	coordinates of top left of the bounding box
in	*d*	diameter == width and height of the bounding box in FLTK units
in	*color*	the base color used to draw the circle

Since

1.4.0

11.8.3.37 fl_draw_symbol()

```
int fl_draw_symbol (
            const char * label,
            int x,
            int y,
            int w,
            int h,
            Fl_Color col )
```

Draw the named symbol in the given rectangle using the given color.

Parameters

in	*label*	name of symbol

Parameters

in	*x,y*	position of symbol
in	*w,h*	size of symbol
in	*col*	color of symbol

Returns

1 on success, 0 on failure

11.8.3.38 fl_expand_text()

```
const char * fl_expand_text (
            const char * from,
            char * buf,
            int maxbuf,
            double maxw,
            int & n,
            double & width,
            int wrap,
            int draw_symbols )
```

Copy `from` to `buf`, replacing control characters with ^X.
Stop at a newline or if `maxbuf` characters written to buffer. Also word-wrap if width exceeds maxw. Returns a pointer to the start of the next line of characters. Sets n to the number of characters put into the buffer. Sets width to the width of the string in the current font.

Parameters

in	*from*	input text, can contain ' ' and single or double '@' characters
out	*buf*	buffer for the output text segment
in	*maxbuf*	size of the buffer, or 0 to use the internal buffer allocated in this function
in	*maxw*	maximum width in pixels of the output text
out	*n*	number of characters in the output text segment
out	*width*	actual width in pixels of the output text
in	*wrap*	if true, wrap at maxw, else wrap at newlines
in	*draw_symbols*	if true, double '@' characters are escaped into a single '@'

Returns

pointer to the next character in the input text

11.8.3.39 fl_focus_rect()

```
void fl_focus_rect (
            int x,
            int y,
            int w,
            int h )  [inline]
```

Draw a dotted rectangle, used to indicate keyboard focus on a widget.
This method draws the rectangle in the current color and independent of the Fl::visible_focus() option. You may need to set the current color with fl_color() before you call this.

11.8.3.40 fl_frame()

```
void fl_frame (
```

```
            const char * s,
            int x,
            int y,
            int w,
            int h )
```

Draws a series of line segments around the given box.
The string `s` must contain groups of 4 letters which specify one of 24 standard grayscale values, where 'A' is black and 'X' is white. The order of each set of 4 characters is: top, left, bottom, right. The result of calling fl_frame() with a string that is not a multiple of 4 characters in length is undefined. The only difference between this function and fl_frame2() is the order of the line segments.

Parameters

in	*s*	sets of 4 grayscale values in top, left, bottom, right order
in	*x,y,w,h*	position and size

11.8.3.41 fl_frame2()

```
void fl_frame2 (
            const char * s,
            int x,
            int y,
            int w,
            int h )
```

Draws a series of line segments around the given box.
The string `s` must contain groups of 4 letters which specify one of 24 standard grayscale values, where 'A' is black and 'X' is white. The order of each set of 4 characters is: bottom, right, top, left. The result of calling fl_frame2() with a string that is not a multiple of 4 characters in length is undefined. The only difference between this function and fl_frame() is the order of the line segments.

Parameters

in	*s*	sets of 4 grayscale values in bottom, right, top, left order
in	*x,y,w,h*	position and size

11.8.3.42 fl_gap()

```
void fl_gap ( )  [inline]
```

Separate loops of the path.
It is unnecessary but harmless to call fl_gap() before the first vertex, after the last vertex, or several times in a row.

11.8.3.43 fl_line_style()

```
void fl_line_style (
            int style,
            int width = 0,
            char * dashes = 0 )  [inline]
```

Set how to draw lines (the "pen").
If you change this it is your responsibility to set it back to the default using `fl_line_style(0)`.

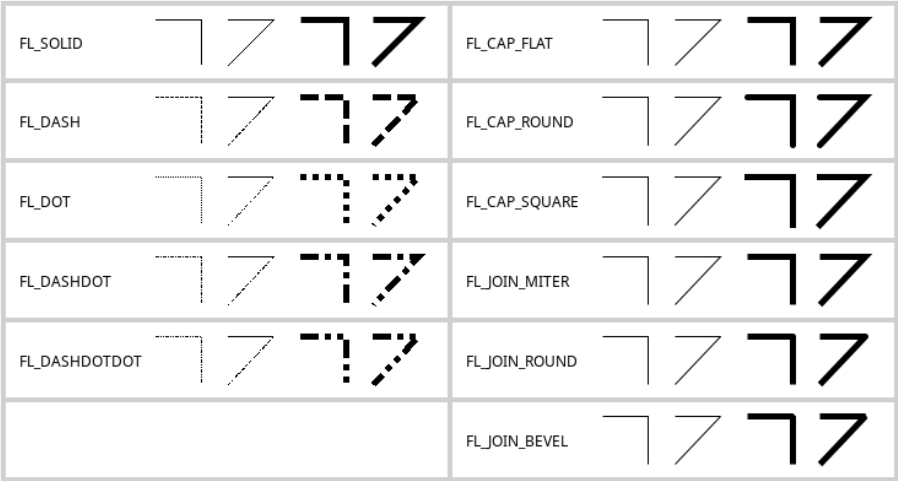


Figure 11.5 fl_line_style() styles

Parameters

in	*style*	A bitmask which is a bitwise-OR of Line Styles, a cap style, and a join style. If you don't specify a dash type you will get a solid line. If you don't specify a cap or join type you will get a system-defined default of whatever value is fastest.
in	*width*	The thickness of the lines in pixels. Zero results in the system defined default, which on both X and Windows is somewhat different and nicer than 1.
in	*dashes*	A pointer to an array of dash lengths, measured in pixels. The first location is how long to draw a solid portion, the next is how long to draw the gap, then the solid, etc. It is terminated with a zero-length entry. A `NULL` pointer or a zero-length array results in a solid line. Odd array sizes are not supported and result in undefined behavior.

In scaled mode, horizontal and vertical lines are drawn with different line widths, depending on the position of the line on screen, to adapt for gaps when drawing adjacent lines. Setting `FL_UNIFORM_WIDTH` will make all lines the same width, regardless of their position. This improves line drawings, but may produce small gaps when drawing gradients or filling areas.

Note

When changing the line style inside a `draw()` method, make sure to reset the line styles by calling `fl_line_style(0)` at the end of a drawing routine or FLTK frame and box drawing will fail.

Because of how line styles are implemented on Win32 systems, you *must* set the line style *after* setting the drawing color. If you set the color after the line style you will lose the line style settings.

The `dashes` array does not work under the (unsupported!) operating systems Windows 95, 98 or Me, since those operating systems do not support complex line styles.

11.8.3.44 fl_load_matrix()

```
void fl_load_matrix (
            double a,
            double b,
            double c,
            double d,
            double x,
            double y )  [inline]
```

Set the current transformation matrix.

Parameters

in	*a,b,c,d,x,y*	transformation matrix elements

11.8.3.45 fl_measure()

```
void fl_measure (
            const char * str,
            int & w,
            int & h,
            int draw_symbols )
```

Measure how wide and tall the string will be when printed by the fl_draw() function with `align` parameter.
If the incoming `w` is non-zero it will wrap to that width.
The current font is used to do the width/height calculations, so unless its value is known at the time fl_measure() is called, it is advised to first set the current font with fl_font(). With event-driven GUI programming you can never be sure which widget was exposed and redrawn last, nor which font it used. If you have not called fl_font() explicitly in your own code, the width and height may be set to unexpected values, even zero!
Note: In the general use case, it's a common error to forget to set `w` to 0 before calling fl_measure() when wrap behavior isn't needed.

Parameters

in	*str*	nul-terminated string
in,out	*w*	call with w=0, or with the prefered width for word wrapping, returns with the width of the string in current font
out	*h*	height of string in current font
in	*draw_symbols*	non-zero to enable @symbol handling [default=1]

```
// Example: Common use case for fl_measure()
const char *s = "This is a test";
int wi=0, hi=0;             // initialize to zero before calling fl_measure()
fl_font(FL_HELVETICA, 14);  // set current font face/size to be used for measuring
fl_measure(s, wi, hi);      // returns pixel width/height of string in current font
```

11.8.3.46 fl_measure_pixmap() [1/2]

```
int fl_measure_pixmap (
            char *const * data,
            int & w,
            int & h )
```

Get the dimensions of a pixmap.
An XPM image contains the dimensions in its data. This function returns the width and height.

Parameters

in	*data*	pointer to XPM image data.
out	*w,h*	width and height of image

Returns

non-zero if the dimensions were parsed OK

0 if there were any problems

11.8.3.47 fl_measure_pixmap() [2/2]

```
int fl_measure_pixmap (
            const char *const * cdata,
```

```
            int & w,
            int & h )
```

Get the dimensions of a pixmap.

See also

fl_measure_pixmap(char* const* data, int &w, int &h)

11.8.3.48 fl_mult_matrix()

```
void fl_mult_matrix (
            double a,
            double b,
            double c,
            double d,
            double x,
            double y )  [inline]
```

Concatenate another transformation onto the current one.

Parameters

in	*a,b,c,d,x,y*	transformation matrix elements such that `X' = aX + cY + x` and `Y' = bX +dY + y`

11.8.3.49 fl_not_clipped()

```
int fl_not_clipped (
            int x,
            int y,
            int w,
            int h )  [inline]
```

Does the rectangle intersect the current clip region?

Parameters

in	*x,y,w,h*	position and size of rectangle

Returns

non-zero if any of the rectangle intersects the current clip region. If this returns 0 you don't have to draw the object.

Note

Under X this returns 2 if the rectangle is partially clipped and 1 if it is entirely inside the clip region.

See also

fl_clip_box()

11.8.3.50 fl_old_shortcut()

```
unsigned int fl_old_shortcut (
            const char * s )
```

Emulation of XForms named shortcuts.
Converts ASCII shortcut specifications (eg. "^c") into the FLTK integer equivalent (eg. FL_CTRL+'c')
These ASCII characters are used to specify the various keyboard modifier keys:

```
# - Alt
+ - Shift
^ - Control
! - Meta
@ - Command (Ctrl on linux/win, Meta on OSX)
```

These special characters can be combined to form chords of modifier keys. (See 'Remarks' below)
After the optional modifier key prefixes listed above, one can either specify a single keyboard character to use as the shortcut, or a numeric sequence in hex, decimal or octal.
Examples:

```
"c"      -- Uses 'c' as the shortcut
"#^c"    -- Same as FL_ALT|FL_CTRL|'c'
"#^!c"   -- Same as FL_ALT|FL_CTRL|FL_META|'c'
"@c"     -- Same as FL_COMMAND|'c' (see FL_COMMAND for platform specific behavior)
"0x63"   -- Same as "c" (hex 63=='c')
"99"     -- Same as "c" (dec 99=='c')
"0143"   -- Same as "c" (octal 0143=='c')
"^0x63"  -- Same as (FL_CTRL|'c'), or (FL_CTRL|0x63)
"^99"    -- Same as (FL_CTRL|'c'), or (FL_CTRL|99)
"^0143"  -- Same as (FL_CTRL|'c'), or (FL_CTRL|0143)
```

Remarks

Due to XForms legacy, there are some odd things to consider when using the modifier characters.

(1) You can use the special modifier keys for chords *only* if the modifiers are provided in this order: #, +, ^, !, @. Other ordering can yield undefined results.

So for instance, Ctrl-Alt-c must be specified as "#^c" (and not "^#c"), due to the above ordering rule.

(2) If you want to make a shortcut that uses one of the special modifier characters (as the character being modified), then to avoid confusion, specify the numeric equivalent, e.g.

```
If you want..                    Then use..
-----------------------------    -----------------------------
'#' as the shortcut..            "0x23"  (instead of just "#").
'+' as the shortcut..            "0x2b"  (instead of just "+").
'^' as the shortcut..            "0x5e"  (instead of just "^").
Alt-+ as the shortcut..          "#0x2b" (instead of "#+").
Alt-^ as the shortcut..          "#0x5e" (instead of "#^").
..etc..
```

As a general rule that's easy to remember, unless the shortcut key to be modified is a single alpha-numeric character [A-Z,a-z,0-9), it's probably best to use the numeric equivalents.

Don't fix these silly legacy issues in a future release. Nobody is using this anymore.

11.8.3.51 fl_overlay_clear()

`void fl_overlay_clear ( )`

Erase a selection rectangle without drawing a new one.

See also

fl_overlay_rect(int x, int y, int w, int h)

11.8.3.52 fl_overlay_rect()

```
void fl_overlay_rect (
            int x,
            int y,
            int w,
            int h )
```

Draw a transient dotted selection rectangle.
This function saves the current screen content and then draws a dotted selection rectangle into the front screen buffer. If another selection rectangle was drawn earlier, the previous screen graphics are restored first.

To clear the selection rectangle, call `fl_overlay_clear()`.
The typical (and only) use for this function is to draw a selection rectangle during a mouse drag event sequence without having to redraw the entire content of the widget.
Your event handle should look similar to this (also see `test/mandelbrot.cxx`):

```
int MyWidget::handle(int event) {
  switch (event) {
    case FL_PUSH:
      ix = Fl::event_x(); // ix defined as (private) class member
      iy = Fl::event_y(); // iy defined as (private) class member
      return 1;
    case FL_DRAG:
      window()->make_current();
      fl_overlay_rect(ix, iy, Fl::event_x() - ix, Fl::event_y() - iy);
      return 1;
    case FL_RELEASE:
      window()->make_current();
      fl_overlay_clear();
      // select the element under the rectangle
      return 1;
  }
  return MySuperWidget::handle(event);
}
```

Note

Between drawing an overlay rect and clearing it, the content of the widget must not change.

fl_overlay_rect() and fl_overlay_clear() should be called when the actual event occurs, and *not* within `My↩Widget::draw()`.

fl_overlay_rect() and fl_overlay_clear() should not be mixed with Fl_Overlay_Window. Fl_Overlay_Window provides an entirely different way of drawing selection outlines and is not limited to rectangles.

Parameters

x,y,w,h	position and size of the overlay rectangle.

See also

fl_overlay_clear()

11.8.3.53 fl_override_scale()

```
float fl_override_scale ( )
```

Removes any GUI scaling factor in subsequent drawing operations.
This must be matched by a later call to fl_restore_scale(). This function can be used to transiently perform drawing operations that are not rescaled by the current value of the GUI scaling factor. It's not supported to call fl_push_clip() while transiently removing scaling.

Returns

The GUI scaling factor value that was in place when the function started.

11.8.3.54 fl_pie()

```
void fl_pie (
            int x,
            int y,
            int w,
            int h,
            double a1,
            double a2 )  [inline]
```

Draw filled ellipse sections using integer coordinates.
Like fl_arc(), but fl_pie() draws a filled-in pie slice. This slice may extend outside the line drawn by fl_arc(); to avoid this use w - 1 and h - 1.

Parameters

in	*x,y,w,h*	bounding box of complete circle
in	*a1,a2*	start and end angles of arc measured in degrees counter-clockwise from 3 o'clock. `a2` must be greater than or equal to `a1`.

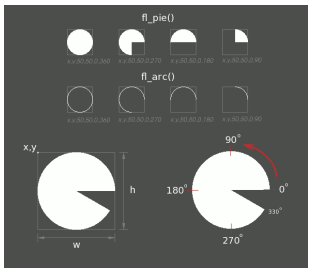


Figure 11.6 fl_pie() and fl_arc()

11.8.3.55 fl_polygon()

```
void fl_polygon (
            int x,
            int y,
            int x1,
            int y1,
            int x2,
            int y2,
            int x3,
            int y3 )  [inline]
```

Fill a 4-sided polygon.
The polygon must be convex.

11.8.3.56 fl_pop_clip()

```
void fl_pop_clip ( )  [inline]
```

Restore the previous clip region.
You must call fl_pop_clip() once for every time you call fl_push_clip(). Unpredictable results may occur if the clip stack is not empty when you return to FLTK.

11.8.3.57 fl_push_clip()

```
void fl_push_clip (
            int x,
            int y,
            int w,
            int h )  [inline]
```

Intersect the current clip region with a rectangle and push this new region onto the stack.

Parameters

in	*x,y,w,h*	position and size

11.8.3.58 fl_push_matrix()

```
void fl_push_matrix ( )  [inline]
```

Save the current transformation matrix on the stack.

The maximum depth of the stack is 32.

11.8.3.59 fl_read_image()

```
uchar * fl_read_image (
            uchar * p,
            int X,
            int Y,
            int w,
            int h,
            int alpha )
```

Reads an RGB(A) image from the current window or off-screen buffer.

Parameters

in	*p*	pixel buffer, or NULL to allocate one
in	*X,Y*	position of top-left of image to read
in	*w,h*	width and height of image to read
in	*alpha*	alpha value for image (0 for none)

Returns

pointer to pixel buffer that should be freed using `delete[]`, or `nullptr` if allocation failed.

The `p` argument points to a buffer that can hold the image and must be at least `w*h*3` bytes when reading RGB images, or `w*h*4` bytes when reading RGBA images. If NULL, fl_read_image() will create an array of the proper size which can be freed using `delete[]`.
The `alpha` parameter controls whether an alpha channel is created and the value that is placed in the alpha channel. If 0, no alpha channel is generated.

See also

fl_capture_window()

11.8.3.60 fl_rect() [1/3]

```
void fl_rect (
            Fl_Rect r )  [inline]
```

Draw a border *inside* the given bounding box.
This is the same as fl_rect(int x, int y, int w, int h) but with Fl_Rect `r` as input argument.

11.8.3.61 fl_rect() [2/3]

```
void fl_rect (
            int x,
            int y,
            int w,
            int h )  [inline]
```

Draw a border *inside* the given bounding box.
This function is meant for quick drawing of simple boxes. The behavior is undefined for line widths that are not 1.

11.8.3.62 fl_rect() [3/3]

```
void fl_rect (
            int x,
            int y,
            int w,
            int h,
            Fl_Color c )  [inline]
```

Draw with passed color a border *inside* the given bounding box.

Warning

The current color is changed to `c` upon return.

11.8.3.63 fl_rectf() [1/4]

```
void fl_rectf (
            Fl_Rect bb,
            uchar r,
            uchar g,
            uchar b )  [inline]
```

Color a rectangle with "exactly" the passed `r,g,b` color.
This is the same as fl_rectf(int x, int y, int w, int h, uchar r, uchar g, uchar b) but with Fl_Rect `bb` (bounding box) as argument instead of (x, y, w, h).

See also

fl_rectf(int x, int y, int w, int h, uchar r, uchar g, uchar b)

11.8.3.64 fl_rectf() [2/4]

```
void fl_rectf (
            Fl_Rect r,
            Fl_Color c )  [inline]
```

Color with passed color a rectangle that exactly fills the given bounding box.

Warning

The current color is changed to `c` upon return.

11.8.3.65 fl_rectf() [3/4]

```
void fl_rectf (
            int x,
            int y,
            int w,
            int h,
            Fl_Color c )  [inline]
```

Color with passed color a rectangle that exactly fills the given bounding box.

Warning

The current color is changed to `c` upon return.

11.8.3.66 fl_rectf() [4/4]

```
void fl_rectf (
            int x,
            int y,
            int w,
            int h,
            uchar r,
            uchar g,
            uchar b )  [inline]
```

Color a rectangle with "exactly" the passed `r,g,b` color.
On screens with less than 24 bits of color this is done by drawing a solid-colored block using fl_draw_image() so that the correct color shade is produced. On other screens, the current color is changed to `fl_color(r,g,b)` upon return.

11.8.3.67 fl_remove_symbol()

```
int fl_remove_symbol (
            const char * name )
```

Removes the symbol with name `name` from the system.
If a symbol with that name is not defined, this function does nothing.

Parameters

in	*name*	name of symbol (without the "@")

Returns

1 on success (always).

11.8.3.68 fl_rescale_offscreen()

```
void fl_rescale_offscreen (
            Fl_Offscreen & ctx )
```

Adapts an offscreen buffer to a changed value of the scale factor.
The `ctx` argument must have been created by fl_create_offscreen() and the calling context must not be between fl_begin_offscreen() and fl_end_offscreen(). The graphical content of the offscreen is preserved. The current scale factor value is given by `Fl_Graphics_Driver::default_driver().scale()`.

Version

1.4

11.8.3.69 fl_reset_spot()

```
void fl_reset_spot (
            void )
```

Resets marked text.
In many languages, typing a character can involve multiple keystrokes. For example, the Ä can be composed of two dots (¨) on top of the character, followed by the letter A (on a Mac with U.S. keyboard, you'd type Alt-U, Shift-A. To inform the user that the dots may be followed by another character, the ¨ is underlined).
Call this function if character composition needs to be aborted for some reason. One such example would be the text input widget losing focus.

11.8.3.70 fl_restore_scale()

```
void fl_restore_scale (
            float s )
```

Restores the GUI scaling factor in subsequent drawing operations.

Parameters

s	Value returned by a previous call to fl_override_scale().

11.8.3.71 fl_rotate()

```
void fl_rotate (
            double d )  [inline]
```

Concatenate rotation transformation onto the current one.

Parameters

in	*d*	- rotation angle, counter-clockwise in degrees (not radians)

11.8.3.72 fl_rounded_rect()

```
void fl_rounded_rect (
            int x,
            int y,
            int w,
            int h,
            int r )  [inline]
```

Draw a rounded border *inside* the given bounding box.
The radius code is optimized for speed and works best for values between 5 and 15 units.

11.8.3.73 fl_rounded_rectf()

```
void fl_rounded_rectf (
            int x,
            int y,
            int w,
            int h,
            int r )  [inline]
```

Color with current color a rounded rectangle that exactly fills the given bounding box.
The radius code is optimized for speed and works best for values between 5 and 15 units.

11.8.3.74 fl_scale() [1/2]

```
void fl_scale (
            double x )  [inline]
```

Concatenate scaling transformation onto the current one.

Parameters

in	*x*	scale factor in both x-direction and y-direction

11.8.3.75 fl_scale() [2/2]

```
void fl_scale (
            double x,
            double y )  [inline]
```

Concatenate scaling transformation onto the current one.

Parameters

in	*x,y*	scale factors in x-direction and y-direction

11.8.3.76 fl_scroll()

```
void fl_scroll (
            int X,
            int Y,
            int W,
            int H,
            int dx,
            int dy,
            void(*)(void *, int, int, int, int) draw_area,
            void * data )
```

Scroll a rectangle and draw the newly exposed portions.

Parameters

in	*X,Y*	position of top-left of rectangle
in	*W,H*	size of rectangle
in	*dx,dy*	pixel offsets for shifting rectangle
in	*draw_area*	callback function to draw rectangular areas
in	*data*	pointer to user data for callback The contents of the rectangular area is first shifted by `dx` and `dy` pixels. The `draw_area` callback is then called for every newly exposed rectangular area.

Warning

With FLTK 1.4 and above, it's recommended to put graphical elements in an Fl_Scroll widget rather than use function fl_scroll() to update those elements. That's because fl_scroll() may not produce a pixel-accurate result when the GUI scaling factor value is not a multiple of 100%. Alternatively, use fl_scroll() when the scaling factor value is a multiple of 100% and perform a full redraw of the graphical elements otherwise. Statement `float s = Fl::screen_scale(win->screen_num());` gives the current scaling factor value given Fl_Window *win.

11.8.3.77 fl_set_spot()

```
void fl_set_spot (
            int font,
            int size,
            int X,
            int Y,
            int W,
            int H,
            Fl_Window * win = 0 )
```

Inform text input methods about the current text insertion cursor.

Parameters

font	Font currently in use in text input.
size	Size of the current font.
X,Y	Position of the bottom of the current text insertion cursor.
W,H	Width and height of the current text insertion cursor.
win	Points to the Fl_Window object containing the current text widget, or NULL.

11.8.3.78 fl_set_status()

```
void fl_set_status (
            int X,
            int Y,
            int W,
            int H )
```

Related to text input methods under X11.

This function is presently used only by the `utf8` test application and only for the X11 platform. This function is apparently not indispensable for text input to work correctly as suggested by other apps that don't use it (e.g., editor).

11.8.3.79 fl_shortcut_label() [1/2]

```
const char * fl_shortcut_label (
            unsigned int shortcut )
```

Get a human-readable string from a shortcut value.
Unparse a shortcut value as used by Fl_Button or Fl_Menu_Item into a human-readable string like "Alt+N". This only works if the shortcut is a character key or a numbered function key. If the shortcut is zero then an empty string is returned. The return value points at a static buffer that is overwritten with each call.

Since

FLTK 1.3.4 modifier key names can be localized, but key names can not yet be localized. This may be added to a future FLTK version.

Modifier key names (human-readable shortcut names) can be defined with the following global const char * pointer variables:

- fl_local_ctrl => name of FL_CTRL
- fl_local_alt => name of FL_ALT
- fl_local_shift => name of FL_SHIFT
- fl_local_meta => name of FL_META

```
fl_local_ctrl = "Strg";      // German for "Ctrl"
fl_local_shift = "Umschalt"; // German for "Shift"
```

Note

Due to **random** static initialization order this should always be done from code in main() or called by main() as opposed to static initialization since the default strings in the FLTK library are set by static initializers. Otherwise this **might** result in the wrong order so FLTK's internal initialization overwrites your strings.

The shortcut name will be constructed by adding all modifier names in the order defined above plus the name of the key. A '+' character is added to each modifier name unless it has a trailing '\' or a trailing '+'.
Example:
Ctrl+Alt+Shift+Meta+F12
The default values for modifier key names are as given above for all platforms except macOS. macOS uses graphical characters that represent the typical macOS modifier names in menus, e.g. cloverleaf, saucepan, etc. You may, however, redefine macOS modifier names as well.

Parameters

in	*shortcut*	the integer value containing the ASCII character or extended keystroke plus modifiers

Returns

a pointer to a static buffer containing human readable text for the shortcut

11.8.3.80 fl_shortcut_label() [2/2]

```
const char * fl_shortcut_label (
            unsigned int shortcut,
            const char ** eom )
```

Get a human-readable string from a shortcut value.

Parameters

in	*shortcut*	the integer value containing the ASCII character or extended keystroke plus modifiers
in	*eom*	if this pointer is set, it will receive a pointer to the end of the modifier text

Returns

a pointer to a static buffer containing human readable text for the shortcut

See also

fl_shortcut_label(unsigned int shortcut)

11.8.3.81 fl_transform_dx()

```
double fl_transform_dx (
            double x,
            double y )  [inline]
```

Transform distance using current transformation matrix.

Parameters

in	*x,y*	coordinate

11.8.3.82 fl_transform_dy()

```
double fl_transform_dy (
            double x,
            double y )  [inline]
```

Transform distance using current transformation matrix.

Parameters

in	*x,y*	coordinate

11.8.3.83 fl_transform_x()

```
double fl_transform_x (
            double x,
            double y )  [inline]
```

Transform coordinate using the current transformation matrix.

Parameters

in	*x,y*	coordinate

11.8.3.84 fl_transform_y()

```
double fl_transform_y (
            double x,
            double y )  [inline]
```

Transform coordinate using the current transformation matrix.

Parameters

in	*x,y*	coordinate

11.8.3.85 fl_transformed_vertex()

```
void fl_transformed_vertex (
            double xf,
            double yf )  [inline]
```

Add coordinate pair to the vertex list without further transformations.

Parameters

in	*xf,yf*	transformed coordinate

11.8.3.86 fl_translate()

```
void fl_translate (
            double x,
            double y )  [inline]
```

Concatenate translation transformation onto the current one.

Parameters

in	*x,y*	translation factor in x-direction and y-direction

11.8.3.87 fl_vertex()

```
void fl_vertex (
            double x,
            double y )  [inline]
```

Add a single vertex to the current path.

Parameters

in	*x,y*	coordinate

11.9 Multithreading support functions

fl multithreading support functions declared in <FL/Fl.H>

Functions

- void Fl::awake ()

  Notifies the main GUI thread from a worker thread.
- int Fl::awake (Fl_Awake_Handler handler, void ∗user_data=nullptr)

  Schedules a callback to be executed by the main thread, then wakes up the main thread.
- void Fl::awake (void ∗message)

  Awake the main GUI thread and leave a message pointer.
- int Fl::awake_once (Fl_Awake_Handler handler, void ∗user_data=nullptr)

  Schedules a callback to be executed once by the main thread, then wakes up the main thread.
- int Fl::lock ()

  Acquire the global UI lock for FLTK.
- void ∗ Fl::thread_message ()

  Returns the last message sent by a child thread.
- void Fl::unlock ()

  Release the global UI lock set by Fl::lock().

11.9.1 Detailed Description

fl multithreading support functions declared in <FL/Fl.H>

11.9.2 Function Documentation

11.9.2.1 awake() [1/3]

```
void Fl::awake ( ) [extern]
```

Notifies the main GUI thread from a worker thread.
In FLTK, worker threads can update the UI, but all UI changes must be wrapped between Fl::lock() and Fl::unlock(). After calling Fl::unlock(), the worker thread should call Fl::awake() to signal the main thread that updates are pending.

Note

Worker threads must not create, show, or hide windows.

See also

Multithreading

Fl::awake(Fl_Awake_Handler, void*)

Fl::awake_once(Fl_Awake_Handler, void*)

11.9.2.2 awake() [2/3]

```
int Fl::awake (
            Fl_Awake_Handler handler,
            void * user_data = nullptr ) [extern]
```

Schedules a callback to be executed by the main thread, then wakes up the main thread.
This function lets a worker thread request that a specific callback function be run by the main thread, passing optional user data. The callback will be executed during the main thread's next event handling cycle.
The queue holding the list of handlers is limited to 1024 entries. If the queue is full, the function will return -1 and the callback will not be scheduled. However the main thread will still be woken up to process any other pending events.

Note

If user_data points to dynamically allocated memory, it is the responsibility of the caller to ensure that the memory is valid until the callback is executed. The callback will be executed during the main thread's next event handling cycle, but depending on the sytems load, this may take several seconds.

Returns

0 if the callback was successfully scheduled

-1 if the queue is full.

See also

Fl::awake()

Fl::awake_once(Fl_Awake_Handler, void*)

Multithreading

11.9.2.3 awake() [3/3]

```
void Fl::awake (
            void * v ) [extern]
```

Awake the main GUI thread and leave a message pointer.

Deprecated "since 1.5.0 - use Fl::awake() or Fl::awake(handler, user_data) instead"

Deprecated Use Fl::awake() or Fl::awake(Fl_Awake_Handler, void∗) instead.

This method is deprecated. The API can not ensure that Fl::thread_message() returns the messages sent by Fl::awake(void ∗v) complete and in the correct order.
Use Fl::awake() instead if you do not need to send a specific message. Use Fl::awake(Fl_Awake_Handler, void∗) or Fl::awake_once(Fl_Awake_Handler, void∗) if you need to send a message to the main thread and ensure that all messages are processed in the order they were sent.

See also

Multithreading

Fl::awake()

Fl::awake(Fl_Awake_Handler, void∗)

Fl::awake_once(Fl_Awake_Handler, void∗)

11.9.2.4 awake_once()

```
int Fl::awake_once (
            Fl_Awake_Handler handler,
            void * user_data = nullptr )  [extern]
```

Schedules a callback to be executed once by the main thread, then wakes up the main thread.
This function lets a worker thread request that a specific callback function be run by the main thread, passing optional user data. If a callback with the same user_data is already scheduled, the previous entry will be removed and the new entry will be appended to the list.

Returns

0 if the callback was successfully scheduled

-1 if the queue is full.

See also

Fl::awake()

Fl::awake(Fl_Awake_Handler, void∗)

Multithreading

11.9.2.5 lock()

```
int Fl::lock ( )  [extern]
```

Acquire the global UI lock for FLTK.
The lock() method blocks the current thread until it can safely access FLTK widgets and data. Child threads should call this method prior to updating any widgets or accessing data. The main thread must call Fl::lock() once before any windows are shown to initialize the threading support in FLTK. The initial Fl::lock() call will return non-zero if threading is not available on the platform.
Child threads enclose calls to FLTK functions between Fl::lock() and Fl::unlock() accessing FLTK. When a child thread has finshed accessing FLTK and wants the main thread to update the UI, it should call Fl::awake().
Child threads can never create, show, or hide windows.
When the wait() method is waiting for input or timeouts, child threads are given access to FLTK. Similarly, when the main thread needs to do processing, it will wait until all child threads have called unlock() before processing additional data.

Returns

0 if threading is available on the platform; non-zero otherwise.

See also

Multithreading

Fl::lock()

Fl::awake()

11.9.2.6 thread_message()

`void * Fl::thread_message ( ) [extern]`
Returns the last message sent by a child thread.

Deprecated "since 1.5.0 - use Fl::awake() or Fl::awake(handler, user_data) instead"

Deprecated Use Fl::awake(Fl_Awake_Handler, void∗) or Fl::awake_once(Fl_Awake_Handler, void∗) instead.

The thread_message() method returns the last message that was sent from a child by the Fl::awake(void∗) method. This method is deprecated. The API can not ensure that Fl::thread_message() returns the messages sent by Fl::awake(void ∗v) complete and in the correct order.

See also

Multithreading

Fl::awake()

Fl::awake(Fl_Awake_Handler, void∗)

Fl::awake_once(Fl_Awake_Handler, void∗)

11.9.2.7 unlock()

`void Fl::unlock ( ) [extern]`
Release the global UI lock set by Fl::lock().
The unlock() method releases the lock that was set using the lock() method. Child threads should call this method as soon as they are finished accessing FLTK.

See also

Multithreading

Fl::lock()

11.10 Cairo Support Functions and Classes

Classes

- class Fl_Cairo_State

  Contains all the necessary info on the current cairo context.
- class Fl_Cairo_Window

  This defines an FLTK window with Cairo support.

Functions

- bool Fl::cairo_autolink_context ()

  Gets the current autolink mode for Cairo support.
- void Fl::cairo_autolink_context (bool alink)

  When FLTK_HAVE_CAIRO is defined and cairo_autolink_context() is true, any current window dc is linked to a current Cairo context.
- cairo_t ∗ **Fl::cairo_cc** ()

  Gets the current Cairo context linked with a fltk window.
- void Fl::cairo_cc (cairo_t ∗c, bool own=false)

  Sets the current Cairo context to `c`.
- void Fl::cairo_flush (cairo_t ∗c)

  Flush Cairo drawings on Cairo context `c`.
- cairo_t ∗ Fl::cairo_make_current (Fl_Window ∗w)

  Provides a Cairo context for window wi.

11.10.1 Detailed Description

11.10.2 Function Documentation

11.10.2.1 cairo_autolink_context() [1/2]

```
bool Fl::cairo_autolink_context ( ) [extern]
```

Gets the current autolink mode for Cairo support.

Return values

false	if no Cairo context autolink is made for each window.
true	if any fltk window is attached a Cairo context when it is current.

See also

void cairo_autolink_context(bool alink)

Note

Only available if built with CMake option FLTK_OPTION_CAIRO_EXT=ON.

11.10.2.2 cairo_autolink_context() [2/2]

```
void Fl::cairo_autolink_context (
            bool alink ) [extern]
```

When FLTK_HAVE_CAIRO is defined and cairo_autolink_context() is true, any current window dc is linked to a current Cairo context.
This is not the default, because it may not be necessary to add Cairo support to all fltk supported windows. When you wish to associate a Cairo context in this mode, you need to call explicitly in your draw() overridden method, Fl::cairo_make_current(Fl_Window*). This will create a Cairo context only for this Window. Still in custom Cairo application it is possible to handle completely this process automatically by setting `alink` to true. In this last case, you don't need anymore to call Fl::cairo_make_current(). You can use Fl::cairo_cc() to get the current Cairo context anytime.

Note

Only available if built with CMake option FLTK_OPTION_CAIRO_WINDOW=ON.

11.10.2.3 cairo_cc()

```
void Fl::cairo_cc (
            cairo_t * c,
            bool own = false ) [extern]
```

Sets the current Cairo context to `c`.
Set `own` to true if you want fltk to handle this cc deletion.

Note

Only available if built with CMake option FLTK_OPTION_CAIRO_WINDOW=ON.

11.10.2.4 cairo_flush()

```
void Fl::cairo_flush (
            cairo_t * c ) [extern]
```

Flush Cairo drawings on Cairo context `c`.
This is **required** on Windows if you use the Cairo context provided by the "Cairo autolink" option. Call this when all your drawings on the Cairo context are finished. This is maybe not necessary on other platforms than Windows but it does no harm if you call it always.

You don't need to use this if you use an Fl_Cairo_Window which does this automatically after the draw callback returns.
Code example for "Cairo autolink" mode:
In the overridden draw() method of your subclass of Fl_Window or any widget:

```
cairo_t *cc = Fl::cairo_cc();   // get the "autolink" Cairo context
// ... your Cairo drawings are here ...
Fl::cairo_flush(cc);            // flush Cairo drawings to the device
```

If you configure FLTK with CMake option 'FLTK_OPTION_CAIRO_WINDOW' (i.e. without CMake option 'FLTK_OPTION_CAIRO_EXT') or if you don't enable the 'autolink' Cairo context you may do the equivalent to use Cairo drawings in an overridden draw() method of derived classes by using

```
// get the  Cairo context for the \c window
cairo_t *cc = Fl::cairo_make_current(window);
// ... your Cairo drawings are here ...
Fl::cairo_flush(cc); // flush Cairo drawings to the device
```

See also

Fl::cairo_autolink_context(bool)

Fl::cairo_make_current(Fl_Window*);

11.10.2.5 cairo_make_current()

```
cairo_t * Fl::cairo_make_current (
            Fl_Window * wi )  [extern]
```

Provides a Cairo context for window *wi*.
This is needed in a draw() override if Fl::cairo_autolink_context() returns false, which is the default. The cairo_context() does not need to be freed as it is freed every time a new Cairo context is created. When the program terminates, a call to Fl::cairo_make_current(0) will destroy any residual context.

Note

A new Cairo context is not always re-created when this method is used. In particular, if the current graphical context and the current window didn't change between two calls, the previous gc is internally kept, thus optimizing the drawing performances. Also, after this call, Fl::cairo_cc() is adequately updated with this Cairo context.

Only available when CMake option '-D FLTK_OPTION_CAIRO_WINDOW' is set.

Returns

The valid cairo_t *cairo context associated to this window.

Return values

NULL	if *wi* is NULL or maybe with GL windows under Wayland

11.11 Unicode and UTF-8 functions

fl global Unicode and UTF-8 handling functions declared in <FL/fl_utf8.h>

Macros

- #define ERRORS_TO_CP1252 1

  Set to 1 to turn bad UTF-8 bytes in the 0x80-0x9f range into the Unicode index for Microsoft's CP1252 character set.
- #define ERRORS_TO_ISO8859_1 1

  Set to 1 to turn bad UTF-8 bytes into ISO-8859-1.
- #define **NBC** 0xFFFF + 1
- #define STRICT_RFC3629 0

  A number of Unicode code points are in fact illegal and should not be produced by a UTF-8 converter.

Functions

- int fl_access (const char *f, int mode)

  Cross-platform function to test a files access() with a UTF-8 encoded name or value.
- int fl_chdir (const char *path)

  Cross-platform function to change the current working directory, given as a UTF-8 encoded string.
- int fl_chmod (const char *f, int mode)

  Cross-platform function to set a files mode() with a UTF-8 encoded name or value.
- int fl_close_fd (int fd)

  Cross-platform function to close a file descriptor.
- int fl_execvp (const char *file, char *const *argv)

  Calls `execvp` on all platforms.
- FILE * fl_fopen (const char *f, const char *mode)

  Cross-platform function to open files with a UTF-8 encoded name.
- char * fl_getcwd (char *buf, int len)

  Cross-platform function to get the current working directory as a UTF-8 encoded value.
- char * fl_getenv (const char *v)

  Cross-platform function to get environment variables with a UTF-8 encoded name or value.
- char fl_make_path (const char *path)

  Cross-platform function to recursively create a path in the file system.
- void fl_make_path_for_file (const char *path)

  Cross-platform function to create a path for the file in the file system.
- int fl_mkdir (const char *f, int mode)

  Cross-platform function to create a directory with a UTF-8 encoded name.
- unsigned int fl_nonspacing (unsigned int ucs)

  Returns true if the Unicode character `ucs` is non-spacing.
- int fl_open (const char *fname, int oflags,...)

  Cross-platform function to open files with a UTF-8 encoded name.
- int fl_open_ext (const char *fname, int binary, int oflags,...)

  Cross-platform function to open files with a UTF-8 encoded name.
- int fl_putenv (const char *var)

  Cross-platform function to write environment variables with a UTF-8 encoded name or value.
- int fl_rename (const char *f, const char *n)

  Cross-platform function to rename a filesystem object using UTF-8 encoded names.
- int fl_rmdir (const char *f)

  Cross-platform function to remove a directory with a UTF-8 encoded name.
- int fl_stat (const char *f, struct stat *b)

  Cross-platform function to stat() a file using a UTF-8 encoded name or value.
- int fl_system (const char *cmd)

  Cross-platform function to run a system command with a UTF-8 encoded string.
- int fl_tolower (unsigned int ucs)

  Returns the Unicode lower case value of `ucs`.
- int fl_toupper (unsigned int ucs)

  Returns the Unicode upper case value of `ucs`.
- unsigned fl_ucs_to_Utf16 (const unsigned ucs, unsigned short *dst, const unsigned dstlen)

  Convert a single 32-bit Unicode codepoint into an array of 16-bit characters.
- int fl_unlink (const char *fname)

  Cross-platform function to unlink() (that is, delete) a file using a UTF-8 encoded filename.
- char * fl_utf2mbcs (const char *s)

  Converts UTF-8 string `s` to a local multi-byte character string.
- const char * fl_utf8_clean (const char *str, int *len)

Test and optionally re-encode a UTF-8 string (character sequence).

- int **fl_utf8_is_continuation** (char byte)

  Evaluates true if 'byte' is in the middle of a UTF-8 character (0x80..0xBF).
- const char * fl_utf8_next_composed_char (const char *from, const char *end)

  Returns pointer to beginning of character after given location in UTF8 string accounting for emoji sequences.
- const char * fl_utf8_previous_composed_char (const char *from, const char *begin)

  Returns pointer to beginning of character before given location in UTF8 string accounting for emoji sequences.
- const char * fl_utf8back (const char *p, const char *start, const char *end)

  Move `p` backward until it points to the start of a UTF-8 character.
- int fl_utf8bytes (unsigned ucs)

  Returns number of bytes that utf8encode() will use to encode the character `ucs`.
- unsigned fl_utf8decode (const char *p, const char *end, int *len)

  Decode a single UTF-8 encoded character starting at p.
- int fl_utf8encode (unsigned ucs, char *buf)

  Write the UTF-8 encoding of ucs into buf and return the number of bytes written.
- unsigned fl_utf8from_mb (char *dst, unsigned dstlen, const char *src, unsigned srclen)

  Convert a filename from the locale-specific multibyte encoding used by Windows to UTF-8 as used by FLTK.
- unsigned fl_utf8froma (char *dst, unsigned dstlen, const char *src, unsigned srclen)

  Convert an ISO-8859-1 (ie normal c-string) byte stream to UTF-8.
- unsigned fl_utf8fromwc (char *dst, unsigned dstlen, const wchar_t *src, unsigned srclen)

  Turn "wide characters" as returned by some system calls (especially on Windows) into UTF-8.
- const char * fl_utf8fwd (const char *p, const char *start, const char *end)

  Move `p` forward until it points to the start of a UTF-8 character.
- int fl_utf8len (char c)

  Returns the byte length of the UTF-8 sequence, or -1 if `c` is not a valid leading byte.
- int fl_utf8len1 (char c)

  Returns the byte length of the UTF-8 sequence with first byte `c`, or 1 if `c` is not valid.
- int fl_utf8locale ()

  Return true if the "locale" seems to indicate that UTF-8 encoding is used.
- int fl_utf8strlen (const char *text, int len)

  Return the length in bytes of a UTF-8 string.
- int fl_utf8test (const char *src, unsigned srclen)

  Examines the first `srclen` bytes in `src` and returns a verdict on whether it is UTF-8 or not.
- unsigned fl_utf8to_mb (const char *src, unsigned srclen, char *dst, unsigned dstlen)

  Convert the UTF-8 used by FLTK to the locale-specific encoding used for filenames (and sometimes used for data in files).
- unsigned fl_utf8toa (const char *src, unsigned srclen, char *dst, unsigned dstlen)

  Convert a UTF-8 sequence into an array of 1-byte characters.
- unsigned fl_utf8toUtf16 (const char *src, unsigned srclen, unsigned short *dst, unsigned dstlen)

  Convert a UTF-8 sequence into an array of 16-bit characters.
- unsigned fl_utf8towc (const char *src, unsigned srclen, wchar_t *dst, unsigned dstlen)

  Converts a UTF-8 string into a wide character string.
- int fl_utf_nb_char (const unsigned char *buf, int len)

  Returns the number of Unicode chars in the UTF-8 string.
- int fl_utf_strcasecmp (const char *s1, const char *s2)

  UTF-8 aware strcasecmp - converts to Unicode and tests.
- int fl_utf_strncasecmp (const char *s1, const char *s2, int n)

  UTF-8 aware strncasecmp - converts to lower case Unicode and tests.
- int fl_utf_tolower (const unsigned char *str, int len, char *buf)

  Converts the string `str` to its lower case equivalent into buf.
- int fl_utf_toupper (const unsigned char *str, int len, char *buf)

Converts the string `str` to its upper case equivalent into buf.

- int fl_wcwidth (const char *src)

  extended wrapper around fl_wcwidth_(unsigned int ucs) function.

- int fl_wcwidth_ (unsigned int ucs)

  Wrapper to adapt Markus Kuhn's implementation of wcwidth() for FLTK.

11.11.1 Detailed Description

fl global Unicode and UTF-8 handling functions declared in <FL/fl_utf8.h>

11.11.2 Macro Definition Documentation

11.11.2.1 ERRORS_TO_CP1252

```
#define ERRORS_TO_CP1252 1
```

Set to 1 to turn bad UTF-8 bytes in the 0x80-0x9f range into the Unicode index for Microsoft's CP1252 character set.
You should also set ERRORS_TO_ISO8859_1. With this a huge amount of more available text (such as all web pages) are correctly converted to Unicode.

11.11.2.2 ERRORS_TO_ISO8859_1

```
#define ERRORS_TO_ISO8859_1 1
```

Set to 1 to turn bad UTF-8 bytes into ISO-8859-1.
If this is zero they are instead turned into the Unicode REPLACEMENT CHARACTER, of value 0xfffd. If this is on fl_utf8decode() will correctly map most (perhaps all) human-readable text that is in ISO-8859-1. This may allow you to completely ignore character sets in your code because virtually everything is either ISO-8859-1 or UTF-8.

11.11.2.3 STRICT_RFC3629

```
#define STRICT_RFC3629 0
```

A number of Unicode code points are in fact illegal and should not be produced by a UTF-8 converter.
Turn this on will replace the bytes in those encodings with errors. If you do this then converting arbitrary 16-bit data to UTF-8 and then back is not an identity, which will probably break a lot of software.

11.11.3 Function Documentation

11.11.3.1 fl_access()

```
int fl_access (
            const char * f,
            int mode )
```

Cross-platform function to test a files access() with a UTF-8 encoded name or value.
This function is especially useful on the Windows platform where the standard access() function fails with UTF-8 encoded non-ASCII filenames.
Windows defines the mode values 0 for existence, 2 for writable, 4 for readable, and 6 of readable and writable. On other systems, the modes `X_OK`, `W_OK`, and `R_OK` are usually defined as 1, 2, and 4.
Upon successful completion, the value 0 is returned on all platforms.

Parameters

in	*f*	the UTF-8 encoded filename
in	*mode*	the mode to test

Returns

the return value of _waccess() on Windows or access() on other platforms.

11.11.3.2 fl_chdir()

```
int fl_chdir (
            const char * path )
```

Cross-platform function to change the current working directory, given as a UTF-8 encoded string.
This function is especially useful on the Windows platform where the standard _wchdir() function needs a `path` in UTF-16 encoding.
The `path` is converted to a system specific encoding if necessary and the system specific `chdir(converted_path)` function is called.
The function returns 0 on success and -1 on error. Depending on the platform, `errno` **may** be set if an error occurs.

Note

The possible errno values are platform specific. Refer to the documentation of the platform specific chdir() function.

If the function is not implemented on a particular platform the default implementation returns -1 and `errno` is **not** set.
If the `path` is `NULL` the function returns -1, but `errno` is **not** changed. This is a convenience feature of fl_chdir() as opposed to chdir().

Parameters

in	*path*	the target directory for chdir (may be `NULL`)

Returns

0 if successful, -1 on error (errno may be set)

11.11.3.3 fl_chmod()

```
int fl_chmod (
            const char * f,
            int mode )
```

Cross-platform function to set a files mode() with a UTF-8 encoded name or value.
This function is especially useful on the Windows platform where the standard chmod() function fails with UTF-8 encoded non-ASCII filenames.

Parameters

in	*f*	the UTF-8 encoded filename
in	*mode*	the mode to set

Returns

the return value of _wchmod() on Windows or chmod() on other platforms.

11.11.3.4 fl_close_fd()

```
int fl_close_fd (
            int fd )
```

Cross-platform function to close a file descriptor.

Returns

0 in case of success, or -1 in case of error.

11.11.3.5 fl_execvp()

```
int fl_execvp (
            const char * file,
            char *const * argv )
```

Calls `execvp` on all platforms.
On Windows, this converts the file and all arguments in argv into Windows multibyte characters and calls _↩ `wexecvp`.

Parameters

in	*file*	path to the new process
in	*argv*	array of command line arguments, last array member must be a `nullptr`.

Returns

the result of `execvp` call.

11.11.3.6 fl_fopen()

```
FILE * fl_fopen (
            const char * f,
            const char * mode )
```

Cross-platform function to open files with a UTF-8 encoded name.
This function is especially useful on the Windows platform where the standard fopen() function fails with UTF-8 encoded non-ASCII filenames.

```
#include <FL/fl_utf8.h>

void test() {
  FILE *f = fl_fopen("¡Peligro!", "rw");
  if (f) fl_fclose(f);
}
```

Parameters

f	the UTF-8 encoded filename
mode	same as the second argument of the standard fopen() function

Returns

a FILE pointer upon successful completion, or NULL in case of error.

See also

fl_open().

11.11.3.7 fl_getcwd()

```
char * fl_getcwd (
            char * buf,
            int len )
```

Cross-platform function to get the current working directory as a UTF-8 encoded value.
This function is especially useful on the Windows platform where the standard _wgetcwd() function returns UTF-16 encoded non-ASCII filenames.
If `buf` is `NULL` a buffer of size (len+1) is allocated, filled with the current working directory, and returned. In this case the buffer must be released by the caller with free() to prevent memory leaks.

Parameters

in	*buf*	the buffer to populate (may be NULL)
in	*len*	the length of the buffer

Returns

the CWD encoded as UTF-8

11.11.3.8 fl_getenv()

```
char * fl_getenv (
            const char * v )
```

Cross-platform function to get environment variables with a UTF-8 encoded name or value.
This function is especially useful on the Windows platform where non-ASCII environment variables are encoded as wide characters. The returned value of the variable is encoded in UTF-8 as well.
On platforms other than Windows this function calls getenv directly. The return value is returned as-is.
The return value is a pointer to an implementation defined buffer:

- an internal buffer that is (re)allocated as needed (Windows) or
- the string in the environment itself (Unix, Linux, MaOS) or
- any other implementation (other platforms). This string must be considered read-only and must not be freed by the caller.

If the resultant string is to be used later it must be copied to a safe place. The next call to fl_getenv() or any other environment changes may overwrite the string.
Example:

```
#include <FL/fl_utf8.h>

fl_putenv("cœur=frère")
return fl_getenv("cœur");  // return UTF-8 value on all platforms
```

Note

This function is not thread-safe.

Parameters

in	*v*	the UTF-8 encoded environment variable

Returns

the environment variable in UTF-8 encoding, or NULL in case of error.

See also

fl_putenv(const char* var)

11.11.3.9 fl_make_path()

```
char fl_make_path (
            const char * path )
```

Cross-platform function to recursively create a path in the file system.
This function creates a `path` in the file system by recursively creating all directories.

Parameters

in	*path*	a Unix style ('/' forward slashes) absolute or relative pathname

Returns

1 if the path was created, 0 if creating the path failed at some point

11.11.3.10 fl_make_path_for_file()

```
void fl_make_path_for_file (
            const char * path )
```

Cross-platform function to create a path for the file in the file system.
This function strips the filename from the given `path` and creates a path in the file system by recursively creating all directories.

Parameters

in	*path*	path to a file, ending in a file name. The separator between the path and the filename must be the forward slash.

11.11.3.11 fl_mkdir()

```
int fl_mkdir (
            const char * f,
            int mode )
```

Cross-platform function to create a directory with a UTF-8 encoded name.
This function is especially useful on the Windows platform where the standard _wmkdir() function expects UTF-16 encoded non-ASCII filenames.

Parameters

in	*f*	the UTF-8 encoded filename
in	*mode*	the mode of the directory

Returns

the return value of _wmkdir() on Windows or mkdir() on other platforms.

11.11.3.12 fl_nonspacing()

```
unsigned int fl_nonspacing (
            unsigned int ucs )
```

Returns true if the Unicode character `ucs` is non-spacing.
Non-spacing characters in Unicode are typically combining marks like tilde (~), diaeresis (¨), or other marks that are added to a base character, for instance 'a' (base character) + '¨' (combining mark) = 'ä' (German Umlaut).

- http://unicode.org/glossary/#base_character
- http://unicode.org/glossary/#nonspacing_mark
- http://unicode.org/glossary/#combining_character

Parameters

in	*ucs*	32 bit Unicode character

Returns

0 if this is a spacing character, or any other value if not

Note

This function has not been verified. It's not working for ASCII characters

11.11.3.13 fl_open()

```
int fl_open (
            const char * fname,
            int oflags,
             ...  )
```

Cross-platform function to open files with a UTF-8 encoded name.
This function is especially useful on the Windows platform where the standard open() function fails with UTF-8 encoded non-ASCII filenames.

```
#include <FL/fl_utf8.h>

void test() {
  int fd = fl_open("¡Peligro!", O_RDWR);
  if (fd != -1) fl_close_fd(fd);
}
```

Parameters

in	*fname*	the UTF-8 encoded filename
in	*oflags*	other arguments are as in the standard open() function

Returns

a file descriptor upon successful completion, or -1 in case of error.

See also

fl_fopen(), fl_open_ext(), fl_close_fd(int fd).

11.11.3.14 fl_open_ext()

```
int fl_open_ext (
            const char * fname,
            int binary,
            int oflags,
             ...  )
```

Cross-platform function to open files with a UTF-8 encoded name.
In comparison with fl_open(), this function allows to control whether the file is opened in binary (a.k.a. untranslated) mode. This is especially useful on the Windows platform where files are by default opened in text (translated) mode.

Parameters

in	*fname*	the UTF-8 encoded filename
in	*binary*	if non-zero, the file is to be accessed in binary (a.k.a. untranslated) mode.
in	*oflags,...*	these arguments are as in the standard open() function. Setting `oflags` to zero opens the file for reading.

Returns

a file descriptor upon successful completion, or -1 in case of error.

See also

fl_open(), fl_fopen()

11.11.3.15 fl_putenv()

```
int fl_putenv (
            const char * var )
```

Cross-platform function to write environment variables with a UTF-8 encoded name or value.
This function is especially useful on the Windows platform where non-ASCII environment variables are encoded as wide characters.
The given argument `var` must be encoded in UTF-8 in the form "name=value". The `'name'` part must conform to platform dependent restrictions on environment variable names.
The string given in `var` is copied and optionally converted to the required encoding for the platform. On platforms other than Windows this function calls putenv directly.
The return value is zero on success and non-zero in case of error. The value in case of error is platform specific and returned as-is.

Note

The copied string is allocated on the heap and "lost" on some platforms, i.e. calling fl_putenv() to change environment variables frequently may cause memory leaks. There may be an option to avoid this in a future implementation.

This function is not thread-safe.

Parameters

in	*var*	the UTF-8 encoded environment variable `'name=value'`

Returns

0 on success, non-zero in case of error.

See also

fl_getenv(const char* var)

11.11.3.16 fl_rename()

```
int fl_rename (
            const char * f,
            const char * n )
```

Cross-platform function to rename a filesystem object using UTF-8 encoded names.
This function is especially useful on the Windows platform where the standard _wrename() function expects UTF-16 encoded non-ASCII filenames.

Parameters

in	*f*	the UTF-8 encoded filename to change
in	*n*	the new UTF-8 encoded filename to set

Returns

the return value of _wrename() on Windows or rename() on other platforms.

11.11.3.17 fl_rmdir()

```
int fl_rmdir (
            const char * f )
```

Cross-platform function to remove a directory with a UTF-8 encoded name.
This function is especially useful on the Windows platform where the standard _wrmdir() function expects UTF-16 encoded non-ASCII filenames.

Parameters

in	*f*	the UTF-8 encoded filename to remove

Returns

the return value of _wrmdir() on Windows or rmdir() on other platforms.

11.11.3.18 fl_stat()

```
int fl_stat (
            const char * f,
            struct stat * b )
```

Cross-platform function to stat() a file using a UTF-8 encoded name or value.
This function is especially useful on the Windows platform where the standard stat() function fails with UTF-8 encoded non-ASCII filenames.

Parameters

in	*f*	the UTF-8 encoded filename
out	*b*	the stat struct to populate

Returns

the return value of _wstat() on Windows or stat() on other platforms.

Note

the contents of `struct stat` is returned unchanged in the host format.

11.11.3.19 fl_system()

```
int fl_system (
            const char * cmd )
```

Cross-platform function to run a system command with a UTF-8 encoded string.
This function is especially useful on the Windows platform where non-ASCII program (file) names must be encoded as wide characters.
On platforms other than Windows this function calls system() directly.

```
#include <FL/fl_utf8.h>

fl_system("echo \"Hauptstraße 2\"""");
```

Parameters

in	*cmd*	the UTF-8 encoded command string

Returns

the return value of _wsystem() on Windows or system() on other platforms.

11.11.3.20 fl_tolower()

```
int fl_tolower (
            unsigned int ucs )
```

Returns the Unicode lower case value of `ucs`.
This is a relatively naive algorithm, good enough for basic Unicode string handling. If the conversion code can't call a system native function, FLTK will use lookup tables that are limited to the character range from 0x0 to 0xffff. Characters with a higher value are not converted.

Perfect per-language character handling is a project larger than FLTK itself.

Parameters

in	*ucs*	Unicode 32 bit character

Returns

lower case version of that character in Unicode 32, or the same code if there is no lower case version.

See also

fl_toupper(unsigned int ucs), fl_utf8decode(), fl_utf8encode()

11.11.3.21 fl_toupper()

```
int fl_toupper (
            unsigned int ucs )
```

Returns the Unicode upper case value of `ucs`.

Parameters

in	*ucs*	Unicode 32 bit character

Returns

upper case version of that character in Unicode 32, or the same code if there is no upper case version.

See also

fl_tolower(unsigned int ucs), fl_utf8decode(), fl_utf8encode()

11.11.3.22 fl_ucs_to_Utf16()

```
unsigned fl_ucs_to_Utf16 (
            const unsigned ucs,
            unsigned short * dst,
            const unsigned dstlen )
```

Convert a single 32-bit Unicode codepoint into an array of 16-bit characters.
These are used by some system calls, especially on Windows.
Unicode characters in the range 0x10000 to 0x10ffff are converted to "surrogate pairs" which take two words each (in UTF-16 encoding). Typically, setting `dstlen` to 2 will ensure that any valid Unicode value can be converted, and setting `dstlen` to 3 or more will allow a NULL terminated sequence to be returned.

Parameters

in	*ucs*	is the value to convert.
out	*dst*	points at an array to write, and
in	*dstlen*	is the number of locations in this array. At most `dstlen` words will be written, and a 0 terminating word will be added if `dstlen` is large enough. Thus this function will never overwrite the buffer and will attempt return a zero-terminated string if space permits. If `dstlen` is zero then `dst` can be set to NULL and no data is written, but the length is returned.

Returns

The return value is the number of 16-bit words that *would* be written to `dst` if it is large enough, not counting any terminating zero. If the return value is greater than `dstlen` it indicates truncation, you should then allocate a new array of size return+1 and call this again.

11.11.3.23 fl_unlink()

```
int fl_unlink (
            const char * fname )
```

Cross-platform function to unlink() (that is, delete) a file using a UTF-8 encoded filename.
This function is especially useful on the Windows platform where the standard function expects UTF-16 encoded non-ASCII filenames.

Parameters

fname	the filename to unlink

Returns

the return value of _wunlink() on Windows or unlink() on other platforms.

11.11.3.24 fl_utf2mbcs()

```
char * fl_utf2mbcs (
            const char * s )
```

Converts UTF-8 string `s` to a local multi-byte character string.
On Linux and macOS, this returns the same string. On Windows machines, this function will convert the string to UTF-16, and the to the current encoding using `wcstombs()`.

Parameters

in	*s*	a UTF-8 encode string

Returns

a string that matches the current locale of the host machine. This is a pointer to an internal buffer that changes size as needed.

Note

Do not deallocate the returned pointer. This function is not thread safe.

11.11.3.25 fl_utf8_clean()

```
const char * fl_utf8_clean (
            const char * str,
            int * len )
```

Test and optionally re-encode a UTF-8 string (character sequence).
Scans the input string `str` with fl_utf8decode() that, by default, accepts also non-UTF-8 and processes it as if encoded in CP-1252. Returns a true UTF-8 string and its length, possibly transformed from CP-1252. If the input string is true UTF-8, the returned string is the same pointer as input. Otherwise, the returned string is in private memory allocated by fl_utf8_clean() and extended when necessary.
The input string `str` (UTF-8) need not be nul-terminated.
The ouput string (return value) is nul-terminated if

- the original string was valid and nul-terminated (not changed), or

- the original string contained invalid UTF-8 characters and was re-encoded. Its length is returned in `len`.

The purpose of this method is to return a clean, UTF-8 encoded string that can be used for system output functions like Pango text formatting and output.

Note

The output string (buffer address and size) is only valid until this function is subsequently called or the input string buffer is modified, whichever is returned by this function. See above.

Parameters

`in`	*str*	UTF-8 input sequence (may contain invalid encodings)
`in,out`	*len*	length of the input and output UTF-8 sequences, resp.

Returns

clean UTF-8 string (may not be nul-terminated) and its `len`

11.11.3.26 fl_utf8_next_composed_char()

```
const char * fl_utf8_next_composed_char (
            const char * from,
            const char * end )
```

Returns pointer to beginning of character after given location in UTF8 string accounting for emoji sequences.
While Unicode encodes most characters as a single codepoint, some emojis (examples: "woman pilot", "↩ San Marino flag", 9 "keycap 9") are encoded via an **emoji sequence**, that is, they are represented by sequences of consecutive Unicode points. An emoji sequence may pair two codepoints with "zero-width joiner" and may qualify any component with "variation selectors" or "Fitzpatrick emoji modifiers". Most flag emojis are encoded with two consecutive "regional indicator symbols". Keycap emojis are encoded with key + "emoji variation selector" + "combining enclosing keycap".
Use this function to advance to the next character within a UTF8 string processing an entire emoji sequence if present as a single character.

Parameters

from	points to a location within a UTF8 string. If this location is inside the UTF8 encoding of a codepoint or is an invalid byte, this function returns `from` + 1.
end	points past last codepoint of the string.

Returns

pointer to beginning of first codepoint after character, possibly an emoji sequence, that begins at `from`.

11.11.3.27 fl_utf8_previous_composed_char()

```
const char * fl_utf8_previous_composed_char (
            const char * from,
            const char * begin )
```

Returns pointer to beginning of character before given location in UTF8 string accounting for emoji sequences.
See fl_utf8_next_composed_char() for a hint about what is an emoji sequence. Use this function to step back to the previous character within a UTF8 string processing an entire emoji sequence if present as a single character. The string is assumed to be NULL-terminated. The returned value cannot be lower than `begin`.

Parameters

from	points to a location within a UTF8 string or to terminal null byte. If this location is inside the UTF8 encoding of a codepoint or is an invalid byte, this function returns `from` - 1.
begin	points to start of first codepoint of the string.

Returns

pointer to beginning of first character, possibly an emoji sequence, before the codepoint that begins at `from`.

11.11.3.28 fl_utf8back()

```
const char * fl_utf8back (
            const char * p,
            const char * start,
            const char * end )
```

Move `p` backward until it points to the start of a UTF-8 character.
If it already points at the start of one then it is returned unchanged. Any UTF-8 errors are treated as though each byte of the error is an individual character.

Parameters

in	*p*	points somewhere into a UTF-8 encoded string, need not be on a UTF-8 sequence start or end. If you wish to decrement a UTF-8 pointer, pass p-1 to this.
in	*start*	is the start of the string and is used to limit the backwards search for the start of a UTF-8 character.
in	*end*	is the end of the string and is assumed to be a break between characters. It is assumed to be greater than p.

Returns

pointer to the start of a UTF-8 sequence.

11.11.3.29 fl_utf8bytes()

```
int fl_utf8bytes (
            unsigned ucs )
```

Returns number of bytes that utf8encode() will use to encode the character `ucs`.

Parameters

in	*ucs*	32-bit Unicode character

Returns

number of bytes for UTF-8 encoded sequence.

11.11.3.30 fl_utf8decode()

```
unsigned fl_utf8decode (
            const char * p,
            const char * end,
            int * len )
```

Decode a single UTF-8 encoded character starting at *p*.
The resulting Unicode value (in the range 0-0x10ffff) is returned, and *len* is set to the number of bytes in the UTF-8 encoding (adding *len* to *p* will point at the next character).

If `p` points at an illegal UTF-8 encoding, including one that would go past *end*, or where a code uses more bytes than necessary, then *(unsigned char*)p is translated as though it is in the Microsoft CP1252 character set and *len* is set to 1. Treating errors this way allows this to decode almost any ISO-8859-1 or CP1252 text that has been mistakenly placed where UTF-8 is expected, and has proven very useful.
If you want errors to be converted to error characters (as the standards recommend), adding a test to see if the length is unexpectedly 1 will work:

```
if (*p & 0x80) {              // what should be a multibyte encoding
  code = fl_utf8decode(p,end,&len);
  if (len<2) code = 0xFFFD;   // Turn errors into REPLACEMENT CHARACTER
} else {                      // handle the 1-byte UTF-8 encoding:
  code = *p;
  len = 1;
}
```

Direct testing for the 1-byte case (as shown above) will also speed up the scanning of strings where the majority of characters are ASCII.

Parameters

in	*p*	pointer to a UTF-8 encoded character
in	*end*	if set, points after the last character that may be read
out	*len*	if set, returns the length of the input UTF-8 sequence

Returns

32 bit Unicode character, or Unicode REPLACEMENT CHARACTER

11.11.3.31 fl_utf8encode()

```
int fl_utf8encode (
            unsigned ucs,
            char * buf )
```

Write the UTF-8 encoding of *ucs* into *buf* and return the number of bytes written.
Up to 4 bytes may be written. If you know that `ucs` is less than 0x10000 then at most 3 bytes will be written. If you wish to speed this up, remember that anything less than 0x80 is written as a single byte.
If ucs is greater than 0x10ffff this is an illegal character according to RFC 3629. These are converted as though they are 0xFFFD (REPLACEMENT CHARACTER).
RFC 3629 also says many other values for `ucs` are illegal (in the range 0xd800 to 0xdfff, or ending with 0xfffe or 0xffff). However I encode these as though they are legal, so that utf8encode/fl_utf8decode will be the identity for all codes between 0 and 0x10ffff.

Parameters

in	*ucs*	32-bit Unicode character
out	*buf*	a buffer of at least four bytes to receive the UTF-8 byte sequence. No terminating NUL is added.

Returns

number of bytes in UTF-8 sequence.

11.11.3.32 fl_utf8from_mb()

```
unsigned fl_utf8from_mb (
            char * dst,
            unsigned dstlen,
            const char * src,
            unsigned srclen )
```

Convert a filename from the locale-specific multibyte encoding used by Windows to UTF-8 as used by FLTK.
Up to `dstlen` bytes are written to `dst`, including a null terminator. The return value is the number of bytes that would be written, not counting the null terminator. If greater or equal to `dstlen` then if you malloc a new array of

size n+1 you will have the space needed for the entire string. If `dstlen` is zero then nothing is written and this call just measures the storage space needed.
On Unix or on Windows when a UTF-8 locale is in effect, this does not change the data. You may also want to check if fl_utf8test() returns non-zero, so that the filesystem can store filenames in UTF-8 encoding regardless of the locale.

11.11.3.33 fl_utf8froma()

```
unsigned fl_utf8froma (
            char * dst,
            unsigned dstlen,
            const char * src,
            unsigned srclen )
```

Convert an ISO-8859-1 (ie normal c-string) byte stream to UTF-8.
It is possible this should convert Microsoft's CP1252 to UTF-8 instead. This would translate the codes in the range 0x80-0x9f to different characters. Currently it does not do this.

Parameters

out	*dst*	Up to `dstlen` bytes are written to `dst`, including a null terminator. The return value is the number of bytes that would be written, not counting the null terminator. If greater or equal to...
in	*dstlen*	then if you malloc a new array of size n+1 you will have the space needed for the entire string. If `dstlen` is zero then nothing is written and this call just measures the storage space needed.
in	*src*	pointer to ISO-8859-1 string.
in	*srclen*	is the number of bytes in `src` to convert.

Returns

Number of bytes written. If the return value equals `srclen` then this indicates that no conversion is necessary, as only ASCII characters are in the string.

11.11.3.34 fl_utf8fromwc()

```
unsigned fl_utf8fromwc (
            char * dst,
            unsigned dstlen,
            const wchar_t * src,
            unsigned srclen )
```

Turn "wide characters" as returned by some system calls (especially on Windows) into UTF-8.
Up to `dstlen` bytes are written to `dst`, including a null terminator. The return value is the number of bytes that would be written, not counting the null terminator. If greater or equal to `dstlen` then if you malloc a new array of size n+1 you will have the space needed for the entire string. If `dstlen` is zero then nothing is written and this call just measures the storage space needed.
`srclen` is the number of words in `src` to convert. On Windows this is not necessarily the number of characters, due to there possibly being "surrogate pairs" in the UTF-16 encoding used. On Unix wchar_t is 32 bits and each location is a character.
On Unix if a `src` word is greater than 0x10ffff then this is an illegal character according to RFC 3629. These are converted as though they are 0xFFFD (REPLACEMENT CHARACTER). Characters in the range 0xd800 to 0xdfff, or ending with 0xfffe or 0xffff are also illegal according to RFC 3629. However I encode these as though they are legal, so that fl_utf8towc will return the original data.
On Windows "surrogate pairs" are converted to a single character and UTF-8 encoded (as 4 bytes). Mismatched halves of surrogate pairs are converted as though they are individual characters.

Parameters

out	*dst*	a destination buffer provided by the caller

Parameters

in	*dstlen*	size of dst buffer
in	*src*	pointer to Windows wide char string
in	*srclen*	number of characters to convert

Returns

number of bytes written, not including the terminating NUL

11.11.3.35 fl_utf8fwd()

```
const char * fl_utf8fwd (
            const char * p,
            const char * start,
            const char * end )
```

Move `p` forward until it points to the start of a UTF-8 character.
If it already points at the start of one then it is returned unchanged. Any UTF-8 errors are treated as though each byte of the error is an individual character.
This function is for moving a pointer that was jumped to the middle of a string, such as when doing a binary search for a position. You should use either this or fl_utf8back() depending on which direction your algorithm can handle the pointer moving. Do not use this to scan strings, use fl_utf8decode() instead.

Parameters

in	*p*	points somewhere into a UTF-8 encoded string, need not be on a UTF-8 sequence start or end.
in	*start*	is the start of the string and is used to limit the backwards search for the start of a UTF-8 character.
in	*end*	is the end of the string and is assumed to be a break between characters. It is assumed to be greater than p.

Returns

pointer to the start of a UTF-8 sequence or pointer to terminating NUL.

11.11.3.36 fl_utf8len()

```
int fl_utf8len (
            char c )
```

Returns the byte length of the UTF-8 sequence, or -1 if `c` is not a valid leading byte.
This function is helpful for identifying invalid UTF-8 leading bytes. It examines only `c`. It does not inspect continuation bytes or determine whether the complete UTF-8 sequence represents a valid Unicode code point.
ASCII bytes and valid modern UTF-8 leading bytes are recognized. Continuation bytes, obsolete leading bytes for 5- and 6-byte encodings, and other invalid leading bytes return -1.
Example:

```
#include <FL/fl_utf8.h>

bool test(const char *str) {
  if (str == nullptr) return true;
  const char *src = str;
  for (int p = 0; ; p++) {
    if (*src == 0) return true;
    int len = fl_utf8len(*src);
    if (len == -1) {
      printf("Invalid UTF-8 character start: 0x%02x\n", (unsigned char)*src);
      return false;
    } else {
      int tlen = len;
      while (tlen > 0) {
        if (*src == 0) {
          printf("Interrupted UTF-8 sequence at %d\n", (int)(src-str));
          return false;
```

```
        }
        src++;
        tlen--;
      }
      printf("Character %d at %d uses %d bytes\n", p, (int)(src-str-len), len);
    }
  }
}

// Examples of interest:
fl_utf8len('A');        // returns 1
fl_utf8len('\xC3');     // returns 2
fl_utf8len('\x80');     // returns -1: continuation byte
fl_utf8len('\xC0');     // returns -1: obsolete/invalid leading byte
fl_utf8len('\xF5');     // returns -1: outside the Unicode range
```

Parameters

in	*c*	the first byte of a potential UTF-8 sequence

Returns

the expected number of bytes in the sequence, or -1 if c is not a valid leading byte for UTF-8 style encoding.

See also

fl_utf8len1

11.11.3.37 fl_utf8len1()

```
int fl_utf8len1 (
            char c )
```

Returns the byte length of the UTF-8 sequence with first byte c, or 1 if c is not valid.
This function can be used to scan faulty UTF-8 sequence's leading bytes, albeit ignoring invalid codes.
This function has the same behavior as fl_utf8len(), except that it returns 1 rather than -1 for a byte that is not a recognized UTF-8 leading byte. This guarantees that a loop scanning possibly malformed input can always advance by at least one byte.
Example:

```
#include <FL/fl_utf8.h>

char mixed_string[] = "Hello \xFF\x80 world"; // Contains invalid UTF-8
char *p = mixed_string;

while (*p) {
  int len = fl_utf8len1(*p);
  printf("Byte 0x%02x uses %d byte(s)\n", (unsigned char)*p, len);
  p += len;  // Always advances by at least 1, even for invalid bytes
}
```

Parameters

in	*c*	the first byte of a potential UTF-8 sequence

Returns

the expected number of bytes in the sequence, or 1 if c is not a recognized UTF-8 leading byte, so a loop can continue to scan a string.

See also

fl_utf8len

11.11.3.38 fl_utf8locale()

```
int fl_utf8locale (
            void )
```

Return true if the "locale" seems to indicate that UTF-8 encoding is used.
If true the fl_utf8to_mb and fl_utf8from_mb don't do anything useful.

Note

It is highly recommended that you change your system so this does return true. On Windows this is done by setting the "codepage" to CP_UTF8. On Unix this is done by setting $LC_CTYPE to a string containing the letters "utf" or "UTF" in it, or by deleting all $LC∗ and $LANG environment variables. In the future it is likely that all non-Asian Unix systems will return true, due to the compatibility of UTF-8 with ISO-8859-1.

11.11.3.39 fl_utf8strlen()

```
int fl_utf8strlen (
            const char * text,
            int len )
```

Return the length in bytes of a UTF-8 string.
An illegal UTF-8 character is counted as 1.
Example:

```
#include <FL/fl_utf8.h>

return fl_utf8strlen("über", 1); // returns 2, the ü is encoded 0xC3, 0xBC
```

Parameters

in	*text*	encoded in UTF-8
in	*len*	number of Unicode characters, -1 to test until the end of text

Returns

number of bytes that make up the Unicode string

See also

fl_utf_nb_char(const unsigned char ∗buf, int len)

11.11.3.40 fl_utf8test()

```
int fl_utf8test (
            const char * src,
            unsigned srclen )
```

Examines the first `srclen` bytes in `src` and returns a verdict on whether it is UTF-8 or not.

- Returns 0 if there is any illegal UTF-8 sequences, using the same rules as fl_utf8decode(). Note that some UCS values considered illegal by RFC 3629, such as 0xffff, are considered legal by this.
- Returns 1 if there are only single-byte characters (ie no bytes have the high bit set). This is legal UTF-8, but also indicates plain ASCII. It also returns 1 if `srclen` is zero.
- Returns 2 if there are only characters less than 0x800.
- Returns 3 if there are only characters less than 0x10000.
- Returns 4 if there are characters in the 0x10000 to 0x10ffff range.

Because there are many illegal sequences in UTF-8, it is almost impossible for a string in another encoding to be confused with UTF-8. This is very useful for transitioning Unix to UTF-8 filenames, you can simply test each filename with this to decide if it is UTF-8 or in the locale encoding. My hope is that if this is done we will be able to cleanly transition to a locale-less encoding.

Parameters

in	*src*	pointer to string of unknown encoding
in	*srclen*	number of bytes to compare, must not be -1

Returns

0 if this is probably not a UTF-8 encode string

11.11.3.41 fl_utf8to_mb()

```
unsigned fl_utf8to_mb (
            const char * src,
            unsigned srclen,
            char * dst,
            unsigned dstlen )
```

Convert the UTF-8 used by FLTK to the locale-specific encoding used for filenames (and sometimes used for data in files).
Unfortunately due to stupid design you will have to do this as needed for filenames. This is a bug on both Unix and Windows.
Up to `dstlen` bytes are written to `dst`, including a null terminator. The return value is the number of bytes that would be written, not counting the null terminator. If greater or equal to `dstlen` then if you malloc a new array of size n+1 you will have the space needed for the entire string. If `dstlen` is zero then nothing is written and this call just measures the storage space needed.
If fl_utf8locale() returns true then this does not change the data.

11.11.3.42 fl_utf8toa()

```
unsigned fl_utf8toa (
            const char * src,
            unsigned srclen,
            char * dst,
            unsigned dstlen )
```

Convert a UTF-8 sequence into an array of 1-byte characters.
If the UTF-8 decodes to a character greater than 0xff then it is replaced with '?'.
Errors in the UTF-8 sequence are converted as individual bytes, same as fl_utf8decode() does. This allows ISO-8859-1 text mistakenly identified as UTF-8 to be printed correctly (and possibly CP1252 on Windows).

Parameters

in	*src*	points at the UTF-8 sequence, and
in	*srclen*	is the number of bytes to convert.
out	*dst*	Up to `dstlen` bytes are written to `dst`, including a null terminator. The return value is the number of bytes that would be written, not counting the null terminator. If greater or equal to...
in	*dstlen*	then if you malloc a new array of size n+1 you will have the space needed for the entire string. If `dstlen` is zero then nothing is written and this call just measures the storage space needed.

Returns

number of characters converted.

11.11.3.43 fl_utf8toUtf16()

```
unsigned fl_utf8toUtf16 (
            const char * src,
```

```
            unsigned srclen,
            unsigned short * dst,
            unsigned dstlen )
```

Convert a UTF-8 sequence into an array of 16-bit characters.
These are used by some system calls, especially on Windows.
Errors in the UTF-8 are converted as though each byte in the erroneous string is in the Microsoft CP1252 encoding. This allows ISO-8859-1 text mistakenly identified as UTF-8 to be printed correctly.
Unicode characters in the range 0x10000 to 0x10ffff are converted to "surrogate pairs" which take two words each (this is called UTF-16 encoding).

Parameters

in	*src*	points at the UTF-8, and
in	*srclen*	is the number of bytes to convert.
out	*dst*	points at an array to write, and
in	*dstlen*	is the number of locations in this array. At most `dstlen-1` words will be written there, plus a 0 terminating word. Thus this function will never overwrite the buffer and will always return a zero-terminated string. If `dstlen` is zero then `dst` can be null and no data is written, but the length is returned.

Returns

The return value is the number of 16-bit words that *would* be written to `dst` if it were long enough, not counting the terminating zero. If the return value is greater or equal to `dstlen` it indicates truncation, you can then allocate a new array of size return+1 and call this again.

11.11.3.44 fl_utf8towc()

```
unsigned fl_utf8towc (
            const char * src,
            unsigned srclen,
            wchar_t * dst,
            unsigned dstlen )
```

Converts a UTF-8 string into a wide character string.
This function generates 32-bit wchar_t (e.g. "ucs4" as it were) except on Windows where it is equivalent to fl_↩utf8toUtf16 and returns UTF-16.
Notice that sizeof(wchar_t) is 2 on Windows and is 4 on Linux and most other systems. Where wchar_t is 16 bits, Unicode characters in the range 0x10000 to 0x10ffff are converted to "surrogate pairs" which take two words each (this is called UTF-16 encoding). If wchar_t is 32 bits this rather nasty problem is avoided.

Note

Windows includes Cygwin, i.e. compiled with Cygwin's POSIX layer (cygwin1.dll, –enable-cygwin), either native (GDI) or X11.

Parameters

in	*src*	points at the UTF-8, and
in	*srclen*	is the number of bytes to convert.
out	*dst*	points at an array to write, and
in	*dstlen*	is the number of locations in this array. At most `dstlen-1` wchar_t will be written there, plus a 0 terminating wchar_t.

Returns

The return value is the number of wchar_t that *would* be written to `dst` if it were long enough, not counting the terminating zero. If the return value is greater or equal to `dstlen` it indicates truncation, you can then allocate a new array of size return+1 and call this again.

11.11.3.45 fl_utf_nb_char()

```
int fl_utf_nb_char (
            const unsigned char * buf,
            int len )
```

Returns the number of Unicode chars in the UTF-8 string.
An illegal UTF-8 character is counted as 1.
Example:

```
#include <FL/fl_utf8.h>

const char *str = "äöü"; // strlen(str) is 6
return fl_utf_nb_char(str, strlen(str)); // returns 3
```

Parameters

in	*buf*	text encoded in UTF-8
in	*len*	number of bytes to check. This code does not check for the trailing NUL character of "C" style strings. -1 is *not* valid here.

Returns

number of characters in that sequence. If a trailing UTF-8 sequence is not complete, the entire character is still counted.

See also

fl_utf8strlen(const char ∗text, int len)

11.11.3.46 fl_utf_strcasecmp()

```
int fl_utf_strcasecmp (
            const char * s1,
            const char * s2 )
```

UTF-8 aware strcasecmp - converts to Unicode and tests.

Returns

result of comparison

Return values

0	if the strings are equal
1	if s1 is greater than s2
-1	if s1 is less than s2

See also

fl_utf_strncasecmp(const char ∗s1, const char ∗s2, int n)

11.11.3.47 fl_utf_strncasecmp()

```
int fl_utf_strncasecmp (
            const char * s1,
```

```
                const char * s2,
                int n )
```

UTF-8 aware strncasecmp - converts to lower case Unicode and tests.
Characters are extracted as Unicode, the converted to their lower case version, and the compared numerically. This does not take language specific collation sequences into account. The limitations of `fl_tolower()` apply.

```
#include <FL/fl_utf8.h>

return fl_utf_strncasecmp("über", "Über"); // returns 0
return fl_utf_strncasecmp("Meier", "Müller"); // returns a negative value
```

Parameters

s1,s2	the UTF-8 strings to compare
n	the maximum number of UTF-8 characters to compare

Returns

result of comparison

Return values

0	if the strings are equal
>0	if s1 is greater than s2
<0	if s1 is less than s2

See also

fl_tolower().

11.11.3.48 fl_utf_tolower()

```
int fl_utf_tolower (
                const unsigned char * str,
                int len,
                char * buf )
```

Converts the string `str` to its lower case equivalent into buf.
The destination buffer `buf` must be provided by the caller. The `buf` length should be at least `3 * len` for 16-bit Unicode.
Example:

```
#include <FL/fl_utf8.h>

const char *str = "ÇA VA?";
auto src_len = strlen(str)
char *buf = malloc(src_len*3 + 1);
puts(fl_utf_tolower(str, src_len, buf));
// should print "ça va?"
```

Parameters

in	*str*	UTF-8 string that will be converted
in	*len*	number of *bytes* to convert, must not be -1
out	*buf*	write the conversion result into this buffer

Returns

number of characters converted

See also

fl_utf_toupper(), fl_tolower(unsigned int ucs)

11.11.3.49 fl_utf_toupper()

```
int fl_utf_toupper (
            const unsigned char * str,
            int len,
            char * buf )
```

Converts the string `str` to its upper case equivalent into buf.
The destination buffer `buf` must be provided by the caller. The `buf` length should be at least `3 * len` for 16-bit Unicode.

Parameters

in	*str*	UTF-8 string that will be converted
in	*len*	number of *bytes* to convert, must not be -1
out	*buf*	write the conversion result into this buffer

Returns

number of characters converted

See also

fl_utf_tolower(), fl_tolower(unsigned int ucs)

11.11.3.50 fl_wcwidth()

```
int fl_wcwidth (
            const char * src )
```

extended wrapper around fl_wcwidth_(unsigned int ucs) function.

Parameters

in	*src*	pointer to start of UTF-8 byte sequence

Returns

width of character in columns

Depending on build options, this function may map C1 control characters (0x80 to 0x9f) to CP1252, and return the width of that character instead. This is not the same behaviour as fl_wcwidth_(unsigned int ucs) .
Note that other control characters and DEL will still return -1, so if you want different behaviour, you need to test for those characters before calling fl_wcwidth(), and handle them separately.

11.11.3.51 fl_wcwidth_()

```
int fl_wcwidth_ (
            unsigned int ucs )
```

Wrapper to adapt Markus Kuhn's implementation of wcwidth() for FLTK.

Parameters

in	*ucs*	Unicode character value

Returns

width of character in columns

See http://www.cl.cam.ac.uk/~mgk25/ucs/wcwidth.c for Markus Kuhn's original implementation of wcwidth() and wcswidth() (defined in IEEE Std 1002.1-2001) for Unicode.
WARNING: this function returns widths for "raw" Unicode characters. It does not even try to map C1 control characters (0x80 to 0x9F) to CP1252, and C0/C1 control characters and DEL will return -1. You are advised to use fl_width(const char* src) instead.

11.12 String handling functions

String handling functions declared in <FL/fl_string_functions.h>

Functions

- char * fl_strdup (const char *s)

  Cross platform interface to POSIX function strdup().
- size_t **fl_strlcpy** (char *, const char *, size_t)

11.12.1 Detailed Description

String handling functions declared in <FL/fl_string_functions.h>

11.12.2 Function Documentation

11.12.2.1 fl_strdup()

```
char * fl_strdup (
            const char * s )
```

Cross platform interface to POSIX function strdup().
The fl_strdup() function returns a pointer to a new string which is a duplicate of the string 's'. Memory for the new string is obtained with malloc(3), and can be freed with free(3).
Implementation:

- POSIX: strdup()
- WinAPI: _strdup()

11.13 Safe widget deletion support functions

These functions, declared in <FL/Fl.H>, support deletion of widgets inside callbacks.

Functions

- void Fl::clear_widget_pointer (Fl_Widget const *w)

  Clears a widget pointer in the watch list.
- void Fl::delete_widget (Fl_Widget *w)

  Schedules a widget for deletion at the next call to the event loop.
- void Fl::do_widget_deletion ()

  Deletes widgets previously scheduled for deletion.
- void Fl::release_widget_pointer (Fl_Widget *&w)

  Releases a widget pointer from the watch list.
- void Fl::watch_widget_pointer (Fl_Widget *&w)

  Adds a widget pointer to the widget watch list.

11.13.1 Detailed Description

These functions, declared in <FL/Fl.H>, support deletion of widgets inside callbacks.
Fl::delete_widget() should be called when deleting widgets or complete widget trees (Fl_Group, Fl_Window, ...) inside callbacks.
The other functions are intended for internal use. The preferred way to use them is by using the helper class Fl_Widget_Tracker.
The following is to show how it works ...
There are three groups of related methods:

1. scheduled widget deletion

- Fl::delete_widget() schedules widgets for deletion
- Fl::do_widget_deletion() deletes all scheduled widgets

1. widget watch list ("smart pointers")

- Fl::watch_widget_pointer() adds a widget pointer to the watch list
- Fl::release_widget_pointer() removes a widget pointer from the watch list
- Fl::clear_widget_pointer() clears a widget pointer *in* the watch list

1. the class Fl_Widget_Tracker:

- the constructor calls Fl::watch_widget_pointer()
- the destructor calls Fl::release_widget_pointer()
- the access methods can be used to test, if a widget has been deleted

See also

Fl_Widget_Tracker.

11.13.2 Function Documentation

11.13.2.1 clear_widget_pointer()

```
void Fl::clear_widget_pointer (
            Fl_Widget const * w )  [extern]
```

Clears a widget pointer *in* the watch list.
This is called when a widget is destroyed (by its destructor). You should never call this directly.

Note

Internal use only !

This method searches the widget watch list for pointers to the widget and clears each pointer that points to it. Widget pointers can be added to the widget watch list by calling Fl::watch_widget_pointer() or by using the helper class Fl_Widget_Tracker (recommended).

See also

Fl::watch_widget_pointer()

class Fl_Widget_Tracker

11.13.2.2 delete_widget()

```
void Fl::delete_widget (
            Fl_Widget * wi )  [extern]
```

Schedules a widget for deletion at the next call to the event loop.
Use this method to delete a widget inside a callback function.
To avoid early deletion of widgets, this function should be called toward the end of a callback and only after any call to the event loop (Fl::wait(), Fl::flush(), Fl::check(), fl_choice(), etc.).
When deleting groups or windows, you must only delete the group or window widget and not the individual child widgets.

Since

FLTK 1.3.4 the widget will be hidden immediately, but the actual destruction will be delayed until the event loop is finished. Up to FLTK 1.3.3 windows wouldn't be hidden before the event loop was done, hence you had to hide() a window in your window close callback if you called Fl::delete_widget() to destroy (and hide) the window.

FLTK 1.3.0 it is not necessary to remove widgets from their parent groups or windows before calling this, because it will be done in the widget's destructor, but it is not a failure to do this nevertheless.

Note

In FLTK 1.1 you **must** remove widgets from their parent group (or window) before deleting them.

See also

Fl_Widget::~Fl_Widget()

11.13.2.3 do_widget_deletion()

```
void Fl::do_widget_deletion ( )  [extern]
```

Deletes widgets previously scheduled for deletion.
This is for internal use only. You should never call this directly.
Fl::do_widget_deletion() is called from the FLTK event loop or whenever you call Fl::wait(). The previously scheduled widgets are deleted in the same order they were scheduled by calling Fl::delete_widget().

See also

Fl::delete_widget(Fl_Widget *wi)

11.13.2.4 release_widget_pointer()

```
void Fl::release_widget_pointer (
            Fl_Widget *& w )  [extern]
```

Releases a widget pointer from the watch list.
This is used to remove a widget pointer that has been added to the watch list with Fl::watch_widget_pointer(), when it is not needed anymore.

Note

Internal use only, please use class Fl_Widget_Tracker instead.

See also

Fl::watch_widget_pointer()

11.13.2.5 watch_widget_pointer()

```
void Fl::watch_widget_pointer (
            Fl_Widget *& w )  [extern]
```

Adds a widget pointer to the widget watch list.

Note

Internal use only, please use class Fl_Widget_Tracker instead.

This can be used, if it is possible that a widget might be deleted during a callback or similar function. The widget pointer must be added to the watch list before calling the callback. After the callback the widget pointer can be queried, if it is NULL. *If* it is NULL, then the widget has been deleted during the callback and must not be accessed anymore. If the widget pointer is *not* NULL, then the widget has not been deleted and can be accessed safely.
After accessing the widget, the widget pointer must be released from the watch list by calling Fl::release_widget_pointer().
Example for a button that is clicked (from its handle() method):

```
Fl_Widget *wp = this;           // save 'this' in a pointer variable
Fl::watch_widget_pointer(wp);   // add the pointer to the watch list
set_changed();                  // set the changed flag
do_callback();                  // call the callback
if (!wp) {                      // the widget has been deleted

  // DO NOT ACCESS THE DELETED WIDGET !

} else {                        // the widget still exists
  clear_changed();              // reset the changed flag
}

Fl::release_widget_pointer(wp); // remove the pointer from the watch list
```

This works, because all widgets call Fl::clear_widget_pointer() in their destructors.

See also

Fl::release_widget_pointer()

Fl::clear_widget_pointer()

An easier and more convenient method to control widget deletion during callbacks is to use the class Fl_Widget_Tracker with a local (automatic) variable.

See also

class Fl_Widget_Tracker

11.14 Mac OS X-specific symbols

Mac OS X-specific symbols declared in <FL/platform.H>

Classes

- class Fl_Mac_App_Menu

Functions

- Fl_Window * **fl_mac_find** (FLWindow *)

  Returns the Fl_Window corresponding to the given macOS-specific window reference.
- CGContextRef **fl_mac_gc** ()

  Returns the macOS-specific graphics context for the current window.
- void fl_mac_set_about (Fl_Callback *cb, void *user_data, int shortcut=0)

  Attaches a callback to the "About myprog" item of the system application menu.
- FLWindow * **fl_mac_xid** (const Fl_Window *win)

  Returns the macOS-specific window reference corresponding to the given Fl_Window object.
- void fl_open_callback (void(*cb)(const char *))

  Register a function called for each file dropped onto an application icon.

Variables

- int fl_mac_os_version

  The version number of the running Mac OS X (e.g., 100604 for 10.6.4, 101300 for 10.13, 140102 for 14.1.2).

11.14.1 Detailed Description

Mac OS X-specific symbols declared in <FL/platform.H>

See also

The Apple OS X Interface

11.14.2 Function Documentation

11.14.2.1 fl_mac_set_about()

```
void fl_mac_set_about (
            Fl_Callback * cb,
            void * user_data,
            int shortcut = 0 )
```

Attaches a callback to the "About myprog" item of the system application menu.
For back-compatibility. Equivalent to Fl_Sys_Menu_Bar::about(Fl_Callback *cb, void *user_data).

11.14.2.2 fl_open_callback()

```
void fl_open_callback (
            void(*)(const char *) cb )
```

Register a function called for each file dropped onto an application icon.
This function is effective only on the Mac OS X platform. `cb` will be called with a single Unix-style file name and path. If multiple files were dropped, `cb` will be called multiple times.
This function should be called before `fl_open_display()` is called, either directly or indirectly (this happens at the first `show()` of a window), to be effective for files dropped on the application icon at launch time. It can also be called at any point to change the function used to open dropped files. A call with a NULL argument, after a previous call, makes the app ignore files dropped later.
Use of this function also requires the application bundle's `Info.plist` file to declare what file types are allowed to be dropped on the icon. File `README.macOS.md` describes how to do that in its section *Make an Application Launchable by Dropping Files on its Icon*.

11.14.3 Variable Documentation

11.14.3.1 fl_mac_os_version

```
int fl_mac_os_version [extern]
```

The version number of the running Mac OS X (e.g., 100604 for 10.6.4, 101300 for 10.13, 140102 for 14.1.2).
FLTK initializes this global variable before main() begins running. If the value is needed in a static initializer, a previous call to Fl::system_driver() makes sure fl_mac_os_version has been initialized.

11.15 Common Dialog Classes and Functions

Common dialog functions for file selection, message output, and more.

Files

- file fl_ask.cxx

  Utility functions for common dialogs.

Classes

- class Fl_Color_Chooser

  The Fl_Color_Chooser widget provides a standard RGB color chooser.
- class Fl_File_Chooser

  The Fl_File_Chooser widget displays a standard file selection dialog that supports various selection modes.

Functions

- void fl_alert (const char *fmt,...)

  Shows an alert message dialog box.
- void fl_beep (int type)

  Emits a system beep.
- int fl_choice (const char *fmt, const char *b0, const char *b1, const char *b2,...)

  Shows a dialog displaying the printf style `fmt` message.
- int fl_choice_n (const char *fmt, const char *b0, const char *b1, const char *b2,...)

  Shows a dialog displaying the printf style `fmt` message.
- int fl_color_chooser (const char *name, double &r, double &g, double &b, int cmode)

  Pops up a window to let the user pick an arbitrary RGB color.
- int fl_color_chooser (const char *name, uchar &r, uchar &g, uchar &b, int cmode)

  Pops up a window to let the user pick an arbitrary RGB color.
- char * fl_dir_chooser (const char *message, const char *fname, int relative)

  Shows a file chooser dialog and gets a directory.
- char * fl_file_chooser (const char *message, const char *pat, const char *fname, int relative)

  Shows a file chooser dialog and gets a filename.
- void fl_file_chooser_callback (void(*cb)(const char *))

  Set the file chooser callback.
- void fl_file_chooser_ok_label (const char *l)

  Set the "OK" button label.
- const char * fl_input (const char *fmt, const char *defstr,...)

  Shows an input dialog displaying the `fmt` message with variable arguments.
- const char * fl_input (int maxchar, const char *fmt, const char *defstr,...)

  Shows an input dialog displaying the `fmt` message with variable arguments.
- std::string fl_input_str (int &ret, int maxchar, const char *fmt, const char *defstr,...)

  Shows an input dialog displaying the `fmt` message with variable arguments.
- std::string fl_input_str (int maxchar, const char *fmt, const char *defstr,...)

  Shows an input dialog displaying the `fmt` message with variable arguments.
- void fl_message (const char *fmt,...)

  Shows an information message dialog box.
- int fl_message_hotspot ()

  Gets whether or not to move the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password() to follow the mouse pointer.
- void fl_message_hotspot (int enable)

  Sets whether or not to move the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password() to follow the mouse pointer.
- Fl_Widget * fl_message_icon ()

  Gets the Fl_Box icon container of the current default dialog used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().
- void fl_message_icon_label (const char *str)

  Sets the icon label of the dialog window used in many common dialogs.
- void fl_message_position (const int x, const int y, const int center)

  Sets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().

- void fl_message_position (Fl_Widget *widget)

  Sets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().
- int fl_message_position (int *x, int *y)

  Gets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().
- void fl_message_title (const char *title)

  Sets the title of the dialog window used in many common dialogs.
- void fl_message_title_default (const char *title)

  Sets the default title of the dialog window used in many common dialogs.
- const char * fl_password (const char *fmt, const char *defstr,...)

  Shows an input dialog displaying the `fmt` message with variable arguments.
- const char * fl_password (int maxchar, const char *fmt, const char *defstr,...)

  Shows an input dialog displaying the `fmt` message with variable arguments.
- std::string fl_password_str (int &ret, int maxchar, const char *fmt, const char *defstr,...)

  Shows an input dialog displaying the `fmt` message with variable arguments.
- std::string fl_password_str (int maxchar, const char *fmt, const char *defstr,...)

  Shows an input dialog displaying the `fmt` message with variable arguments.

Variables

- void(* Fl::error)(const char *,...) = Fl_System_Driver::error

  FLTK calls Fl::error() to output a normal error message.
- void(* Fl::fatal)(const char *,...) = Fl_System_Driver::fatal

  FLTK calls Fl::fatal() to output a fatal error message.
- const char * **fl_cancel** = "Cancel"

  string pointer used in common dialogs, you can change it to another language
- const char * **fl_close** = "Close"

  string pointer used in common dialogs, you can change it to another language
- Fl_Font **fl_message_font_** = FL_HELVETICA
- Fl_Fontsize **fl_message_size_** = -1
- const char * **fl_no** = "No"

  string pointer used in common dialogs, you can change it to another language
- const char * **fl_ok** = "OK"

  string pointer used in common dialogs, you can change it to another language
- const char * **fl_yes** = "Yes"

  string pointer used in common dialogs, you can change it to another language
- void(* Fl::warning)(const char *,...) = Fl_System_Driver::warning

  FLTK calls Fl::warning() to output a warning message.

11.15.1 Detailed Description

Common dialog functions for file selection, message output, and more.

11.15.2 Function Documentation

11.15.2.1 fl_alert()

```
void fl_alert (
            const char * fmt,
             ...  )
```

Shows an alert message dialog box.

```
#include <FL/fl_ask.H>
```

Parameters

in	*fmt*	can be used as an sprintf-like format and variables for the message text

11.15.2.2 fl_beep()

```
void fl_beep (
            int type )
```

Emits a system beep.
This function is platform specific. Depending on the input `type` a different sound may be played or the system speaker may beep with a different volume.
On X the system speaker is used which may not work at all on newer systems that don't have a speaker. Since 1.4.0 `FL_BEEP_DEFAULT` and other types honor the system or user settings whereas `FL_BEEP_ERROR` uses 100% volume. This may be changed in a future version.
On Wayland an ASCII `BEL` (0x07) is output to stderr.
On Windows the `MessageBeep()` function is used to play different sounds depending on the `type` argument.
On macOS the system beep function `NSBeep()` is used for `FL_BEEP_DEFAULT` and `FL_BEEP_ERROR`. Other types are ignored.
On other platforms the behavior is undefined and may change in the future.

Parameters

in	*type*	The beep type from the Fl_Beep enumeration (optional)

```
#include <FL/fl_ask.H>
```

11.15.2.3 fl_choice()

```
int fl_choice (
            const char * fmt,
            const char * b0,
            const char * b1,
            const char * b2,
             ...  )
```

Shows a dialog displaying the printf style `fmt` message.
This dialog features up to 3 customizable choice buttons which are specified in order of *right-to-left* in the dialog, e.g.

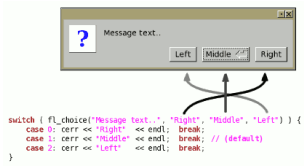


Figure 11.7 fl_choice() button ordering

```
#include <FL/fl_ask.H>
```

Three choices with printf() style formatting:

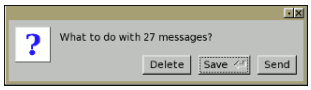


Figure 11.8 fl_choice() three choices with printf formatting

```
int num_msgs = GetNumberOfMessages();
switch ( fl_choice("What to do with %d messages?", "Send", "Save", "Delete", num_msgs) ) {
  case 0: .. // Send
  case 1: .. // Save (default)
  case 2: .. // Delete
```

```
  ..
}
```

Three choice example:

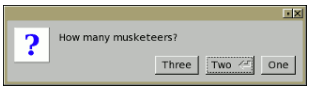


Figure 11.9 fl_choice() three choices

```
switch ( fl_choice("How many bedrooms?", "Zero", "One", "Two") ) {
  case 0: .. // "Zero"
  case 1: .. // "One" (default)
  case 2: .. // "Two"
}
```

Two choice example:

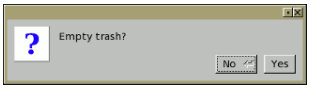


Figure 11.10 fl_choice() two choices

```
switch ( fl_choice("Empty trash?", "Yes", "No", 0) ) {
  case 0: .. // Yes
  case 1: .. // No (default)
}
```

One choice example:

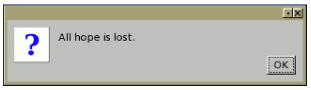


Figure 11.11 fl_choice() one choice

```
fl_choice("All hope is lost.", "OK", 0, 0);   // "OK" default
```

Parameters

in	*fmt*	can be used as an sprintf-like format and variables for the message text
in	*b0*	text label for right button 0
in	*b1*	text label for middle button 1 (can be 0)
in	*b2*	text label for left button 2 (can be 0)

Return values

0	if the button with `b0` text is pushed or the user pressed the `Escape` key or clicked the window close button
1	if the button with `b1` text is pushed or the user pressed the `Return` key
2	if the button with `b2` text is pushed

11.15.2.4 fl_choice_n()

```
int fl_choice_n (
            const char * fmt,
            const char * b0,
            const char * b1,
            const char * b2,
             ...  )
```

Shows a dialog displaying the printf style `fmt` message.

This function is like fl_choice() but returns `-1` if the dialog window was closed by pressing the `Escape` key or the window close button rather than pushing one of the dialog buttons.

See also

fl_choice()

Parameters

in	*fmt*	can be used as an sprintf-like format and variables for the message text
in	*b0*	text label for right button 0
in	*b1*	text label for middle button 1 (can be 0)
in	*b2*	text label for left button 2 (can be 0)

Return values

-3	reserved, FLTK 1.3 only: another dialog is still open (not possible in 1.4)
-2	if the dialog was closed by pushing the window close button
-1	if the dialog was closed by hitting Escape
0	if the button with `b0` text is pushed
1	if the button with `b1` text is pushed
2	if the button with `b2` text is pushed

11.15.2.5 fl_color_chooser() [1/2]

```
int fl_color_chooser (
            const char * name,
            double & r,
            double & g,
            double & b,
            int cmode )  [related]
```

Pops up a window to let the user pick an arbitrary RGB color.

Note

#include <FL/Fl_Color_Chooser.H>

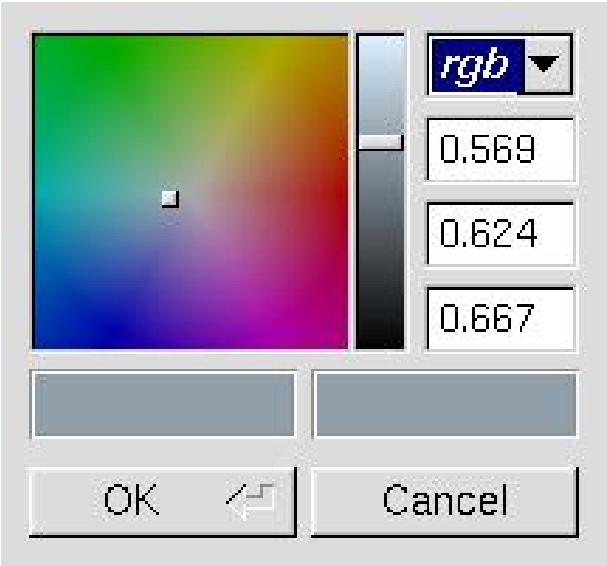


Figure 11.12 fl_color_chooser

Parameters

in	*name*	Title label for the window
in,out	*r,g,b*	Color components in the range 0.0 to 1.0.
in	*cmode*	Optional mode for color chooser. See mode(int). Default -1 if none (rgb mode).

Return values

1	if user confirms the selection
0	if user cancels the dialog

11.15.2.6 fl_color_chooser() [2/2]

```
int fl_color_chooser (
            const char * name,
            uchar & r,
            uchar & g,
            uchar & b,
            int cmode )  [related]
```

Pops up a window to let the user pick an arbitrary RGB color.

Note

#include <FL/Fl_Color_Chooser.H>

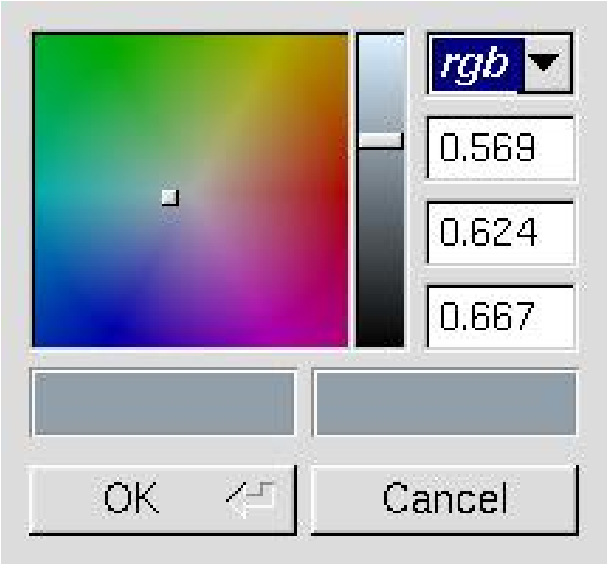


Figure 11.13 fl_color_chooser

Parameters

in	*name*	Title label for the window
in,out	*r,g,b*	Color components in the range 0 to 255.
in	*cmode*	Optional mode for color chooser. See mode(int). Default -1 if none (rgb mode).

Return values

1	if user confirms the selection

Return values

0	if user cancels the dialog

11.15.2.7 fl_dir_chooser()

```
char * fl_dir_chooser (
            const char * message,
            const char * fname,
            int relative )  [related]
```

Shows a file chooser dialog and gets a directory.

Note

#include <FL/Fl_File_Chooser.H>

Parameters

in	*message*	title bar text
in	*fname*	initial/default directory name
in	*relative*	0 for absolute path return, relative otherwise

Returns

the directory path string chosen by the user or NULL if user cancels

11.15.2.8 fl_file_chooser()

```
char * fl_file_chooser (
            const char * message,
            const char * pat,
            const char * fname,
            int relative )  [related]
```

Shows a file chooser dialog and gets a filename.

Note

#include <FL/Fl_File_Chooser.H>

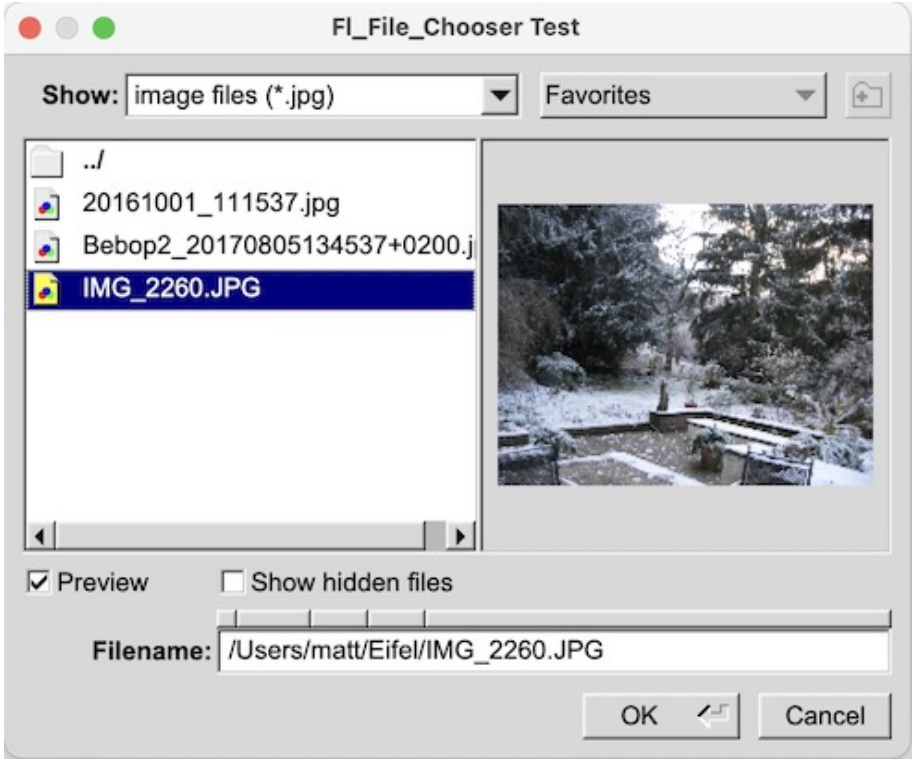


Figure 11.14 Fl_File_Chooser

Parameters

in	*message*	text in title bar
in	*pat*	filename pattern filter
in	*fname*	initial/default filename selection
in	*relative*	0 for absolute path name, relative path name otherwise

Returns

the user selected filename, in absolute or relative format or NULL if user cancels

11.15.2.9 fl_file_chooser_callback()

```
void fl_file_chooser_callback (
            void(*)(const char *) cb )  [related]
```

Set the file chooser callback.

Note

#include <FL/Fl_File_Chooser.H>

11.15.2.10 fl_file_chooser_ok_label()

```
void fl_file_chooser_ok_label (
            const char * l )  [related]
```

Set the "OK" button label.

Note

#include <FL/Fl_File_Chooser.H>

11.15.2.11 fl_input() [1/2]

```
const char * fl_input (
            const char * fmt,
            const char * defstr,
            ...  )
```

Shows an input dialog displaying the `fmt` message with variable arguments.
Returns the string in an internally allocated buffer that may be changed later. You **must** copy the string immediately after return from this method - at least before the next execution of the event loop.
`#include <FL/fl_ask.H>`

Parameters

in	*fmt*	can be used as an sprintf-like format and variables for the message text
in	*defstr*	defines the default returned string if no text is entered

Returns

the user string input if OK was pushed

Return values

NULL	if Cancel was pushed or the window was closed by the user

11.15.2.12 fl_input() [2/2]

```
const char * fl_input (
            int maxchar,
            const char * fmt,
            const char * defstr,
            ...  )
```

Shows an input dialog displaying the `fmt` message with variable arguments.
This is the same as const char ∗fl_input(const char ∗fmt, const char ∗defstr, ...) except that it has an additional parameter to limit the number of characters the user can input.
Returns the string in an internally allocated buffer that may be changed later. You **must** copy the string immediately after return from this method - at least before the next execution of the event loop.
`#include <FL/fl_ask.H>`

Parameters

in	*maxchar*	maximum number of characters the user can input (UTF-8 aware)
in	*fmt*	can be used as an sprintf-like format and variables for the message text
in	*defstr*	defines the default returned string if no text is entered

Returns

the user string input if OK was pushed

Return values

NULL	if Cancel was pushed or the window was closed by the user

11.15.2.13 fl_input_str() [1/2]

```
std::string fl_input_str (
            int & ret,
            int maxchar,
            const char * fmt,
            const char * defstr,
            ...  )
```

Shows an input dialog displaying the `fmt` message with variable arguments.
Like fl_input(), but this method has the additional argument `maxchar` that limits the number of **characters** that can be input. Since the string is encoded in UTF-8 it is possible that the number of bytes in the string is larger than `maxchar`.
Other than the deprecated fl_input() method w/o the `maxchar` argument, this one returns the string in an std::string object that must be released after use. This can be a local/automatic variable.
The `ret` variable is set to 0 if the user clicked OK, and to a negative value if the user canceled the dialog. If the dialog was canceled, the returned string will be empty.

```
#include <FL/fl_ask.H>
```

Example:

```
{ int ret;
  std::string str = fl_input_str(ret, 0, "Enter text:", "");
  if (ret < 0)
    printf("Text input was canceled.\n");
  else
    printf("Text is: '%s'\n", str.c_str());
} // (str goes out of scope)
```

Parameters

out	*ret*	0 if user clicked OK, negative if dialog was canceled
in	*maxchar*	input size limit in characters (not bytes), use 0 for no limit
in	*fmt*	can be used as an sprintf-like format and variables for the message text
in	*defstr*	defines the default returned string if no text is entered

Returns

the user string input if OK was clicked which can be empty

an empty string and set `ret` to a negative value if the user canceled the dialog

Since

1.4.0

11.15.2.14 fl_input_str() [2/2]

```
std::string fl_input_str (
            int maxchar,
            const char * fmt,
            const char * defstr,
            ...  )
```

Shows an input dialog displaying the `fmt` message with variable arguments.

Note

No information is given if the user canceled the dialog or clicked OK.

See also

fl_input_str(int &ret, int maxchar, const char ∗label, const char ∗deflt = 0, ...)

11.15.2.15 fl_message()

```
void fl_message (
            const char * fmt,
             ...  )
```

Shows an information message dialog box.

```
#include <FL/fl_ask.H>
```

Parameters

in	*fmt*	can be used as an sprintf-like format and variables for the message text

11.15.2.16 fl_message_hotspot() [1/2]

```
int fl_message_hotspot (
            void  )
```

Gets whether or not to move the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password() to follow the mouse pointer.

This is a permanent setting. It remains active and affects the window position unless overridden by an explicit positioning request by means of one of the fl_message_position() variants.

```
#include <FL/fl_ask.H>
```

Returns

0 if disabled, non-zero otherwise

See also

void fl_message_hotspot(int)

int fl_message_position(int *x, int *y)

void fl_message_position(Fl_Widget *)

fl_message_position()

11.15.2.17 fl_message_hotspot() [2/2]

```
void fl_message_hotspot (
            int enable )
```

Sets whether or not to move the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password() to follow the mouse pointer.

The default is *enabled*, so that the default button is the hotspot and appears at the mouse position.

```
#include <FL/fl_ask.H>
```

Parameters

in	*enable*	non-zero enables hotspot behavior, 0 disables hotspot

11.15.2.18 fl_message_icon()

```
Fl_Widget * fl_message_icon ( )
```

Gets the Fl_Box icon container of the current default dialog used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().

The return value cannot be Null. The object pointed to is an Fl_Box widget. The returned pointer (Fl_Widget *) can be safely cast to an Fl_Box* pointer.

Note

You can set some attributes of this **default** icon box. These attributes are sticky, i.e. they will be used in all subsequent common dialogs unless overridden by specific "one shot" variables. Setting any attribute except those mentioned below causes undefined behavior.

Supported icon attributes:

- box()
- labelfont()
- labelsize()
- color()
- labelcolor()
- image()
- align()

The icon size can not be changed. If you set an image() you should scale it to the available size, i.e. `w()` and `h()` of the icon box.

```
#include <FL/fl_ask.H>
```

11.15.2.19 fl_message_icon_label()

```
void fl_message_icon_label (
            const char * str )
```

Sets the icon label of the dialog window used in many common dialogs.
This icon label will be used in the next call of one of the common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().
The label `str` is stored internally as a reference, it must be in scope until the dialog function (e.g. fl_choice) is called.
It applies only to the **next** call of one of the common dialogs and will be reset after that call so the next dialog will use its default label unless set again.

Note

This label string must be short, usually only one character so it fits in the icon box. You can use any valid UTF-8 character, e.g. the Euro sign ("€") which is three bytes in UTF-8 encoding.

```
#include <FL/fl_ask.H>
```

Parameters

in	*str*	icon label

11.15.2.20 fl_message_position() [1/3]

```
void fl_message_position (
            const int x,
            const int y,
            const int center )
```

Sets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().
The position set with this method overrides the hotspot setting, i.e. setting a position has higher priority than the hotspot mode set by fl_message_hotspot(int).
The preferred position set by any of the fl_message_position() variants affects only the next call of one of the common dialogs. The preferred position is reset to 0 (unset) as soon as the dialog is shown.
If the optional argument `center` is non-zero (true) the message box will be centered at the given coordinates rather than using the X/Y position as the window position (top left corner).

```
#include <FL/fl_ask.H>
```

Parameters

in	*x*	Preferred X position
in	*y*	Preferred Y position
in	*center*	1 = centered, 0 = absolute

See also

int fl_message_position(int *x, int *y)

11.15.2.21 fl_message_position() [2/3]

```
void fl_message_position (
            Fl_Widget * widget )
```

Sets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().
The message box will be centered over the given widget or window extensions.
Everything else is like fl_message_position(int, int, int) with argument 'center' set to 1.

```
#include <FL/fl_ask.H>
```

Parameters

in	*widget*	Widget or window to position the message box over.

See also

int fl_message_position(int x, int y, int center)

11.15.2.22 fl_message_position() [3/3]

```
int fl_message_position (
            int * x,
            int * y )
```

Gets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().

```
#include <FL/fl_ask.H>
```

The position set with this method overrides the hotspot setting, i.e. setting a position has higher priority than the hotspot mode set by fl_message_hotspot(int).
The preferred position set by any of the fl_message_position() variants affects only the next call of one of the common dialogs. The preferred position is reset to 0 (unset) as soon as the dialog is shown.

Parameters

out	*x*	Preferred X position, returns -1 if not set
out	*y*	Preferred Y position, returns -1 if not set

Returns

whether position is currently set or not

Return values

0	position is not set (hotspot may be enabled or not)
1	position is set (window position)

Return values

2	position is set (message box centered)

See also

fl_message_hotspot()

fl_message_hotspot(int)

fl_message_position(int, int)

fl_message_position(const int x, const int y, const int center)

fl_message_position(Fl_Widget *)

11.15.2.23 fl_message_title()

```
void fl_message_title (
            const char * title )
```

Sets the title of the dialog window used in many common dialogs.
This window `title` will be used in the next call of one of the common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().
The `title` string is copied internally, so that you can use a local variable or free the string immediately after this call. It applies only to the **next** call of one of the common dialogs and will be reset to an empty title (the default for all dialogs) after that call.
`#include <FL/fl_ask.H>`

Parameters

in	*title*	window label, string copied internally

11.15.2.24 fl_message_title_default()

```
void fl_message_title_default (
            const char * title )
```

Sets the default title of the dialog window used in many common dialogs.
This window `title` will be used in all subsequent calls of one of the common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password(), unless a specific title has been set with fl_message_title(const char *title).
The default is no title. You can override the default title for a single dialog with fl_message_title(const char *title).
The `title` string is copied internally, so that you can use a local variable or free the string immediately after this call.
`#include <FL/fl_ask.H>`

Parameters

in	*title*	default window label, string copied internally

11.15.2.25 fl_password() [1/2]

```
const char * fl_password (
            const char * fmt,
            const char * defstr,
             ...  )
```

Shows an input dialog displaying the `fmt` message with variable arguments.
Like fl_input() except the input text is not shown, '*' or similar replacement characters are displayed instead.
`#include <FL/fl_ask.H>`

Parameters

in	*fmt*	can be used as an sprintf-like format and variables for the message text
in	*defstr*	defines the default returned string if no text is entered

Returns

the user string input if OK was pushed

Return values

NULL	if Cancel was pushed or the window was closed by the user

11.15.2.26 fl_password() [2/2]

```
const char * fl_password (
            int maxchar,
            const char * fmt,
            const char * defstr,
             ...  )
```

Shows an input dialog displaying the `fmt` message with variable arguments.
Like fl_input() except the input text is not shown, '*' or similar replacement characters are displayed instead.
`#include <FL/fl_ask.H>`

Parameters

in	*maxchar*	input lenght limit in chars, 0 = no limit
in	*fmt*	can be used as an sprintf-like format and variables for the message text
in	*defstr*	defines the default returned string if no text is entered

Returns

the user string input if OK was pushed

Return values

NULL	if Cancel was pushed or the window was closed by the user

11.15.2.27 fl_password_str() [1/2]

```
std::string fl_password_str (
            int & ret,
            int maxchar,
            const char * fmt,
            const char * defstr,
             ...  )
```

Shows an input dialog displaying the `fmt` message with variable arguments.
Like fl_input_str() except the input text is not shown, '*' or similar replacement characters are displayed instead.
Other than the fl_password() method w/o the `maxchar` argument, this one returns the string in an std::string object that must be released after use. This can be a local/automatic variable.
For an example see fl_input_str()
`#include <FL/fl_ask.H>`

Parameters

out	*ret*	0 if user clicked OK, negative if dialog was canceled
in	*maxchar*	input size limit in characters (not bytes), use 0 for no limit
in	*fmt*	can be used as an sprintf-like format and variables for the message text
in	*defstr*	defines the default returned string if no text is entered

Returns

the user string input if OK was clicked which can be empty

an empty string and set `ret` to a negative value if the user canceled the dialog

Since

1.4.0

11.15.2.28 fl_password_str() [2/2]

```
std::string fl_password_str (
            int maxchar,
            const char * fmt,
            const char * defstr,
             ...  )
```

Shows an input dialog displaying the `fmt` message with variable arguments.

Note

No information is given if the user canceled the dialog or clicked OK.

See also

fl_password_str(int &ret, int maxchar, const char ∗label, const char ∗deflt = 0, ...)

11.15.3 Variable Documentation

11.15.3.1 error

```
void(* Fl::error)(const char *format,...)  (
            const char * format,
             ...  )  = Fl_System_Driver::error  [extern]
```

FLTK calls Fl::error() to output a normal error message.
The default version on Windows displays the error message in a MessageBox window.
The default version on all other platforms prints the error message to stderr.
You can override the behavior by setting the function pointer to your own routine.
Fl::error() means there is a recoverable error such as the inability to read an image file. The default implementation returns after displaying the message.

Note

#include <FL/Fl.H>

11.15.3.2 fatal

```
void(* Fl::fatal)(const char *format,...)  (
            const char * format,
             ...  )  = Fl_System_Driver::fatal  [extern]
```

FLTK calls Fl::fatal() to output a fatal error message.
The default version on Windows displays the error message in a MessageBox window.

The default version on all other platforms prints the error message to stderr.
You can override the behavior by setting the function pointer to your own routine.
Fl::fatal() must not return, as FLTK is in an unusable state, however your version may be able to use longjmp or an exception to continue, as long as it does not call FLTK again. The default implementation exits with status 1 after displaying the message.

Note

#include <FL/Fl.H>

11.15.3.3 warning

```
void(* Fl::warning)(const char *format,...)  (
            const char * format,
             ...  )  = Fl_System_Driver::warning  [extern]
```

FLTK calls Fl::warning() to output a warning message.
The default version on Windows returns *without* printing a warning message, because Windows programs normally don't have stderr (a console window) enabled.
The default version on all other platforms prints the warning message to stderr.
You can override the behavior by setting the function pointer to your own routine.
Fl::warning() means that there was a recoverable problem, the display may be messed up, but the user can probably keep working - all X protocol errors call this, for example. The default implementation returns after displaying the message.

Note

#include <FL/Fl.H>

11.16 File names and URI utility functions

File names and URI functions defined in <FL/filename.H>

Macros

- #define **FL_PATH_MAX** 2048

  all path buffers should use this length

Typedefs

- typedef int() Fl_File_Sort_F(struct dirent **, struct dirent **)

  File sorting function.

Functions

- void fl_decode_uri (char *uri)

  Decodes a URL-encoded string.
- int fl_filename_absolute (char *to, int tolen, const char *from)

  Makes a filename absolute from a relative filename to the current working directory.
- int fl_filename_absolute (char *to, int tolen, const char *from, const char *cwd)

  Concatenate the absolute path `base` with `from` to form the new absolute path in `to`.
- std::string fl_filename_absolute_str (const std::string &from)

  Makes a filename absolute from a filename relative to the current working directory.
- std::string fl_filename_absolute_str (const std::string &from, const std::string &base)

  Append the relative filename `from` to the absolute filename `base` to form the new absolute path.
- int fl_filename_expand (char *to, int tolen, const char *from)

  Expands a filename containing shell variables and tilde (~).
- std::string fl_filename_expand_str (const std::string &from)

Expands a filename containing shell variables and tilde (~).

- const char * fl_filename_ext (const char *buf)

  Gets the extension of a filename.
- std::string fl_filename_ext_str (const std::string &filename)

  Return a new string that contains the filename extension.
- void fl_filename_free_list (struct dirent ***l, int n)

  Free the list of filenames that is generated by fl_filename_list().
- int fl_filename_isdir (const char *name)

  Determines if a file exists and is a directory from its filename.
- int fl_filename_list (const char *d, struct dirent ***l, Fl_File_Sort_F *s=fl_numericsort)

  Portable and const-correct wrapper for the scandir() function.
- int fl_filename_match (const char *name, const char *pattern)

  Checks if a string `s` matches a pattern `p`.
- const char * fl_filename_name (const char *filename)

  Gets the file name from a path.
- std::string fl_filename_name_str (const std::string &filename)

  Return a new string that contains the name part of the filename.
- std::string fl_filename_path_str (const std::string &filename)

  Return a new string that contains the path part of the filename.
- int fl_filename_relative (char *to, int tolen, const char *from)

  Makes a filename relative to the current working directory.
- int fl_filename_relative (char *to, int tolen, const char *from, const char *cwd)

  Makes a filename relative to any other directory.
- std::string fl_filename_relative_str (const std::string &from)

  Makes a filename relative to the current working directory.
- std::string fl_filename_relative_str (const std::string &from, const std::string &base)

  Makes a filename relative to any directory.
- char * fl_filename_setext (char *to, int tolen, const char *ext)

  Replaces the extension in `buf` of max.
- std::string fl_filename_setext_str (const std::string &filename, const std::string &new_extension)

  Return a copy of the old filename with the new extension.
- std::string fl_getcwd_str ()

  Cross-platform function to get the current working directory as a UTF-8 encoded value in an std::string.
- int fl_open_uri (const char *uri, char *msg, int msglen)

  Opens the specified Uniform Resource Identifier (URI).

11.16.1 Detailed Description

File names and URI functions defined in <FL/filename.H>

11.16.2 Typedef Documentation

11.16.2.1 Fl_File_Sort_F

```
typedef int() Fl_File_Sort_F(struct dirent **, struct dirent **)
```

File sorting function.

See also

fl_filename_list()

11.16.3 Function Documentation

11.16.3.1 fl_decode_uri()

```
void fl_decode_uri (
            char * uri )
```

Decodes a URL-encoded string.

In a Uniform Resource Identifier (URI), all non-ASCII bytes and several others (e.g., '<', '', ' ') are URL-encoded using 3 bytes by "%XY" where XY is the hexadecimal value of the byte. This function decodes the URI restoring its original UTF-8 encoded content. Decoding is done in-place.

11.16.3.2 fl_filename_absolute() [1/2]

```
int fl_filename_absolute (
            char * to,
            int tolen,
            const char * from )
```

Makes a filename absolute from a relative filename to the current working directory.

```
#include <FL/filename.H>
[..]
fl_chdir("/var/tmp");
fl_filename_absolute(out, sizeof(out), "foo.txt");          // out="/var/tmp/foo.txt"
fl_filename_absolute(out, sizeof(out), "./foo.txt");        // out="/var/tmp/foo.txt"
fl_filename_absolute(out, sizeof(out), "../log/messages"); // out="/var/log/messages"
```

Parameters

out	*to*	resulting absolute filename
in	*tolen*	size of the absolute filename buffer
in	*from*	relative filename

Returns

0 if no change, non zero otherwise

11.16.3.3 fl_filename_absolute() [2/2]

```
int fl_filename_absolute (
            char * to,
            int tolen,
            const char * from,
            const char * base )
```

Concatenate the absolute path `base` with `from` to form the new absolute path in `to`.

```
#include <FL/filename.H>
char out[FL_PATH_MAX];
fl_filename_absolute(out, sizeof(out), "../foo.txt", "/var/tmp");   // out="/var/foo.txt"
fl_filename_absolute(out, sizeof(out), "../local/bin", "/usr/bin");  // out="/usr/local/bin"
```

Parameters

out	*to*	resulting absolute filename
in	*tolen*	size of the absolute filename buffer
in	*from*	relative filename
in	*base*	`from` is relative to this absolute file path

Returns

0 if no change, non zero otherwise

11.16.3.4 fl_filename_absolute_str() [1/2]

```
std::string fl_filename_absolute_str (
            const std::string & from )
```

Makes a filename absolute from a filename relative to the current working directory.

Parameters

in	*from*	relative filename

Returns

the new, absolute filename

See also

fl_filename_absolute(char *to, int tolen, const char *from)

11.16.3.5 fl_filename_absolute_str() [2/2]

```
std::string fl_filename_absolute_str (
            const std::string & from,
            const std::string & base )
```

Append the relative filename `from` to the absolute filename `base` to form the new absolute path.

Parameters

in	*from*	relative filename
in	*base*	`from` is relative to this absolute file path

Returns

the new, absolute filename

See also

fl_filename_absolute(char *to, int tolen, const char *from, const char *base)

11.16.3.6 fl_filename_expand()

```
int fl_filename_expand (
            char * to,
            int tolen,
            const char * from )
```

Expands a filename containing shell variables and tilde (~).

Currently handles these variants:

```
"~username"              // if 'username' does not exist, result will be unchanged
"~/file"
"$VARNAME"               // does NOT handle ${VARNAME}
```

Examples:

```
#include <FL/filename.H>
[..]
putenv("TMPDIR=/var/tmp");
fl_filename_expand(out, sizeof(out), "~fred/.cshrc");     // out="/usr/fred/.cshrc"
fl_filename_expand(out, sizeof(out), "~/.cshrc");         // out="/usr/<yourname>/.cshrc"
fl_filename_expand(out, sizeof(out), "$TMPDIR/foo.txt");  // out="/var/tmp/foo.txt"
```

Parameters

out	*to*	resulting expanded filename
in	*tolen*	size of the expanded filename buffer
in	*from*	filename containing shell variables

Returns

0 if no change, non zero otherwise

11.16.3.7 fl_filename_expand_str()

```
std::string fl_filename_expand_str (
            const std::string & from )
```

Expands a filename containing shell variables and tilde (~).

Parameters

in	*from*	file path and name

Returns

the new, expanded filename

See also

fl_filename_expand(char *to, int tolen, const char *from)

11.16.3.8 fl_filename_ext()

```
const char * fl_filename_ext (
            const char * buf )
```

Gets the extension of a filename.

```
#include <FL/filename.H>
[..]
const char *out;
out = fl_filename_ext("/some/path/foo.txt");        // result: ".txt"
out = fl_filename_ext("/some/path/foo");            // result: ""
```

Parameters

in	*buf*	the filename to be parsed

Returns

a pointer to the extension (including '.') if one was found, or a pointer to the terminating NUL character (effectively a pointer to an empty string) if no extension was found.

See also

fl_filename_ext_str(const std::string &filename)

11.16.3.9 fl_filename_ext_str()

```
std::string fl_filename_ext_str (
            const std::string & filename )
```

Return a new string that contains the filename extension.

Parameters

in	*filename*	file path and name

Returns

the filename extension including the prepending '.', or an empty string if the filename has no extension

See also

fl_filename_ext(const char *buf)

11.16.3.10 fl_filename_free_list()

```
void fl_filename_free_list (
            struct dirent *** list,
            int n )
```

Free the list of filenames that is generated by fl_filename_list().
Free everything that was allocated by a previous call to fl_filename_list(). Use the return values as parameters for this function.

Parameters

in,out	*list*	table containing the resulting directory listing
in	*n*	number of entries in the list

11.16.3.11 fl_filename_isdir()

```
int fl_filename_isdir (
            const char * n )
```

Determines if a file exists and is a directory from its filename.

```
#include <FL/filename.H>
[..]
fl_filename_isdir("/etc");          // returns non-zero
fl_filename_isdir("/etc/hosts");    // returns 0
```

Parameters

in	*n*	the filename to parse

Returns

non zero if file exists and is a directory, zero otherwise

11.16.3.12 fl_filename_list()

```
int fl_filename_list (
            const char * d,
            dirent *** list,
            Fl_File_Sort_F * sort )
```

Portable and const-correct wrapper for the scandir() function.
For each file in that directory a "dirent" structure is created. The only portable thing about a dirent is that dirent.↩ d_name is the nul-terminated file name. A pointers array to these dirent's is created and a pointer to the array is returned in *list. The number of entries is given as a return value. If there is an error reading the directory a number less than zero is returned, and errno has the reason; errno does not work under Windows.
Include:

```
#include <FL/filename.H>
```

Parameters

in	*d*	the name of the directory to list. It does not matter if it has a trailing slash.
out	*list*	table containing the resulting directory listing
in	*sort*	sorting functor: • fl_alphasort: The files are sorted in ascending alphabetical order; upper and lowercase letters are compared according to their ASCII ordering uppercase before lowercase. • fl_casealphasort: The files are sorted in ascending alphabetical order; upper and lowercase letters are compared equally case is not significant. • fl_casenumericsort: The files are sorted in ascending "alphanumeric" order, where an attempt is made to put unpadded numbers in consecutive order; upper and lowercase letters are compared equally case is not significant. • fl_numericsort: The files are sorted in ascending "alphanumeric" order, where an attempt is made to put unpadded numbers in consecutive order; upper and lowercase letters are compared according to their ASCII ordering - uppercase before lowercase.

Returns

the number of entries if no error, a negative value otherwise.

Todo should support returning OS error messages

11.16.3.13 fl_filename_match()

```
int fl_filename_match (
            const char * s,
            const char * p )
```

Checks if a string `s` matches a pattern `p`.
The following syntax is used for the pattern:

- ∗ matches any sequence of 0 or more characters.
- ? matches any single character.
- [set] matches any character in the set. Set can contain any single characters, or a-z to represent a range. To match] or - they must be the first characters. To match ^ or ! they must not be the first characters.
- [^set] or [!set] matches any character not in the set.
- {X|Y|Z} or {X,Y,Z} matches any one of the subexpressions literally.
- \x quotes the character x so it has no special meaning.
- x all other characters are matched "exactly" on a **case-insensitive** basis.

Notes:

- `s` and `p` are matched on a char/byte basis, not as UCS codepoints or UTF-8 sequences.
- [set] ranges must run from low to high, i.e. [a-z] and not [z-a]
- [set] comparison is **case-sensitive**, i.e. [a-z] won't match "A".
- \x only applies to the fl_filename_match special characters ∗ ? [{
- \x needs a double \ or the compiler will complain about non-standard escape sequences.

Include:
`#include <FL/filename.H>`

Parameters

in	*s*	the string to check for a match
in	*p*	the string pattern

Returns

non zero if the string matches the pattern

11.16.3.14 fl_filename_name()

```
const char * fl_filename_name (
            const char * filename )
```

Gets the file name from a path.
Similar to basename(3), exceptions shown below.

```
#include <FL/filename.H>
[..]
const char *out;
out = fl_filename_name("/usr/lib");     // out="lib"
out = fl_filename_name("/usr/");        // out=""      (basename(3) returns "usr" instead)
out = fl_filename_name("/usr");         // out="usr"
out = fl_filename_name("/");            // out=""      (basename(3) returns "/" instead)
out = fl_filename_name(".");            // out="."
out = fl_filename_name("..");           // out=".."
```

Returns

a pointer to the char after the last slash, or to `filename` if there is none.

11.16.3.15 fl_filename_name_str()

```
std::string fl_filename_name_str (
            const std::string & filename )
```

Return a new string that contains the name part of the filename.

Parameters

in	*filename*	file path and name

Returns

the name part of a filename

See also

fl_filename_name(const char *filename)

11.16.3.16 fl_filename_path_str()

```
std::string fl_filename_path_str (
            const std::string & filename )
```

Return a new string that contains the path part of the filename.

Parameters

in	*filename*	file path and name

Returns

the path part of a filename without the name

See also

fl_filename_name(const char *filename)

11.16.3.17 fl_filename_relative() [1/2]

```
int fl_filename_relative (
            char * to,
            int tolen,
            const char * from )
```

Makes a filename relative to the current working directory.
Return the *from* path made relative to the working directory, similar to C++17 `std::filesystem::path::lexically_relative`. This function can also be called with a fourth argument for a user supplied *base* directory path
These conversions are purely lexical. They do not check that the paths exist, do not follow symlinks, and do not access the filesystem at all.
Path arguments must be absolute (start at the root directory) and must not contain `.` or `..` segments, or double separators. A single trailing separator is ok.
On Windows, path arguments must start with a drive name, e.g. `c:\`. Windows network paths and other special paths starting with a double separator are not supported (`\\cloud\drive\path`, `\\?\`, etc.) . Separators can be `\` and `/` and will be preserved. Newly created separators are alway the forward slash `/`.
On Windows and macOS, the path segment tests are case insensitive.
If the path can not be generated, *from* path is copied into the *to* buffer and 0 is returned.

```
#include <FL/filename.H>
[..]
fl_chdir("/var/tmp/somedir");       // set cwd to /var/tmp/somedir
[..]
char out[FL_PATH_MAX];
fl_filename_relative(out, sizeof(out), "/var/tmp/somedir/foo.txt");  // out="foo.txt",    return=1
fl_filename_relative(out, sizeof(out), "/var/tmp/foo.txt");          // out="../foo.txt", return=1
fl_filename_relative(out, sizeof(out), "foo.txt");                   // out="foo.txt",    return=0 (no
      change)
fl_filename_relative(out, sizeof(out), "./foo.txt");                 // out="./foo.txt",  return=0 (no
      change)
fl_filename_relative(out, sizeof(out), "../foo.txt");                // out="../foo.txt", return=0 (no
      change)
```

Parameters

out	*to*	resulting relative filename
in	*tolen*	size of the relative filename buffer
in	*from*	absolute filename

Returns

0 if no change, non zero otherwise

See also

fl_filename_relative(char *to, int tolen, const char *from, const char *base)

11.16.3.18 fl_filename_relative() [2/2]

```
int fl_filename_relative (
            char * to,
            int tolen,
            const char * from,
            const char * base )
```

Makes a filename relative to any other directory.

Parameters

out	*to*	resulting relative filepath
in	*tolen*	size of the relative filename buffer
in	*from*	absolute filepath
in	*base*	generate filepath relative to this absolute filepath

Returns

0 if no change, non zero otherwise

See also

fl_filename_relative(char *to, int tolen, const char *from)

11.16.3.19 fl_filename_relative_str() [1/2]

```
std::string fl_filename_relative_str (
            const std::string & from )
```

Makes a filename relative to the current working directory.

Parameters

in	*from*	file path and name

Returns

the new, relative filename

See also

fl_filename_relative(char *to, int tolen, const char *from)

11.16.3.20 fl_filename_relative_str() [2/2]

```
std::string fl_filename_relative_str (
            const std::string & from,
            const std::string & base )
```

Makes a filename relative to any directory.

Parameters

in	*from*	file path and name
in	*base*	relative to this absolute path

Returns

the new, relative filename

See also

fl_filename_relative(char *to, int tolen, const char *from, const char *base)

11.16.3.21 fl_filename_setext()

```
char * fl_filename_setext (
            char * buf,
```

```
int buflen,
const char * ext )
```

Replaces the extension in `buf` of max.

size `buflen` with the extension in `ext`.
If there's no '.' in `buf`, `ext` is appended.
If `ext` is NULL, behaves as if it were an empty string ("").

Example

```
#include <FL/filename.H>
[..]
char buf[FL_PATH_MAX] = "/path/myfile.cxx";
fl_filename_setext(buf, sizeof(buf), ".txt");      // buf[] becomes "/path/myfile.txt"
```

Returns

buf itself for calling convenience.

11.16.3.22 fl_filename_setext_str()

```
std::string fl_filename_setext_str (
            const std::string & filename,
            const std::string & new_extension )
```

Return a copy of the old filename with the new extension.

Parameters

in	*filename*	file path and name
in	*new_extension*	new filename extension, starts with a '.'

Returns

the new filename

See also

fl_filename_setext(char *to, int tolen, const char *ext)

11.16.3.23 fl_getcwd_str()

```
std::string fl_getcwd_str ( )
```

Cross-platform function to get the current working directory as a UTF-8 encoded value in an std::string.

Returns

the CWD encoded as UTF-8

11.16.3.24 fl_open_uri()

```
int fl_open_uri (
            const char * uri,
            char * msg,
            int msglen )
```

Opens the specified Uniform Resource Identifier (URI).
Uses an operating-system dependent program or interface. For URIs using the "ftp", "http", or "https" schemes, the system default web browser is used to open the URI, while "mailto" and "news" URIs are typically opened using the system default mail reader and "file" URIs are opened using the file system navigator.
On success, the (optional) msg buffer is filled with the command that was run to open the URI; on Windows, this will always be "open uri".
On failure, the msg buffer is filled with an English error message.

Note

Platform Specific Issues: Windows
With "file:" based URIs on Windows, you may encounter issues with anchors being ignored. Example: "file:///c:/some/index.html#anchor" may open in the browser without the "#anchor" suffix. The behavior seems to vary across different Windows versions. Workaround: open a link to a separate html file that redirects to the desired "file:" URI.

Example

```
#include <FL/filename.H>
[..]
char errmsg[512];
if ( !fl_open_uri("http://google.com/", errmsg, sizeof(errmsg)) ) {
    char warnmsg[768];
    sprintf(warnmsg, "Error: %s", errmsg);
    fl_alert(warnmsg);
}
```

Parameters

uri	The URI to open
msg	Optional buffer which contains the command or error message
msglen	Length of optional buffer

Returns

1 on success, 0 on failure

Chapter 12

Namespace Documentation

12.1 Fl Namespace Reference

Fl is the FLTK global namespace containing state information and global methods for the current application.

Namespaces

- namespace Pen

  FLTK Pen/Stylus/Tablet input driver API.

Classes

- struct TimeoutData

Enumerations

- enum Fl_Option {
  OPTION_ARROW_FOCUS = 0 , OPTION_VISIBLE_FOCUS , OPTION_DND_TEXT , OPTION_SHOW_TOOLTIPS ,
  OPTION_FNFC_USES_GTK , OPTION_FNFC_USES_ZENITY , OPTION_FNFC_USES_KDIALOG ,
  OPTION_PRINTER_USES_GTK ,
  OPTION_SHOW_SCALING , OPTION_SIMPLE_ZOOM_SHORTCUT , OPTION_LAST }

  Enumerator for global FLTK options.

Functions

- int abi_check (const int val=99999)

  Returns whether the runtime library ABI version is correct.
- int abi_version ()

  Returns the compiled-in value of the FL_ABI_VERSION constant.
- void add_check (Fl_Timeout_Handler, void ∗=0)

  FLTK will call this callback just before it flushes the display and waits for events.
- void add_clipboard_notify (Fl_Clipboard_Notify_Handler h, void ∗data=0)

  FLTK will call the registered callback whenever there is a change to the selection buffer or the clipboard.
- void add_fd (int fd, Fl_FD_Handler cb, void ∗=0)

  Adds file descriptor fd to listen to.
- void add_fd (int fd, int when, Fl_FD_Handler cb, void ∗=0)

  Adds file descriptor fd to listen to.
- void add_handler (Fl_Event_Handler ha)

  Install a function to parse unrecognized events.
- void add_handler (Fl_Event_Handler ha, Fl_Event_Handler before)

  Install a function to parse unrecognized events with less priority than another function.

- void add_idle (Fl_Idle_Handler cb, void *data=0)

  Adds a callback function that is called every time by Fl::wait() and also makes it act as though the timeout is zero (this makes Fl::wait() return immediately, so if it is in a loop it is called repeatedly, and thus the idle function is called repeatedly).
- void **add_idle** (Fl_Old_Idle_Handler cb)
- void add_system_handler (Fl_System_Handler h, void *data)

  Install a function to intercept system events.
- void add_timeout (double t, Fl_Timeout_Handler cb, void *data=0)

  Adds a one-shot timeout callback.
- int api_version ()

  Returns the compiled-in value of the FL_API_VERSION constant.
- int arg (int argc, char **argv, int &i)

  Parse a single switch from `argv`, starting at word `i`.
- void args (int argc, char **argv)

  Parse all command line switches matching standard FLTK options only.
- int args (int argc, char **argv, int &i, Fl_Args_Handler cb=0)

  Parse command line switches using the `cb` argument handler.
- int args_to_utf8 (int argc, char **&argv)

  Convert Windows commandline arguments to UTF-8.
- void awake ()

  Notifies the main GUI thread from a worker thread.
- int awake (Fl_Awake_Handler handler, void *user_data=nullptr)

  Schedules a callback to be executed by the main thread, then wakes up the main thread.
- void awake (void *message)

  Awake the main GUI thread and leave a message pointer.
- int awake_once (Fl_Awake_Handler handler, void *user_data=nullptr)

  Schedules a callback to be executed once by the main thread, then wakes up the main thread.
- void background (uchar, uchar, uchar)

  Changes fl_color(FL_BACKGROUND_COLOR) to the given color, and changes the gray ramp from 32 to 56 to black to white.
- void background2 (uchar, uchar, uchar)

  Changes the alternative background color.
- Fl_Widget * belowmouse ()

  Gets the widget that is below the mouse.
- void belowmouse (Fl_Widget *)

  Sets the widget that is below the mouse.
- bool box_bg (Fl_Boxtype)

  Returns whether the given boxtype draws its background.
- int box_border_radius_max ()

  Get the maximum border radius of all "rounded" boxtypes in pixels.
- void box_border_radius_max (int R)

  Set the maximum border radius of all "rounded" boxtypes in pixels.
- Fl_Color box_color (Fl_Color)

  Gets the drawing color to be used for the background of a box.
- int box_dh (Fl_Boxtype)

  Returns the height offset for the given boxtype.
- int box_dw (Fl_Boxtype)

  Returns the width offset for the given boxtype.
- int box_dx (Fl_Boxtype)

  Returns the X offset for the given boxtype.
- int box_dy (Fl_Boxtype)

Returns the Y offset for the given boxtype.

- int box_shadow_width ()

  Get the box shadow width of all "shadow" boxtypes in pixels.
- void box_shadow_width (int W)

  Set the box shadow width of all "shadow" boxtypes in pixels.
- bool cairo_autolink_context ()

  Gets the current autolink mode for Cairo support.
- void cairo_autolink_context (bool alink)

  When FLTK_HAVE_CAIRO is defined and cairo_autolink_context() is true, any current window dc is linked to a current Cairo context.
- cairo_t ∗ **cairo_cc** ()

  Gets the current Cairo context linked with a fltk window.
- void cairo_cc (cairo_t ∗c, bool own=false)

  Sets the current Cairo context to `c`.
- void cairo_flush (cairo_t ∗c)

  Flush Cairo drawings on Cairo context `c`.
- cairo_t ∗ cairo_make_current (Fl_Window ∗w)

  Provides a Cairo context for window wi.
- Fl_Callback_Reason callback_reason ()

  Give the reason for calling a callback.
- int check ()

  Same as Fl::wait(0).
- void clear_widget_pointer (Fl_Widget const ∗w)

  Clears a widget pointer in the watch list.
- int clipboard_contains (const char ∗type)

  Returns non 0 if the clipboard contains data matching `type`.
- int compose (int &del)

  Any text editing widget should call this for each FL_KEYBOARD event.
- void compose_reset ()

  If the user moves the cursor, be sure to call Fl::compose_reset().
- void copy (const char ∗stuff, int len, int destination=0, const char ∗type=Fl::clipboard_plain_text)

  Copies data to the selection buffer, the clipboard, or both.
- int **damage** ()

  If true then flush() will do something.
- void **damage** (int d)

  If true then flush() will do something.
- void default_atclose (Fl_Window ∗, void ∗)

  Default callback for window widgets.
- void delete_widget (Fl_Widget ∗w)

  Schedules a widget for deletion at the next call to the event loop.
- void disable_im ()

  Disables the system input methods facilities.
- void display (const char ∗)

  Sets the X or Wayland display to use for all windows.
- int dnd ()

  Initiate a Drag And Drop operation.
- int dnd_text_ops ()

  Gets whether drag and drop text operations are supported.
- void dnd_text_ops (int v)

  Sets whether drag and drop text operations are supported.
- void do_widget_deletion ()

Deletes widgets previously scheduled for deletion.

- int draw_box_active ()

  Determines if the currently drawn box is active or inactive.
- int draw_GL_text_with_textures ()

  returns whether whether OpenGL uses textures to draw all text.
- void draw_GL_text_with_textures (int val)

  sets whether OpenGL uses textures to draw all text.
- void enable_im ()

  Enables the system input methods facilities.
- int event ()

  Returns the last event that was processed.
- int **event_alt** ()

  Returns non-zero if the Alt key is pressed.
- int event_button ()

  Gets which particular mouse button caused the current event.
- int event_button1 ()

  Returns non-zero if mouse button 1 is currently held down.
- int event_button2 ()

  Returns non-zero if mouse button 2 is currently held down.
- int event_button3 ()

  Returns non-zero if mouse button 3 is currently held down.
- int event_button4 ()

  Returns non-zero if mouse button 4 is currently held down.
- int event_button5 ()

  Returns non-zero if mouse button 5 is currently held down.
- int event_buttons ()

  Returns the mouse buttons state bits; if non-zero, then at least one button is pressed now.
- int event_clicks ()

  Returns non zero if we had a double click event.
- void event_clicks (int i)

  Manually sets the number returned by Fl::event_clicks().
- void * event_clipboard ()

  During an FL_PASTE event of non-textual data, returns a pointer to the pasted data.
- const char * event_clipboard_type ()

  Returns the type of the pasted data during an FL_PASTE event.
- int **event_command** ()

  Returns non-zero if the FL_COMMAND key is pressed, either FL_CTRL or on OSX FL_META.
- int **event_ctrl** ()

  Returns non-zero if the Control key is pressed.
- Fl_Event_Dispatch **event_dispatch** ()

  Return the current event dispatch function.
- void event_dispatch (Fl_Event_Dispatch d)

  Set a new event dispatch function.
- int event_dx ()

  Returns the current horizontal mouse scrolling associated with the FL_MOUSEWHEEL event.
- float event_dx_f ()

  Returns the current horizontal mouse scrolling associated with the FL_MOUSEWHEEL event.
- int event_dy ()

  Returns the current vertical mouse scrolling associated with the FL_MOUSEWHEEL event.
- float event_dy_f ()

  Returns the current vertical mouse scrolling associated with the FL_MOUSEWHEEL event.

- int event_inside (const Fl_Widget ∗)

  Returns whether or not the mouse event is inside a given child widget.
- int event_inside (int, int, int, int)

  Returns whether or not the mouse event is inside the given rectangle.
- int event_is_click ()

  Returns non-zero if the mouse has not moved far enough and not enough time has passed since the last FL_PUSH or FL_KEYBOARD event for it to be considered a "drag" rather than a "click".
- void event_is_click (int i)

  Clears the value returned by Fl::event_is_click().
- int event_key ()

  Gets which key on the keyboard was last pushed.
- int event_key (int key)

  Returns true if the given `key` was held down (or pressed) during the last event.
- int event_length ()

  Returns the length of the text in Fl::event_text().
- int event_original_key ()

  Returns the keycode of the last key event, regardless of the NumLock state.
- int **event_shift** ()

  Returns non-zero if the Shift key is pressed.
- int event_state ()

  Returns the keyboard and mouse button states of the last event.
- int event_state (int mask)

  Returns non-zero if any of the passed event state bits are turned on.
- const char ∗ event_text ()

  Returns the text associated with the current event, including FL_PASTE or FL_DND_RELEASE events.
- int **event_x** ()

  Returns the mouse position of the event relative to the Fl_Window it was passed to.
- int event_x_root ()

  Returns the mouse position on the screen of the event.
- int **event_y** ()

  Returns the mouse position of the event relative to the Fl_Window it was passed to.
- int event_y_root ()

  Returns the mouse position on the screen of the event.
- Fl_Window ∗ first_window ()

  Returns the first top-level window in the list of shown() windows.
- void first_window (Fl_Window ∗)

  Sets the window that is returned by first_window().
- void flush ()

  Causes all the windows that need it to be redrawn and graphics forced out through the pipes.
- Fl_Widget ∗ focus ()

  Gets the current Fl::focus() widget.
- void focus (Fl_Widget ∗)

  Sets the widget that will receive FL_KEYBOARD events.
- void **foreground** (uchar, uchar, uchar)

  Changes fl_color(FL_FOREGROUND_COLOR).
- void free_color (Fl_Color i, int overlay=0)

  Frees the specified color from the colormap, if applicable.
- Fl_Box_Draw_F ∗ **get_boxtype** (Fl_Boxtype)

  Gets the current box drawing function for the specified box type.
- unsigned get_color (Fl_Color i)

  Returns the RGB value(s) for the given FLTK color index.

- void get_color (Fl_Color i, uchar &red, uchar &green, uchar &blue)

  Returns the RGB value(s) for the given FLTK color index.
- void get_color (Fl_Color i, uchar &red, uchar &green, uchar &blue, uchar &alpha)

  Returns the RGBA value(s) for the given FLTK color index.
- const char * get_font (Fl_Font)

  Gets the string for this face.
- const char * get_font_name (Fl_Font, int *attributes=0)

  Get a human-readable string describing the family of this face.
- int get_font_sizes (Fl_Font, int *&sizep)

  Return an array of sizes in `sizep`.
- int get_key (int key)

  Returns true if the given `key` is held down now.
- int get_mouse (int &X, int &Y)

  Return the number of the mouse-containing screen and computes the mouse screen coordinates.
- void get_system_colors ()

  Read the user preference colors from the system and use them to call Fl::foreground(), Fl::background(), and Fl::background2().
- int gl_visual (int, int *alist=0)

  This does the same thing as Fl::visual(int) but also requires OpenGL drawing to work.
- Fl_Window * grab ()

  Returns the window that currently receives all events.
- void **grab** (Fl_Window &win)

  See grab(Fl_Window)*
- void grab (Fl_Window *)

  Selects the window to grab.
- int **h** ()

  Returns the height in pixels of the main screen work area.
- int handle (int, Fl_Window *)

  Handle events from the window system.
- int handle_ (int, Fl_Window *)

  Handle events from the window system.
- int **has_check** (Fl_Timeout_Handler, void *=0)

  Returns 1 if the check exists and has not been called yet, 0 otherwise.
- int has_idle (Fl_Idle_Handler cb, void *data=0)

  Returns true if the specified idle callback is currently installed.
- int has_timeout (Fl_Timeout_Handler cb, void *data=0)

  Returns true if the timeout exists and has not been called yet.
- void hide_all_windows ()

  Hide all visible windows to make FLTK leave Fl::run().
- bool idle ()

  Returns whether at least one idle callback is currently set.
- int is_scheme (const char *name)

  Returns whether the current scheme is the given name.
- void keyboard_screen_scaling (int value)

  Controls the possibility to scale all windows by ctrl/+/-/0/ or cmd/+/-/0/.
- Fl_Event_Handler last_handler ()

  Returns the last function installed by a call to Fl::add_handler().
- int lock ()

  Acquire the global UI lock for FLTK.
- int menu_linespacing ()

  Gets the default line spacing used by menus.

- void menu_linespacing (int H)

  Sets the default line spacing used by menus.
- Fl_Window * modal ()

  Returns the top-most modal() window currently shown.
- Fl_Window * next_window (const Fl_Window *)

  Returns the next top-level window in the list of shown() windows.
- float normalized_screen_scale (int n)

  Gets the normalized GUI scaling factor of screen number `n`.
- void normalized_screen_scale (int n, float factor)

  Sets the normalized GUI scaling factor of screen number `n`.
- Fl_Timestamp now (double offset=0)

  Set a time stamp at this point in time with optional signed offset in seconds.
- bool option (Fl_Option opt)

  FLTK library options management.
- void option (Fl_Option opt, bool val)

  Override an option while the application is running.
- void own_colormap ()

  Makes FLTK use its `own colormap`.
- void paste (Fl_Widget &receiver)

  Backward compatibility only.
- void paste (Fl_Widget &receiver, int source, const char *type=Fl::clipboard_plain_text)

  Pastes the data from the selection buffer (`source` is 0) or the clipboard (`source` is 1) into `receiver`.
- int program_should_quit ()

  Returns non-zero when a request for program termination was received and accepted.
- void program_should_quit (int should_i)

  Indicate to the FLTK library whether a program termination request was received and accepted.
- Fl_Widget * pushed ()

  Gets the widget that is being pushed.
- void pushed (Fl_Widget *)

  Sets the widget that is being pushed.
- Fl_Widget * readqueue ()

  Reads the default callback queue and returns the first widget.
- int ready ()

  This is similar to Fl::check() except this does not call Fl::flush() or any callbacks, which is useful if your program is in a state where such callbacks are illegal.
- void **redraw** ()

  Redraws all widgets.
- void release_widget_pointer (Fl_Widget *&w)

  Releases a widget pointer from the watch list.
- int reload_scheme ()

  Called internally when setting a new scheme according to scheme name.
- void remove_check (Fl_Timeout_Handler, void *=0)

  Removes a check callback.
- void **remove_clipboard_notify** (Fl_Clipboard_Notify_Handler h)

  Stop calling the specified callback when there are changes to the selection buffer or the clipboard.
- void **remove_fd** (int)

  Removes a file descriptor handler.
- void **remove_fd** (int, int when)

  Removes a file descriptor handler.
- void remove_handler (Fl_Event_Handler h)

  Removes a previously added event handler.

- void remove_idle (Fl_Idle_Handler cb, void *data=0)

  Removes the specified idle callback, if it is installed.
- int remove_next_timeout (Fl_Timeout_Handler cb, void *data=0, void **data_return=0)

  Remove the next matching timeout callback and return its `data` pointer.
- void remove_system_handler (Fl_System_Handler h)

  Removes a previously added system event handler.
- void remove_timeout (Fl_Timeout_Handler cb, void *data=0)

  Remove one or more matching timeout callbacks from the timer queue.
- void repeat_timeout (double t, Fl_Timeout_Handler cb, void *data=0)

  Repeats a timeout callback from the expiration of the previous timeout, allowing for more accurate timing.
- int run ()

  Calls Fl::wait()repeatedly as long as any windows are displayed.
- const char * **scheme** ()

  *See void scheme(const char *name)*
- int scheme (const char *name)

  Sets the current widget scheme.
- int screen_count ()

  Gets the total count of available screens.
- void screen_dpi (float &h, float &v, int n=0)

  Gets the screen resolution in dots-per-inch for the given screen.
- Fl_Screen_Driver * **screen_driver** ()

  Returns a pointer to the unique Fl_Screen_Driver object of the platform.
- int screen_num (int x, int y)

  Gets the screen number of a screen that contains the specified screen position `x`, `y`.
- int screen_num (int x, int y, int w, int h)

  Gets the screen number of the screen which intersects the most with the rectangle defined by `x`, `y`, `w`, `h`.
- float screen_scale (int n)

  Gets the GUI scaling factor of screen number `n`.
- void screen_scale (int n, float factor)

  Sets the GUI scaling factor of screen number `n`.
- int screen_scaling_supported ()

  Returns whether scaling factors are supported by this platform.
- void screen_work_area (int &X, int &Y, int &W, int &H)

  Gets the bounding box of the work area of the screen that contains the mouse pointer.
- void screen_work_area (int &X, int &Y, int &W, int &H, int mx, int my)

  Gets the bounding box of the work area of a screen that contains the specified screen position `mx`, `my`.
- void screen_work_area (int &X, int &Y, int &W, int &H, int n)

  Gets the bounding box of the work area of the given screen.
- void screen_xywh (int &X, int &Y, int &W, int &H)

  Gets the bounding box of a screen that contains the mouse pointer.
- void screen_xywh (int &X, int &Y, int &W, int &H, int mx, int my)

  Gets the bounding box of a screen that contains the specified screen position `mx`, `my`.
- void screen_xywh (int &X, int &Y, int &W, int &H, int mx, int my, int mw, int mh)

  Gets the screen bounding rect for the screen which intersects the most with the rectangle defined by `mx`, `my`, `mw`, `mh`.
- void screen_xywh (int &X, int &Y, int &W, int &H, int n)

  Gets the screen bounding rect for the given screen.
- int scrollbar_size ()

  Gets the default scrollbar size used by Fl_Browser_, Fl_Help_View, Fl_Scroll, and Fl_Text_Display widgets.
- void scrollbar_size (int W)

  Sets the default scrollbar size that is used by the Fl_Browser_, Fl_Help_View, Fl_Scroll, and Fl_Text_Display widgets.
- double seconds_between (Fl_Timestamp &back, Fl_Timestamp &further_back)

Return the time in seconds between two time stamps.

- double seconds_since (Fl_Timestamp &then)

  Return the time in seconds between now and a previously taken time stamp.
- void selection (Fl_Widget &owner, const char *, int len)

  Changes the current selection.
- Fl_Widget * selection_owner ()

  back-compatibility only: Gets the widget owning the current selection
- void selection_owner (Fl_Widget *)

  Back-compatibility only: The single-argument call can be used to move the selection to another widget or to set the owner to NULL, without changing the actual text of the selection.
- int selection_to_clipboard ()

  Returns the current selection_to_clipboard mode.
- void selection_to_clipboard (int mode)

  Copies selections on X11 directly to the clipboard if enabled.
- void **set_abort** (Fl_Abort_Handler f)

  For back compatibility, sets the void Fl::fatal handler callback.
- void set_atclose (Fl_Atclose_Handler f)

  For back compatibility, sets the Fl::atclose handler callback.
- void set_box_color (Fl_Color)

  Sets the drawing color for the box that is currently drawn.
- void set_boxtype (Fl_Boxtype, Fl_Box_Draw_F *, uchar, uchar, uchar, uchar, Fl_Box_Draw_Focus_F *=NULL)

  Sets the function to call to draw a specific box type.
- void **set_boxtype** (Fl_Boxtype, Fl_Boxtype from)

  Copies the from boxtype.
- void set_color (Fl_Color i, unsigned c)

  Sets an entry in the fl_color index table.
- void set_color (Fl_Color, uchar, uchar, uchar)

  Sets an entry in the fl_color index table.
- void set_color (Fl_Color, uchar, uchar, uchar, uchar)

  Sets an entry in the fl_color index table.
- void set_font (Fl_Font, const char *)

  Changes a face.
- void **set_font** (Fl_Font, Fl_Font)

  Copies one face to another.
- Fl_Font set_fonts (const char *=0)

  FLTK will open the display, and add every fonts on the server to the face table.
- void **set_labeltype** (Fl_Labeltype, Fl_Label_Draw_F *, Fl_Label_Measure_F *)

  Sets the functions to call to draw and measure a specific labeltype.
- void **set_labeltype** (Fl_Labeltype, Fl_Labeltype from)

  Sets the functions to call to draw and measure a specific labeltype.
- int **system** (const char *command)

  Run a command line on the computer.
- Fl_System_Driver * **system_driver** ()

  Returns a pointer to the unique Fl_System_Driver object of the platform.
- int test_shortcut (Fl_Shortcut)

  Tests the current event, which must be an FL_KEYBOARD or FL_SHORTCUT, against a shortcut value (described in Fl_Button).
- void * thread_message ()

  Returns the last message sent by a child thread.
- long ticks_between (Fl_Timestamp &back, Fl_Timestamp &further_back)

Return the time in ticks (60Hz) between two time stamps.

- long ticks_since (Fl_Timestamp &then)

  Return the time in ticks (60Hz) between now and a previously taken time stamp.
- std::vector< TimeoutData > timeout_list ()

  Return a list of all currently running timeouts.
- void unlock ()

  Release the global UI lock set by Fl::lock().
- int use_high_res_GL ()

  Returns whether GL windows should be drawn at high resolution on Apple computers with retina displays.
- void use_high_res_GL (int val)

  Sets whether GL windows should be drawn at high resolution on Apple computers with retina displays.
- double version ()

  Returns the compiled-in value of the FL_VERSION constant.
- int visible_focus ()

  Gets or sets the visible keyboard focus on buttons and other non-text widgets.
- void visible_focus (int v)

  Gets or sets the visible keyboard focus on buttons and other non-text widgets.
- int visual (int)

  Selects a visual so that your graphics are drawn correctly.
- int **w** ()

  Returns the width in pixels of the main screen work area.
- int wait ()

  Waits until "something happens" and then returns.
- double wait (double time)

  Waits a maximum of `time_to_wait` seconds or until "something happens".
- void watch_widget_pointer (Fl_Widget *&w)

  Adds a widget pointer to the widget watch list.
- int **x** ()

  Returns the leftmost x coordinate of the main screen work area.
- int **y** ()

  Returns the topmost y coordinate of the main screen work area.

Variables

- void(* atclose)(Fl_Window *, void *) = Fl::default_atclose

  Back compatibility: default window callback handler.
- char const *const **clipboard_image** = "image"

  Denotes image data.
- char const *const **clipboard_plain_text** = "text/plain"

  Denotes plain textual data.
- void(* error)(const char *,...) = Fl_System_Driver::error

  FLTK calls Fl::error() to output a normal error message.
- void(* fatal)(const char *,...) = Fl_System_Driver::fatal

  FLTK calls Fl::fatal() to output a fatal error message.
- const char *const help = helpmsg+13

  Usage string displayed if Fl::args() detects an invalid argument.
- void(* warning)(const char *,...) = Fl_System_Driver::warning

  FLTK calls Fl::warning() to output a warning message.

12.1.1 Detailed Description

Fl is the FLTK global namespace containing state information and global methods for the current application. FLTK private global variables and functions.

12.1.2 Enumeration Type Documentation

12.1.2.1 Fl_Option

`enum Fl::Fl_Option`

Enumerator for global FLTK options.

These options can be set system wide, per user, or for the running application only.

See also

Fl::option(Fl_Option, bool)

Fl::option(Fl_Option)

Enumerator

OPTION_ARROW_FOCUS	When switched on, moving the text cursor beyond the start or end of a text in a text widget will change focus to the next text widget. (This is considered 'old' behavior) When switched off (default), the cursor will stop at the end of the text. Pressing Tab or Ctrl-Tab will advance the keyboard focus. See also: Fl_Input_::tab_nav()
OPTION_VISIBLE_FOCUS	If visible focus is switched on (default), FLTK will draw a dotted rectangle inside the widget that will receive the next keystroke. If switched off, no such indicator will be drawn and keyboard navigation is disabled.
OPTION_DND_TEXT	If text drag-and-drop is enabled (default), the user can select and drag text from any text widget. If disabled, no dragging is possible, however dropping text from other applications still works.
OPTION_SHOW_TOOLTIPS	If tooltips are enabled (default), hovering the mouse over a widget with a tooltip text will open a little tooltip window until the mouse leaves the widget. If disabled, no tooltip is shown.
OPTION_FNFC_USES_GTK	When switched on (default), Fl_Native_File_Chooser runs GTK file dialogs if the GTK library is available on the platform (linux/unix only). When switched off, GTK file dialogs aren't used even if the GTK library is available.
OPTION_FNFC_USES_ZENITY	Meaningful for the Wayland/X11 platform only. When switched on, the library uses a Zenity-based file dialog. When switched off (default), no zenity-based file dialog is used.
OPTION_FNFC_USES_KDIALOG	Meaningful for the Wayland/X11 platform only. When switched on, the library uses a kdialog-based file dialog if command 'kdialog' is available on the running system. When switched off (default), no kdialog-based file dialog is used.
OPTION_PRINTER_USES_GTK	When switched on (default), Fl_Printer runs the GTK printer dialog if the GTK library is available on the platform (linux/unix only). When switched off, the GTK printer dialog isn't used even if the GTK library is available.
OPTION_SHOW_SCALING	When switched on (default), the library shows in a transient yellow window the zoom factor value. When switched off, no such window gets displayed.
OPTION_SIMPLE_ZOOM_SHORTCUT	When switched on and when the keyboard in use has '+' in the shifted position of its key, pressing that key and ctrl triggers the zoom-in operation. When switched off (default), the zoom-in operation requires that also the shift key is pressed. Under macOS, this option has no effect with some keyboard layouts because the OS itself generates = followed by + when pressing and the '=|+' key without pressing shift.
OPTION_LAST	For internal use only.

12.1.3 Function Documentation

12.1.3.1 abi_check()

```
int Fl::abi_check (
            const int val = 99999 )  [inline]
```

Returns whether the runtime library ABI version is correct.
This enables you to check the ABI version of the linked FLTK library at runtime.
Returns 1 (true) if the compiled ABI version (in the header files) and the linked library ABI version (used at runtime) are the same, 0 (false) otherwise.
Argument `val` can be used to query a particular library ABI version. Use for instance 10303 to query if the runtime library is compatible with FLTK ABI version 1.3.3. This is rarely useful.
The default `val` argument is FL_ABI_VERSION, which checks the version defined at configure time (i.e. in the header files at program compilation time) against the linked library version used at runtime. This is particularly useful if you linked with a shared object library, but it also concerns static linking.

See also

Fl::abi_version()

12.1.3.2 abi_version()

```
int Fl::abi_version ( )  [extern]
```

Returns the compiled-in value of the FL_ABI_VERSION constant.
This is useful for checking the version of a shared library.

12.1.3.3 add_check()

```
void Fl::add_check (
            Fl_Timeout_Handler cb,
            void * argp = 0 )  [extern]
```

FLTK will call this callback just before it flushes the display and waits for events.
This is different than an idle callback because it is only called once, then FLTK calls the system and tells it not to return until an event happens.
This can be used by code that wants to monitor the application's state, such as to keep a display up to date. The advantage of using a check callback is that it is called only when no events are pending. If events are coming in quickly, whole blocks of them will be processed before this is called once. This can save significant time and avoid the application falling behind the events.
Sample code:

```
bool state_changed; // anything that changes the display turns this on

void callback(void*) {
 if (!state_changed) return;
 state_changed = false;
 do_expensive_calculation();
 widget->redraw();
}

main() {
 Fl::add_check(callback);
 return Fl::run();
}
```

12.1.3.4 add_fd() [1/2]

```
void Fl::add_fd (
            int fd,
            Fl_FD_Handler cb,
            void * d = 0 )  [extern]
```

Adds file descriptor fd to listen to.
See Fl::add_fd(int fd, int when, Fl_FD_Handler cb, void∗ = 0) for details

12.1.3.5 add_fd() [2/2]

```
void Fl::add_fd (
            int fd,
            int when,
            Fl_FD_Handler cb,
            void * d = 0 )  [extern]
```

Adds file descriptor fd to listen to.
When the fd becomes ready for reading Fl::wait() will call the callback and then return. The callback is passed the fd and the arbitrary void* argument.
This version takes a when bitfield, with the bits FL_READ, FL_WRITE, and FL_EXCEPT defined, to indicate when the callback should be done.
There can only be one callback of each type for a file descriptor. Fl::remove_fd() gets rid of *all* the callbacks for a given file descriptor.
Under UNIX/Linux/macOS *any* file descriptor can be monitored (files, devices, pipes, sockets, etc.). Due to limitations in Microsoft Windows, Windows applications can only monitor sockets.
Under macOS, Fl::add_fd() opens the display if that's not been done before.

12.1.3.6 add_idle()

```
void Fl::add_idle (
            Fl_Idle_Handler cb,
            void * data = 0 )  [extern]
```

Adds a callback function that is called every time by Fl::wait() and also makes it act as though the timeout is zero (this makes Fl::wait() return immediately, so if it is in a loop it is called repeatedly, and thus the idle function is called repeatedly).
The idle function can be used to get background processing done.
You can have multiple idle callbacks. If this is the case, then all idle callbacks are called in turn. Each idle callback should return after it has done **some** work to let the next idle callback or the FLTK event loop continue processing.
To remove an idle callback use Fl::remove_idle().
Fl::wait() and Fl::check() call idle callbacks, but Fl::ready() does not.
The idle callback can call any FLTK functions, including Fl::wait(), Fl::check(), and Fl::ready().
FLTK will not recursively call the idle callback.

Parameters

in	*cb*	your idle callback
in	*data*	an arbitrary data value provided to your callback

12.1.3.7 add_timeout()

```
void Fl::add_timeout (
            double time,
            Fl_Timeout_Handler cb,
            void * data = 0 )  [extern]
```

Adds a one-shot timeout callback.
The callback function `cb` will be called by Fl::wait() at `time` seconds after this function is called. The callback function must have the signature Fl_Timeout_Handler. The optional `data` argument is passed to the callback (default: NULL).
The timer is removed from the timer queue before the callback function is called. It is safe to reschedule the timeout inside the callback function.
You can have multiple timeout callbacks, even the same timeout callback with different timeout values and/or different `data` values. They are all considered different timer objects.
To remove a timeout while it is active (pending) use Fl::remove_timeout().
If you need more accurate, repeated timeouts, use Fl::repeat_timeout() to reschedule the subsequent timeouts. Please see Fl::repeat_timeout() for an example.
Since version 1.4, a timeout can be started from a child thread under the condition that the call to Fl::add_timeout is wrapped in Fl::lock() and Fl::unlock().

Parameters

in	*time*	delta time in seconds until the timer expires
in	*cb*	callback function
in	*data*	optional user data (default: NULL)

See also

Fl_Timeout_Handler

Fl::repeat_timeout(double time, Fl_Timeout_Handler cb, void *data)

Fl::remove_timeout(Fl_Timeout_Handler cb, void *data)

Fl::has_timeout(Fl_Timeout_Handler cb, void *data)

12.1.3.8 api_version()

```
int Fl::api_version ( ) [extern]
```

Returns the compiled-in value of the FL_API_VERSION constant.
This is useful for checking the version of a shared library.

12.1.3.9 arg()

```
int Fl::arg (
            int argc,
            char ** argv,
            int & i ) [extern]
```

Parse a single switch from `argv`, starting at word `i`.
Returns the number of words eaten (1 or 2, or 0 if it is not recognized) and adds the same value to `i`.
This is the default argument handler used internally by Fl::args(...), but you can use this function if you prefer to step through the standard FLTK switches yourself.
All standard FLTK switches except -bg2 may be abbreviated to just one letter and case is ignored:

- -bg color or -background color
  Sets the background color using Fl::background().
- -bg2 color or -background2 color
  Sets the secondary background color using Fl::background2().
- -display host:n.n
  Sets the X display to use; this option is silently ignored under Windows and MacOS.
- -dnd and -nodnd
  Enables or disables drag and drop text operations using Fl::dnd_text_ops().
- -fg color or -foreground color
  Sets the foreground color using Fl::foreground().
- -geometry WxH+X+Y
  Sets the initial window position and size according to the standard X geometry string.
- -iconic
  Iconifies the window using Fl_Window::iconize().
- -kbd and -nokbd
  Enables or disables visible keyboard focus for non-text widgets using Fl::visible_focus().
- -name string
  Sets the window class using Fl_Window::xclass().

- -scaling_factor f
  Sets the user selected application scaling factor. The valid range for f is 0.25 to 4.0, and the default is 1.0.
- -scheme string
  Sets the widget scheme using Fl::scheme().
- -title string
  Sets the window title using Fl_Window::label().
- -tooltips and -notooltips
  Enables or disables tooltips using Fl_Tooltip::enable().

Color values are commonly given as three digit or six digit hex numbers.

- The order of fg, bg, and bg2 in the command line does not matter
- There is no way at the moment to set the selection color.
- Setting the bg2 color also changes the fg color to have sufficient contrast
- Explicitly setting fg color overrides the bg2/contrast constraint.
- Setting the bg color will update the color lookup table for the gray ramp, so color index values can stay the same for all apps, it's just mapped to different RGB values.
- The calculation of the gray ramp is only based on the bg color, so there is no way at the moment to create an inverted (dark mode) ramp.
- Consequently, setting bg to black creates a an all-black ramp, setting a somewhat dark bg color creates a extremely dark ramp.
- Setting the bg has no influence on bg2 or fg.

If your program requires other switches in addition to the standard FLTK options, you will need to pass your own argument handler to Fl::args(int,char∗∗,int&,Fl_Args_Handler) explicitly.

See also

fl_parse_color(const char∗ p, uchar& r, uchar& g, uchar& b) to see how color values can be defined

12.1.3.10 args() [1/2]

```
void Fl::args (
            int argc,
            char ** argv )  [extern]
```

Parse all command line switches matching standard FLTK options only.
It parses all the switches, and if any are not recognized it calls Fl::abort(Fl::help), i.e. unlike the long form, an unrecognized switch generates an error message and causes the program to exit.

12.1.3.11 args() [2/2]

```
int Fl::args (
            int argc,
            char ** argv,
            int & i,
            Fl_Args_Handler cb = 0 )  [extern]
```

Parse command line switches using the `cb` argument handler.
Returns 0 on error, or the number of words processed.
FLTK provides this as an *entirely optional* command line switch parser. You don't have to call it if you don't want to. Everything it can do can be done with other calls to FLTK.
To use the switch parser, call Fl::args(...) near the start of your program. This does **not** open the display, instead switches that need the display open are stashed into static variables. Then you **must** display your first window by calling `window->show(argc,argv)`, which will do anything stored in the static variables.

Providing an argument handler callback `cb` lets you define your own switches. It is called with the same `argc` and `argv`, and with `i` set to the index of the switch to be processed. The `cb` handler should return zero if the switch is unrecognized, and not change `i`. It should return non-zero to indicate the number of words processed if the switch is recognized, i.e. 1 for just the switch, and more than 1 for the switch plus associated parameters. `i` should be incremented by the same amount.
The `cb` handler is called **before** any other tests, so *you can also override any standard FLTK switch* (this is why FLTK can use very short switches instead of the long ones all other toolkits force you to use). See Fl::arg() for descriptions of the standard switches.
On return `i` is set to the index of the first non-switch. This is either:

- The first word that does not start with '-'.
- The word '-' (used by many programs to name stdin as a file)
- The first unrecognized switch (return value is 0).
- `argc`

The return value is `i` unless an unrecognized switch is found, in which case it is zero. If your program takes no arguments other than switches you should produce an error if the return value is less than `argc`.
A usage string is displayed if Fl::args() detects an invalid argument on the command-line. You can change the message by setting the Fl::help pointer.
A very simple command line parser can be found in `examples/howto-parse-args.cxx`
The simpler Fl::args(int argc, char **argv) form is useful if your program does not have command line switches of its own.

12.1.3.12 args_to_utf8()

```
int Fl::args_to_utf8 (
            int argc,
            char **& argv )  [extern]
```

Convert Windows commandline arguments to UTF-8.

Note

> This function does nothing on other (non-Windows) platforms, hence you may call it on all platforms or only on Windows by using platform specific code like `'#ifdef _WIN32'` etc. - it's your choice. Calling it on other platforms returns quickly w/o wasting much CPU time.

This function *must be called* ***on Windows platforms*** in `main()` before the array `argv` is used if your program uses any commandline argument strings (these should be UTF-8 encoded). This applies also to standard FLTK commandline arguments like "-name" (class name) and "-title" (window title in the title bar).
Unfortunately Windows **neither** provides commandline arguments in UTF-8 encoding **nor** as Windows "Wide Character" strings in the standard `main()` and/or the Windows specific `WinMain()` function.
On Windows platforms (no matter which build system) this function calls a Windows specific function to retrieve commandline arguments as Windows "Wide Character" strings, converts these strings to an internally allocated buffer (or multiple buffers) and returns the result in `argv`. For implementation details please refer to the source code; however these details may be changed in the future.
Note that `argv` is provided by reference so it can be overwritten.
In the recommended simple form the function overwrites the variable `argv` and allocates a new array of strings pointed to by `argv`. You may use this form on all platforms and it is as simple as adding one line to old programs to make them work with international (UTF-8) commandline arguments.

```
int main(int argc, char **argv) {
  Fl::args_to_utf8(argc, argv);   // add this line
  // ... use argc and argv, e.g. for commandline parsing
  window->show(argc, argv);
  return Fl::run();
}
```

For an example see 'examples/howto-parse-args.cxx' in the FLTK sources.
If you want to retain the original `argc` and `argv` variables the following slightly longer and more complicated code works as well on all platforms.

```
int main(int argc, char **argv) {
  char **argvn = argv;            // must copy argv to work on all platforms
  int argcn = Fl::args_to_utf8(argc, argvn);
```

```
  // ... use argcn and argvn, e.g. for commandline parsing
  window->show(argcn, argvn);
  return Fl::run();
}
```

Parameters

in	*argc*	used only on non-Windows platforms
out	*argv*	modified only on Windows platforms

Returns

argument count (always the same as argc)

Since

1.4.0

12.1.3.13 background()

```
void Fl::background (
            uchar r,
            uchar g,
            uchar b )  [extern]
```

Changes fl_color(FL_BACKGROUND_COLOR) to the given color, and changes the gray ramp from 32 to 56 to black to white.
These are the colors used as backgrounds by almost all widgets and used to draw the edges of all the boxtypes.

12.1.3.14 background2()

```
void Fl::background2 (
            uchar r,
            uchar g,
            uchar b )  [extern]
```

Changes the alternative background color.
This color is used as a background by Fl_Input and other text widgets.
This call may change fl_color(FL_FOREGROUND_COLOR) if it does not provide sufficient contrast to FL_↩ BACKGROUND2_COLOR.

12.1.3.15 box_bg()

```
bool Fl::box_bg (
            Fl_Boxtype bt )  [extern]
```

Returns whether the given boxtype draws its background.
FL_NO_BOX and all FL_∗_FRAME boxtypes need their parent to draw the background rather than just calling redraw. This is used internally for some widgets to ensure that the background is fully drawn when, for instance, only the focus box is changed.

Parameters

in	*bt*	Boxtype

Returns

true if the box draws a solid background,
false if it is at least partially transparent.

Since

1.5.0

12.1.3.16 box_border_radius_max() [1/2]

```
int Fl::box_border_radius_max ( ) [extern]
```

Get the maximum border radius of all "rounded" boxtypes in pixels.

Since

1.4.0

12.1.3.17 box_border_radius_max() [2/2]

```
void Fl::box_border_radius_max (
            int R ) [extern]
```

Set the maximum border radius of all "rounded" boxtypes in pixels.
Must be at least 5, default = 15.

Note

This does **not** apply to the "round" boxtypes which have really round sides (i.e. composed of half circles) as opposed to "rounded" boxtypes that have only rounded corners with a straight border between corners.

The box border radius of "rounded" boxtypes is typically calculated as about 2/5 of the box height or width, whichever is smaller. The upper limit can be set by this method for all "rounded" boxtypes.

Since

1.4.0

12.1.3.18 box_color()

```
Fl_Color Fl::box_color (
            Fl_Color c ) [extern]
```

Gets the drawing color to be used for the background of a box.
This method is only useful inside box drawing code. It returns the color to be used, either fl_inactive(c) if the widget is !active_r() or `c` otherwise.

12.1.3.19 box_dh()

```
int Fl::box_dh (
            Fl_Boxtype t ) [extern]
```

Returns the height offset for the given boxtype.

See also

box_dy().

12.1.3.20 box_dw()

```
int Fl::box_dw (
            Fl_Boxtype t ) [extern]
```

Returns the width offset for the given boxtype.

See also

box_dy().

12.1.3.21 box_dx()

```
int Fl::box_dx (
            Fl_Boxtype t )  [extern]
```

Returns the X offset for the given boxtype.

See also

box_dy()

12.1.3.22 box_dy()

```
int Fl::box_dy (
            Fl_Boxtype t )  [extern]
```

Returns the Y offset for the given boxtype.
These functions return the offset values necessary for a given boxtype, useful for computing the area inside a box's borders, to prevent overdrawing the borders.
For instance, in the case of a boxtype like FL_DOWN_BOX where the border width might be 2 pixels all around, the above functions would return 2, 2, 4, and 4 for box_dx, box_dy, box_dw, and box_dh respectively.
An example to compute the area inside a widget's box():

```
int X = yourwidget->x() + Fl::box_dx(yourwidget->box());
int Y = yourwidget->y() + Fl::box_dy(yourwidget->box());
int W = yourwidget->w() - Fl::box_dw(yourwidget->box());
int H = yourwidget->h() - Fl::box_dh(yourwidget->box());
```

These functions are mainly useful in the draw() code for deriving custom widgets, where one wants to avoid drawing over the widget's own border box().

12.1.3.23 box_shadow_width() [1/2]

```
int Fl::box_shadow_width ( )  [extern]
```

Get the box shadow width of all "shadow" boxtypes in pixels.

Since

1.4.0

12.1.3.24 box_shadow_width() [2/2]

```
void Fl::box_shadow_width (
            int W )  [extern]
```

Set the box shadow width of all "shadow" boxtypes in pixels.
Must be at least 1, default = 3. There is no upper limit.

Since

1.4.0

12.1.3.25 check()

```
int Fl::check ( )  [extern]
```

Same as Fl::wait(0).
Calling this during a big calculation will keep the screen up to date and the interface responsive:

```
while (!calculation_done()) {
  calculate();
  Fl::check();
  if (user_hit_abort_button()) break;
}
```

This returns non-zero if any windows are displayed, and 0 if no windows are displayed (this is likely to change in future versions of FLTK).

12.1.3.26 display()

```
void Fl::display (
            const char * d )  [extern]
```

Sets the X or Wayland display to use for all windows.
This sets the environment variable $DISPLAY or $WAYLAND_DISPLAY to the passed string, so this only works before you show() the first window or otherwise open the display.
This does nothing on other platforms.

12.1.3.27 dnd_text_ops() [1/2]

```
int Fl::dnd_text_ops ( )  [inline]
```

Gets whether drag and drop text operations are supported.
This returns whether selected text can be dragged from text fields or dragged within a text field as a cut/paste shortcut.

12.1.3.28 dnd_text_ops() [2/2]

```
void Fl::dnd_text_ops (
            int v )  [inline]
```

Sets whether drag and drop text operations are supported.
This specifically affects whether selected text can be dragged from text fields or dragged within a text field as a cut/paste shortcut.

12.1.3.29 draw_box_active()

```
int Fl::draw_box_active ( )  [extern]
```

Determines if the currently drawn box is active or inactive.
If inactive, the box color should be changed to the inactive color.

See also

Fl::box_color(Fl_Color c)

12.1.3.30 draw_GL_text_with_textures() [1/2]

```
int Fl::draw_GL_text_with_textures ( )  [extern]
```

returns whether whether OpenGL uses textures to draw all text.
Default is yes.

See also

draw_GL_text_with_textures(int val)

Version

1.4.0

12.1.3.31 draw_GL_text_with_textures() [2/2]

```
void Fl::draw_GL_text_with_textures (
            int val )  [extern]
```

sets whether OpenGL uses textures to draw all text.
By default, FLTK draws OpenGL text using textures, if the necessary hardware support is available. Call `Fl::draw_GL_text_with_textures(0)` once in your program before the first call to gl_font() to have FLTK draw instead OpenGL text using a legacy, platform-dependent procedure. It's recommended not to deactivate textures under the MacOS platform because the MacOS legacy procedure is extremely rudimentary.

Parameters

val	use 0 to prevent FLTK from drawing GL text with textures

See also

gl_texture_pile_height(int max)

Version

1.4.0

12.1.3.32 flush()

```
void Fl::flush ( ) [extern]
```

Causes all the windows that need it to be redrawn and graphics forced out through the pipes.
This is what wait() does before looking for events.
Note: in multi-threaded applications you should only call Fl::flush() from the main thread. If a child thread needs to trigger a redraw event, it should instead call Fl::awake() to get the main thread to process the event queue.

12.1.3.33 get_system_colors()

```
void Fl::get_system_colors ( ) [extern]
```

Read the user preference colors from the system and use them to call Fl::foreground(), Fl::background(), and Fl::background2().
This is done by Fl_Window::show(argc,argv) before applying the -fg and -bg switches.
On X this reads some common values from the Xdefaults database. KDE users can set these values by running the "krdb" program, and newer versions of KDE set this automatically if you check the "apply style to other X programs" switch in their control panel.

12.1.3.34 gl_visual()

```
int Fl::gl_visual (
            int ,
            int * alist = 0 ) [extern]
```

This does the same thing as Fl::visual(int) but also requires OpenGL drawing to work.
This *must* be done if you want to draw in normal windows with OpenGL with gl_start() and gl_end(). It may be useful to call this so your X windows use the same visual as an Fl_Gl_Window, which on some servers will reduce colormap flashing.
See Fl_Gl_Window for a list of additional values for the argument.

12.1.3.35 has_idle()

```
int Fl::has_idle (
            Fl_Idle_Handler cb,
            void * data = 0 ) [extern]
```

Returns true if the specified idle callback is currently installed.
An idle callback matches the request only if `data` matches the `data` argument when the callback was installed. There is no "wildcard" search.

Parameters

in	*cb*	idle callback in question
in	*data*	optional data. Default: zero / nullptr.

Returns

Whether the given callback `cb` is queued with `data`.

Return values

1	The callback is currently in the callback queue.
0	The callback is not queued, or `data` doesn't match.

12.1.3.36 has_timeout()

```
int Fl::has_timeout (
            Fl_Timeout_Handler cb,
            void * data = 0 )  [extern]
```

Returns true if the timeout exists and has not been called yet.
Both arguments `cb` and `data` must match with at least one timer in the queue of active timers to return true (1).

Note

It is a known inconsistency that Fl::has_timeout() does not use the `data` argument as a wildcard (match all) if it is zero (NULL) which Fl::remove_timeout() does. This is so for backwards compatibility with FLTK 1.3.x. Therefore using 0 (zero, NULL) as the timeout `data` value is discouraged unless you're sure that you don't need to use `Fl::has_timeout(callback, (void *)0);` or `Fl::remove_timeout(callback, (void *)0);`.

Parameters

in	*cb*	Timer callback
in	*data*	User data

Returns

whether the timer was found in the queue

Return values

0	not found
1	found

12.1.3.37 hide_all_windows()

```
void Fl::hide_all_windows ( )  [extern]
```

Hide all visible windows to make FLTK leave Fl::run().
Fl:run() will run as long as there are visible windows. Call Fl::hide_all_windows() to hide (close) all currently shown (visible) windows, effectively terminating the Fl::run() loop.

See also

Fl::run()

Since

1.4.0

12.1.3.38 idle()

```
bool Fl::idle ( ) [extern]
```

Returns whether at least one idle callback is currently set.
`true` means that at least one callback is currently queued, but not necessarily active. While a callback is being executed, it is also counted as "set" unless (i.e. before) it removes itself from the idle callback queue (ring).

Returns

whether an idle callback is currently set

Return values

true	At least one idle callback is currently set.
false	No idle callback is currently set.

12.1.3.39 is_scheme()

```
int Fl::is_scheme (
            const char * name ) [inline]
```

Returns whether the current scheme is the given name.
This is a fast inline convenience function to support scheme-specific code in widgets, e.g. in their draw() methods, if required.
Use a valid scheme name, not `NULL` (although `NULL` is allowed, this is not a useful argument - see below).
If Fl::scheme() has not been set or has been set to the default scheme ("none" or "base"), then this will always return 0 regardless of the argument, because Fl::scheme() is `NULL` in this case.

Note

The stored scheme name is always lowercase, and this method will do a case-sensitive compare, so you **must** provide a lowercase string to return the correct value. This is intentional for performance reasons.

Example:

```
if (Fl::is_scheme("gtk+")) { your_code_here(); }
```

Parameters

in	*name*	**lowercase** string of requested scheme name.

Returns

1 if the given scheme is active, 0 otherwise.

See also

Fl::scheme(const char *name)

12.1.3.40 menu_linespacing() [1/2]

```
int Fl::menu_linespacing ( ) [extern]
```

Gets the default line spacing used by menus.

Returns

The default line spacing, in pixels.

Since

1.4.0

12.1.3.41 menu_linespacing() [2/2]

```
void Fl::menu_linespacing (
            int H )  [extern]
```

Sets the default line spacing used by menus.
Default is 4.

Parameters

in	*H*	The new default line spacing between menu items, in pixels.

Since

1.4.0

12.1.3.42 now()

```
Fl_Timestamp Fl::now (
            double offset = 0 )  [extern]
```

Set a time stamp at this point in time with optional signed offset in seconds.
The time stamp is an opaque type and does not represent the time of day or some time and date in the calendar. It is used with Fl::seconds_between() and Fl::seconds_since() to measure elapsed time.

```
Fl_Timestamp start = Fl::now();
// do something
double s = Fl::seconds_since(start);
printf("That operation took %g seconds\n", s);
```

Depending on the system the resolution may be milliseconds or microseconds. Under certain conditions (particularly on Windows) the value in member `sec` may wrap around and does not represent a real time (maybe runtime of the system). Function seconds_since() below uses this to subtract two timestamps which is always a correct delta time with milliseconds or microseconds resolution.

Parameters

offset	optional signed offset in seconds added to the current time

Returns

this moment in time offset by `offset` as an opaque time stamp

Todo Fl::system_driver()->gettime() was implemented for the Forms library and has a limited resolution (on Windows: milliseconds). On POSIX platforms it uses gettimeofday() with microsecond resolution. A new function could use a better resolution on Windows with its multimedia timers which requires a new dependency: winmm.lib (dll). This could be a future improvement, maybe set as a build option or generally (requires Win95 or 98?).

See also

Fl::seconds_since(Fl_Timestamp& then)

Fl::seconds_between(Fl_Timestamp& back, Fl_Timestamp& further_back)

Fl::ticks_since(Fl_Timestamp& then)

Fl::ticks_between(Fl_Timestamp& back, Fl_Timestamp& further_back)

12.1.3.43 option() [1/2]

```
bool Fl::option (
            Fl_Option opt )  [extern]
```

FLTK library options management.

Options provide a way for the user to modify the behavior of an FLTK application. For example, clearing the `OPTION_SHOW_TOOLTIPS` will disable tooltips for all FLTK applications.
Options are set by the user or the administrator on user or machine level. In 1.3, FLUID has an Options dialog for that. In 1.4, there is an app named `fltk-options` that can be used from the command line or as a GUI tool. The machine level setting is read first, and the user setting can override the machine setting.
This function is used throughout FLTK to quickly query the user's wishes. There are options for using a native file chooser instead of the FLTK one wherever possible, disabling tooltips, disabling visible focus, disabling FLTK file chooser preview, etc. .
See `Fl::Fl_Option` for a list of available options.
Example:

```
if ( Fl::option(Fl::OPTION_ARROW_FOCUS) )
    { ..on..  }
else
    { ..off..  }
```

Note

Since FLTK 1.4.0 options can be managed with the `fltk-options` program. In FLTK 1.3.x options can be set in FLUID.

Parameters

opt	which option

Returns

true or false

See also

enum Fl::Fl_Option

Fl::option(Fl_Option, bool)

fltk-options application in command line and GUI mode

Since

FLTK 1.3.0

12.1.3.44 option() [2/2]

```
void Fl::option (
            Fl_Option opt,
            bool val )  [extern]
```

Override an option while the application is running.
Apps can override the machine settings and the user settings by calling `Fl::option(option, bool)`. The override takes effect immediately for this option for all widgets in the app for the life time of the app.
The override is not saved anywhere, and relaunching the app will restore the old settings.
Example:

```
Fl::option(Fl::OPTION_ARROW_FOCUS, true);     // on
Fl::option(Fl::OPTION_ARROW_FOCUS, false);    // off
```

Parameters

opt	which option
val	set to true or false

See also

enum Fl::Fl_Option

bool Fl::option(Fl_Option)

12.1.3.45 own_colormap()

`void Fl::own_colormap ( ) [extern]`

Makes FLTK use its `own colormap`.

This may make FLTK display better and will reduce conflicts with other programs that want lots of colors. However the colors may flash as you move the cursor between windows.

This does nothing if the current visual is not colormapped.

12.1.3.46 program_should_quit() [1/2]

`int Fl::program_should_quit ( ) [extern]`

Returns non-zero when a request for program termination was received and accepted.

On the MacOS platform, the "Quit xxx" item of the application menu is such a request, that is considered accepted when all windows are closed. On other platforms, this function returns 0 until `Fl::program_should_quit(1)` is called.

Version

1.4.0

12.1.3.47 program_should_quit() [2/2]

```
void Fl::program_should_quit (
            int should_i ) [extern]
```

Indicate to the FLTK library whether a program termination request was received and accepted.

A program may set this to 1, for example, while performing a platform-independent command asking the program to cleanly terminate, similarly to the "Quit xxx" item of the application menu under MacOS.

Version

1.4.0

12.1.3.48 readqueue()

`Fl_Widget * Fl::readqueue ( ) [extern]`

Reads the default callback queue and returns the first widget.

All Fl_Widgets that don't have a callback defined use the default callback `static` Fl_Widget::default_callback() that puts a pointer to the widget in a queue. This method reads the oldest widget out of this queue.

The queue (FIFO) is limited (currently 20 items). If the queue overflows, the oldest entry (Fl_Widget *) is discarded.

Relying on the default callback and reading the callback queue with Fl::readqueue() is not recommended. If you need a callback, you should set one with Fl_Widget::callback(Fl_Callback *cb, void *data) or one of its variants.

See also

Fl_Widget::callback()

Fl_Widget::callback(Fl_Callback *cb, void *data)

Fl_Widget::default_callback()

12.1.3.49 ready()

`int Fl::ready ( ) [extern]`

This is similar to Fl::check() except this does *not* call Fl::flush() or any callbacks, which is useful if your program is in a state where such callbacks are illegal.

This returns true if Fl::check() would do anything (it will continue to return true until you call Fl::check() or Fl::wait()).

```
while (!calculation_done()) {
  calculate();
  if (Fl::ready()) {
    do_expensive_cleanup();
    Fl::check();
    if (user_hit_abort_button()) break;
  }
}
```

12.1.3.50 reload_scheme()

```
int Fl::reload_scheme ( ) [extern]
```

Called internally when setting a new scheme according to scheme name.
Loads or reloads the current scheme selection.

Returns

Always 1 (this may change in the future)

See void Fl::scheme(const char *name)

12.1.3.51 remove_check()

```
void Fl::remove_check (
            Fl_Timeout_Handler cb,
            void * argp = 0 ) [extern]
```

Removes a check callback.
It is harmless to remove a check callback that no longer exists.

12.1.3.52 remove_idle()

```
void Fl::remove_idle (
            Fl_Idle_Handler cb,
            void * data = 0 ) [extern]
```

Removes the specified idle callback, if it is installed.
The given idle callback is only removed if `data` matches the value used when the idle callback was installed. If the idle callback wants to remove itself, the value provided by the `data` argument can (and should) be used.
Example for a "one-shot" idle callback, i.e. one that removes itself when it is called for the first time.

```
#include <FL/Fl.H>
#include <FL/Fl_Double_Window.H>
#include <FL/Fl_Button.H>
void idle1(void *data) {
  printf("idle1 called with data %4d\n", fl_int(data));
  fflush(stdout);
  // ... do something ...
  Fl::remove_idle(idle1, data);
}
void quit_cb(Fl_Widget *w, void *v) {
  w->window()->hide();
}
int main(int argc, char **argv) {
  auto window = new Fl_Double_Window(200, 100);
  auto button = new Fl_Button(20, 20, 160, 60, "Quit");
  button->callback(quit_cb);
  window->end();
  window->show(argc, argv);
  Fl::add_idle(idle1, (void *)1234);
  return Fl::run();
}
```

Parameters

in	*cb*	idle callback in question
in	*data*	optional data. Default: zero / nullptr.

12.1.3.53 remove_next_timeout()

```
int Fl::remove_next_timeout (
```

```
            Fl_Timeout_Handler cb,
            void * data = 0,
            void ** data_return = 0 )  [extern]
```

Remove the next matching timeout callback and return its `data` pointer.
This method removes only the next matching timeout and returns in `data_return` (if non-NULL) the `data` member given when the timeout was scheduled.
This method is useful if you remove a timeout before it is scheduled and you need to get and use its data value, for instance to free() or delete the data associated with the timeout.
This method returns non-zero if a matching timeout was found and zero if no timeout matched the request.
If the return value is `N` > 1 then there are N - 1 more matching timeouts pending.
If you need to remove all timeouts with a particular callback `cb` you must repeat this call until it returns 1 (all timeouts removed) or zero (no matching timeout), whichever occurs first.

Parameters

`in`	*cb*	Timer callback to be removed (must match)
`in`	*data*	Wildcard if NULL, must match otherwise
`in,out`	*data_return*	Pointer to (void ∗) to receive the data value

Returns

non-zero if a timer was found and removed

Return values

0	no matching timer was found
1	the last matching timeout was found and removed
N>1	a matching timeout was removed and there are (N - 1) matching timeouts pending

See also

Fl::remove_timeout(Fl_Timeout_Handler cb, void ∗data)

Since

1.4.0

12.1.3.54 remove_timeout()

```
void Fl::remove_timeout (
            Fl_Timeout_Handler cb,
            void * data = 0 )  [extern]
```

Remove one or more matching timeout callbacks from the timer queue.
This method removes **all** matching timeouts, not just the first one.
If the `data` argument is `NULL` (the default!) only the callback `cb` must match, i.e. all timer entries with this callback are removed.
It is harmless to remove a timeout callback that no longer exists.
If you want to remove only the next matching timeout you can use Fl::remove_next_timeout(Fl_Timeout_Handler cb, void ∗data, void ∗
(available since FLTK 1.4.0).

Parameters

`in`	*cb*	Timer callback to be removed (must match)
`in`	*data*	Wildcard if NULL (default), must match otherwise

See also

Fl::remove_next_timeout(Fl_Timeout_Handler cb, void *data, void **data_return)

12.1.3.55 repeat_timeout()

```
void Fl::repeat_timeout (
            double time,
            Fl_Timeout_Handler cb,
            void * data = 0 )  [extern]
```

Repeats a timeout callback from the expiration of the previous timeout, allowing for more accurate timing.
You should call this method only inside a timeout callback of the same or a logically related timer from whose expiration time the new timeout shall be scheduled. Otherwise the timing accuracy can't be improved and the exact behavior is undefined.
If you call this outside a timeout callback the behavior is the same as Fl::add_timeout().
Example: The following code will print "TICK" each second on stdout with a fair degree of accuracy:

```
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <stdio.h>

void callback(void *) {
  printf("TICK\n");
  Fl::repeat_timeout(1.0, callback); // retrigger timeout
}

int main() {
  Fl_Window win(100, 100);
  win.show();
  Fl::add_timeout(1.0, callback); // set up first timeout
  return Fl::run();
}
```

Parameters

in	*time*	delta time in seconds until the timer expires
in	*cb*	callback function
in	*data*	optional user data (default: NULL)

12.1.3.56 run()

```
int Fl::run ( )  [extern]
```

Calls Fl::wait()repeatedly as long as any windows are displayed.
When all the windows are closed it returns zero (supposedly it would return non-zero on any errors, but FLTK calls exit directly for these). A normal program will end main() with return Fl::run();.

Note

Fl::run() and Fl::wait() (but not Fl::wait(double)) both return when all FLTK windows are closed. Therefore, a MacOS FLTK application possessing Fl_Sys_Menu_Bar items able to create new windows and expected to keep running without any open window cannot use these two functions. One solution is to run the event loop as follows:

```
while (!Fl::program_should_quit()) Fl::wait(1e20);
```

12.1.3.57 scheme()

```
int Fl::scheme (
            const char * s )  [extern]
```

Sets the current widget scheme.
NULL will use the scheme defined in the FLTK_SCHEME environment variable or the scheme resource under X11. Otherwise, any of the following schemes can be used:

- "none" - This is the default look-n-feel which resembles old Windows (95/98/Me/NT/2000) and old GTK/KDE
- "base" - This is an alias for "none"

- "plastic" - This scheme is inspired by the Aqua user interface on macOS
- "gtk+" - This scheme is inspired by the Red Hat Bluecurve theme
- "gleam" - This scheme is inspired by the Clearlooks Glossy scheme. (Colin Jones and Edmanuel Torres).
- "oxy" - This is a subset of Dmitrij K's oxy scheme (STR 2675, 3477)

If the given scheme name is unknown, the default scheme will be used.
Setting the scheme (name) is case insensitive, but the stored scheme name will always be lowercase and Fl::scheme() will return this lowercase name or `NULL` if no scheme or the default scheme ("none" or "base") was set.

Parameters

in	*s*	Scheme name of NULL

Return values

0	if the scheme has not been set or is the default scheme
1	if a scheme other than "none"/"base" was set

See also

Fl::scheme() to get the name of the current scheme

Fl::is_scheme(const char*) to test if the specified scheme is set

12.1.3.58 scrollbar_size() [1/2]

```
int Fl::scrollbar_size ( ) [extern]
```

Gets the default scrollbar size used by Fl_Browser_, Fl_Help_View, Fl_Scroll, and Fl_Text_Display widgets.

Returns

The default size for widget scrollbars, in pixels.

12.1.3.59 scrollbar_size() [2/2]

```
void Fl::scrollbar_size (
            int W ) [extern]
```

Sets the default scrollbar size that is used by the Fl_Browser_, Fl_Help_View, Fl_Scroll, and Fl_Text_Display widgets.

Parameters

in	*W*	The new default size for widget scrollbars, in pixels.

12.1.3.60 seconds_between()

```
double Fl::seconds_between (
            Fl_Timestamp & back,
            Fl_Timestamp & further_back ) [extern]
```

Return the time in seconds between two time stamps.

Parameters

in	*back*	a previously taken time stamp

Parameters

in	*further_back*	an even earlier time stamp

Returns

elapsed seconds and fractions of a second

See also

Fl::seconds_since(Fl_Timestamp& then)

Fl::now()

12.1.3.61 seconds_since()

```
double Fl::seconds_since (
            Fl_Timestamp & then )  [extern]
```

Return the time in seconds between now and a previously taken time stamp.

Parameters

in	*then*	a previously taken time stamp

Returns

elapsed seconds and fractions of a second

See also

Fl::seconds_between(Fl_Timestamp& back, Fl_Timestamp& further_back)

Fl::now()

Fl::distant_past()

12.1.3.62 set_box_color()

```
void Fl::set_box_color (
            Fl_Color c )  [extern]
```

Sets the drawing color for the box that is currently drawn.

This method sets the current drawing color fl_color() depending on the widget's state to either `c` or fl_inactive(c).

It should be used whenever a box background is drawn in the box (type) drawing code instead of calling fl_color(Fl_Color bg) with the background color `bg`, usually Fl_Widget::color().

This method is only useful inside box drawing code. Whenever a box is drawn with one of the standard box drawing methods, a static variable is set depending on the widget's current state - if the widget is !active_r() then the internal variable is false (0), otherwise it is true (1). This is faster than calling Fl_Widget::active_r() because the state is cached.

See also

Fl::draw_box_active()

Fl::box_color(Fl_Color)

12.1.3.63 set_boxtype()

```
void Fl::set_boxtype (
            Fl_Boxtype t,
            Fl_Box_Draw_F * f,
```

```
            uchar dx,
            uchar dy,
            uchar dw,
            uchar dh,
            Fl_Box_Draw_Focus_F * ff = NULL )  [extern]
```

Sets the function to call to draw a specific box type.

Parameters

in	*t*	index of the box type between 0 (FL_NO_BOX) and up to and including FL_MAX_BOXTYPE
in	*f*	callback function that draws the box
in	*dx,dy*	top left frame width, distance in pixels to box contents
in	*dw,dh*	left plus right frame width, top plus bottom frame width
in	*ff*	optional callback that draws the box focus, defaults to a rectangle, inset by dx, dy, dw, dh

12.1.3.64 ticks_between()

```
long Fl::ticks_between (
            Fl_Timestamp & back,
            Fl_Timestamp & further_back )  [extern]
```

Return the time in ticks (60Hz) between two time stamps.

Parameters

in	*back*	a previously taken time stamp
in	*further_back*	an even earlier time stamp

Returns

elapsed ticks in 60th of a second

See also

Fl::ticks_since(Fl_Timestamp& then)

Fl::now()

12.1.3.65 ticks_since()

```
long Fl::ticks_since (
            Fl_Timestamp & then )  [extern]
```

Return the time in ticks (60Hz) between now and a previously taken time stamp.
Ticks are a convenient way to time animations 'per frame'. Even though modern computers use all kinds of screen refresh rates, 60Hz is a very good base for animation that is typically shown in user interface graphics.

Parameters

in	*then*	a previously taken time stamp

Returns

elapsed ticks in 60th of a second

See also

Fl::ticks_between(Fl_Timestamp& back, Fl_Timestamp& further_back)

Fl::now()

12.1.3.66 timeout_list()

```
std::vector< Fl::TimeoutData > Fl::timeout_list ( ) [extern]
```

Return a list of all currently running timeouts.

Returns

a vector with all relevant timeout data

12.1.3.67 use_high_res_GL() [1/2]

```
int Fl::use_high_res_GL ( ) [extern]
```

Returns whether GL windows should be drawn at high resolution on Apple computers with retina displays.

Version

Default value set to 1 starting with FLTK 1.5

12.1.3.68 use_high_res_GL() [2/2]

```
void Fl::use_high_res_GL (
            int val ) [extern]
```

Sets whether GL windows should be drawn at high resolution on Apple computers with retina displays.
Because the default value is `1`, this function is useful only in the very unlikely situation where a macOS app would choose to draw low resolution GL graphics when mapped on a retina display. This function has effect only under macOS.

Version

1.3.4 (default value changed in 1.5)

12.1.3.69 version()

```
double Fl::version ( ) [extern]
```

Returns the compiled-in value of the FL_VERSION constant.
This is useful for checking the version of a shared library.

Deprecated Use int Fl::api_version() instead.

12.1.3.70 visible_focus() [1/2]

```
int Fl::visible_focus ( ) [inline]
```

Gets or sets the visible keyboard focus on buttons and other non-text widgets.
The default mode is to enable keyboard focus for all widgets.

12.1.3.71 visible_focus() [2/2]

```
void Fl::visible_focus (
            int v ) [inline]
```

Gets or sets the visible keyboard focus on buttons and other non-text widgets.
The default mode is to enable keyboard focus for all widgets.

12.1.3.72 visual()

```
int Fl::visual (
            int flags )  [extern]
```

Selects a visual so that your graphics are drawn correctly.
This is only allowed before you call show() on any windows. This does nothing if the default visual satisfies the capabilities, or if no visual satisfies the capabilities, or on systems that don't have such brain-dead notions.
Only the following combinations do anything useful:

- Fl::visual(FL_RGB)
  Full/true color (if there are several depths FLTK chooses the largest). Do this if you use fl_draw_image for much better (non-dithered) output.

- Fl::visual(FL_RGB8)
  Full color with at least 24 bits of color. FL_RGB will always pick this if available, but if not it will happily return a less-than-24 bit deep visual. This call fails if 24 bits are not available.

This returns true if the system has the capabilities by default or FLTK succeeded in turning them on. Your program will still work even if this returns false (it just won't look as good).

12.1.3.73 wait() [1/2]

```
int Fl::wait ( )  [extern]
```

Waits until "something happens" and then returns.
Call this repeatedly to "run" your program. You can also check what happened each time after this returns, which is quite useful for managing program state.
What this really does is call all idle callbacks, all elapsed timeouts, call Fl::flush() to get the screen to update, and then wait some time (zero if there are idle callbacks, the shortest of all pending timeouts, or infinity), for any events from the user or any Fl::add_fd() callbacks. It then handles the events and calls the callbacks and then returns.

Returns

non-zero if there are any visible windows - this may change in future versions of FLTK.

12.1.3.74 wait() [2/2]

```
double Fl::wait (
            double time_to_wait )  [extern]
```

Waits a maximum of `time_to_wait` seconds or until "something happens".
See Fl::wait() for the description of operations performed when "something happens".

Returns

Always 1 on Windows. Otherwise, it is positive if an event or fd happens before the time elapsed. It is zero if nothing happens. It is negative if an error occurs (this will happen on X11 if a signal happens).

12.1.4 Variable Documentation

12.1.4.1 help

```
const char *const Fl::help = helpmsg+13  [extern]
```

Usage string displayed if Fl::args() detects an invalid argument.
This may be changed to point to customized text at run-time.

12.2 Fl::Pen Namespace Reference

FLTK Pen/Stylus/Tablet input driver API.

Enumerations

- enum Event {
  DETECTED = 0x1000 , CHANGED , IN_RANGE , OUT_OF_RANGE ,
  ENTER , LEAVE , TOUCH , LIFT ,
  HOVER , DRAW , BUTTON_PUSH , BUTTON_RELEASE }

  List of pen events.
- enum class State : uint32_t {
  State::NONE = 0x0000 , State::TIP_HOVERS = 0x0001 , State::TIP_DOWN = 0x0002 , State::ERASER_HOVERS = 0x0004 ,
  State::ERASER_DOWN = 0x0008 , State::BUTTON0 = 0x0100 , State::BUTTON1 = 0x0200 , State::BUTTON2 = 0x0400 ,
  State::BUTTON3 = 0x0800 , State::ANY_DOWN = BUTTON0 | BUTTON1 | BUTTON2 | BUTTON3 | TIP_↩DOWN | ERASER_DOWN }

  Bitfield of pen state flags.
- enum class Trait : uint32_t {
  Trait::NONE = 0x0000 , Trait::DRIVER_AVAILABLE = 0x0001 , Trait::DETECTED = 0x0002 , Trait::DISPLAY = 0x0004 ,
  Trait::PEN_ID = 0x0008 , Trait::ERASER = 0x0010 , Trait::PRESSURE = 0x0020 , Trait::BARREL_PRESSURE = 0x0040 ,
  Trait::TILT_X = 0x0080 , Trait::TILT_Y = 0x0100 , Trait::TWIST = 0x0200 , Trait::PROXIMITY = 0x0400 }

  Bitfield of traits.

Functions

- Trait driver_traits ()

  Query the traits supported by the pen/tablet driver.
- bool driver_traits (Trait bits)

  Return true if the corresponding bit is set in the driver traits.
- constexpr State operator& (State lhs, State rhs)

  Bitwise AND operator for State enum.
- constexpr Trait operator& (Trait lhs, Trait rhs)

  Bitwise AND operator for Trait enum.
- constexpr State operator| (State lhs, State rhs)

  Bitwise OR operator for State enum.
- constexpr Trait operator| (Trait lhs, Trait rhs)

  Bitwise OR operator for Trait enum.
- State & operator|= (State &lhs, State rhs)

  Bitwise OR assignment operator for State enum.
- Trait & operator|= (Trait &lhs, Trait rhs)

  Bitwise OR assignment operator for Trait enum.
- Trait pen_traits (int pen_id=0)

  Query traits of the current pen or stylus.
- bool pen_traits (Trait bits, int pen_id=0)

  Return true if the corresponding bit is set in the pen traits.
- void release ()

  Clear the "pushed" state and forward pen events as mouse events.
- void subscribe (Fl_Widget *widget)

  Receives a Pen::ENTER event when the pen enters this widget.
- void unsubscribe (Fl_Widget *widget)

  Stop receiving Pen::ENTER for this widget.

Query values during event handling

- double event_x ()

Returns the pen x and y position inside the handling widget as doubles.

- double **event_y** ()

  Returns pen Y coordinate in widget space, see event_x().
- double event_x_root ()

  Returns the pen x and y position in global coordinates as doubles.
- double **event_y_root** ()

  Returns pen Y coordinate in screen space, see event_x_root().
- int event_pen_id ()

  Returns the ID of the pen used in the last event.
- double event_pressure ()

  Returns the pressure between the tip or eraser and the surface.
- double event_barrel_pressure ()

  Returns barrel pressure or tangential pressure.
- double event_tilt_x ()

  Returns the tilt of the pen in the x and y directions between -1 and 1.
- double **event_tilt_y** ()

  Returns pen Y-axis tilt, see event_tilt_x()
- double event_twist ()

  Returns the pen's axial rotation in degrees.
- double event_proximity ()

  Returns the proximity of the pen to the surface between 0 and 1.
- State event_state ()

  Returns the state of the various buttons and tips.
- bool event_state (State bits)

  Return true if any of the bits are set in the event state.
- State event_trigger ()

  Returns the state change that triggered the event.

12.2.1 Detailed Description

FLTK Pen/Stylus/Tablet input driver API.

Chapter 13

Class Documentation

13.1 Fl_Grid::Cell Class Reference

Public Member Functions

- Fl_Grid_Align **align** () const
- void **align** (Fl_Grid_Align align)
- **Cell** (Fl_Widget *w, int row, int col)
- **Cell** (int row, int col)
- void **Cell_** ()
- short **col** () const
- short **colspan** () const
- void **colspan** (short v)
- void **minimum_size** (int *w, int *h) const
- void **minimum_size** (int w, int h)
- Cell * **next** () const

  Returns the next widget cell of the same row of this cell.
- void next (Cell *c)

  Sets the `next` pointer of a grid's cell.
- short **row** () const
- short **rowspan** () const
- void **rowspan** (short v)
- Fl_Widget * **widget** () const
- ~Cell ()=default

  The destructor deletes the cell.

Friends

- class **Fl_Grid**

13.1.1 Constructor & Destructor Documentation

13.1.1.1 ~Cell()

`Fl_Grid::Cell::~Cell ( ) [default]`

The destructor deletes the cell.

Todo Fl_Grid's cell destructor should remove the cell from the grid. Currently it does nothing!

13.1.2 Member Function Documentation

13.1.2.1 next()

```
void Fl_Grid::Cell::next (
            Cell * c )  [inline]
```

Sets the `next` pointer of a grid's cell.
Internal use only!
Do not use this method, it may corrupt the allocated memory.
The documentation for this class was generated from the following file:

- Fl_Grid.H

13.2 Fl_Terminal::CharStyle Class Reference

Public Member Functions

- void **attrib** (uchar val)
- uchar **attrib** (void) const
- void **bgcolor** (Fl_Color val)
- void **bgcolor** (int r, int g, int b)
- Fl_Color **bgcolor** (void) const
- void **bgcolor_xterm** (Fl_Color val)
- void **bgcolor_xterm** (uchar val)
- void **charflags** (uchar val)
- uchar **charflags** (void) const
- **CharStyle** (bool fontsize_defer)
- int **charwidth** (void) const
- void **clr_charflag** (uchar val)
- uchar **colorbits_only** (uchar inflags) const
- void **defaultbgcolor** (Fl_Color val)
- Fl_Color **defaultbgcolor** (void) const
- void **defaultfgcolor** (Fl_Color val)
- Fl_Color **defaultfgcolor** (void) const
- void **fgcolor** (Fl_Color val)
- void **fgcolor** (int r, int g, int b)
- Fl_Color **fgcolor** (void) const
- void **fgcolor_xterm** (Fl_Color val)
- void **fgcolor_xterm** (uchar val)
- Fl_Color **fltk_bg_color** (uchar ci)
- Fl_Color **fltk_fg_color** (uchar ci)
- int **fontdescent** (void) const
- void **fontface** (Fl_Font val)
- Fl_Font **fontface** (void) const
- int **fontheight** (void) const
- void **fontsize** (Fl_Fontsize val)
- Fl_Fontsize **fontsize** (void) const
- int **onoff** (bool flag, Attrib a) const
- void **set_charflag** (uchar val)
- void **sgr_blink** (bool val)
- void **sgr_bold** (bool val)
- void **sgr_dbl_under** (bool val)
- void **sgr_dim** (bool val)
- void **sgr_inverse** (bool val)
- void **sgr_italic** (bool val)
- void **sgr_reset** (void)

- void **sgr_strike** (bool val)
- void **sgr_underline** (bool val)
- void **update** (void)
- void **update_fake** (void)

The documentation for this class was generated from the following files:

- Fl_Terminal.H
- Fl_Terminal.cxx

13.3 Fl_GIF_Image::GIF_FRAME::CPAL Struct Reference

Public Attributes

- uchar **b**
- uchar **g**
- uchar **r**

The documentation for this struct was generated from the following file:

- Fl_GIF_Image.H

13.4 Fl_Terminal::Cursor Class Reference

Public Member Functions

- void **bgcolor** (Fl_Color val)
- Fl_Color **bgcolor** (void) const
- void **col** (int val)
- int **col** (void) const
- int **down** (void)
- void **fgcolor** (Fl_Color val)
- Fl_Color **fgcolor** (void) const
- void **h** (int val)
- int **h** (void) const
- void **home** (void)
- bool **is_rowcol** (int drow, int dcol) const
- int **left** (void)
- int **right** (void)
- void **row** (int val)
- int **row** (void) const
- void **scroll** (int nrows)
- int **up** (void)

The documentation for this class was generated from the following files:

- Fl_Terminal.H
- Fl_Terminal.cxx

13.5 Fl_Shared_Image::Deleter Class Reference

Custom deleter for std::unique_ptr<>.

```
#include <Fl_Shared_Image.H>
```

Public Member Functions

- void **operator()** (Fl_Shared_Image *img) const

13.5.1 Detailed Description

Custom deleter for std::unique_ptr<>.
Fl_Shared_Image does not follow the standard new/delete memory lifecycle. To implement RAII, use std::unique↩_ptr with this custom deleter to ensure proper release of the image resource.

```
std::unique_ptr<Fl_Shared_Image, Fl_Shared_Image::Deleter> img;
img = Fl_Shared_Image::get("image.png");
// image will be automatically released when 'img' goes out of scope
```

The documentation for this class was generated from the following file:

- Fl_Shared_Image.H

13.6 Fl_Preferences::Entry Struct Reference

Public Attributes

- char ∗ **name**
- char ∗ **value**

The documentation for this struct was generated from the following file:

- Fl_Preferences.H

13.7 Fl_Terminal::EscapeSeq Class Reference

Public Member Functions

- int **defvalmax** (int dval, int max) const
- void **esc_mode** (char val)
- char **esc_mode** (void) const
- bool **is_csi** (void) const
- int **parse** (char c)
- bool **parse_in_progress** (void) const
- void **reset** (void)
- void **restore_cursor** (int &row, int &col)
- void **save_cursor** (int row, int col)
- int **total_vals** (void) const
- int **val** (int i) const

Static Public Attributes

- static const int **completed** = 1
- static const int **fail** = -1
- static const int **maxbuff** = 80
- static const int **maxvals** = 20
- static const int **success** = 0

The documentation for this class was generated from the following files:

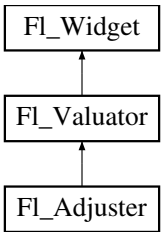
- Fl_Terminal.H
- Fl_Terminal.cxx

13.8 Fl_Adjuster Class Reference

The Fl_Adjuster widget was stolen from Prisms, and has proven to be very useful for values that need a large dynamic range.

```
#include <Fl_Adjuster.H>
```

Inheritance diagram for Fl_Adjuster:



Public Member Functions

- Fl_Adjuster (int X, int Y, int W, int H, const char *l=0)

  Creates a new Fl_Adjuster widget using the given position, size, and label string.
- int soft () const

  If "soft" is turned on, the user is allowed to drag the value outside the range.
- void soft (int s)

  If "soft" is turned on, the user is allowed to drag the value outside the range.

Public Member Functions inherited from Fl_Valuator

- void **bounds** (double a, double b)

  Sets the minimum (a) and maximum (b) values for the valuator widget.
- double **clamp** (double)

  Clamps the passed value to the valuator range.
- virtual int format (char *)

  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.
- virtual std::string format_str ()

  C++11 API for Fl_Valuator::format(char buffer).*
- double increment (double, int)

  Adds n times the step value to the passed value.
- double maximum () const

  Gets the maximum value for the valuator.
- void maximum (double a)

  Sets the maximum value for the valuator.
- double minimum () const

  Gets the minimum value for the valuator.
- void minimum (double a)

  Sets the minimum value for the valuator.
- void precision (int digits)

  Sets the step value to 1.0 / 10^{digits}.
- void range (double a, double b)

  Sets the minimum and maximum values for the valuator.
- double round (double)

  Round the passed value to the nearest step increment.
- double step () const

  Gets or sets the step value.
- void **step** (double a, int b)

  See double Fl_Valuator::step() const

- void **step** (double s)

  See double Fl_Valuator::step() const.
- void **step** (int a)

See double Fl_Valuator::step() const

- double value () const

  Gets the floating point(double) value.
- int value (double)

  Sets the current value.
- ~**Fl_Valuator** () override

  Destructor is accessible despite protected constructor.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.

- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

Sets the image to use as part of the widget label when in the inactive state.

- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.

- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

Changes the size of the widget.

- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void draw () override

  Draws the widget.
- int handle (int) override

  Handles the specified event.
- void value_damage () override

  Asks for partial redraw.

Protected Member Functions inherited from Fl_Valuator

- Fl_Valuator (int X, int Y, int W, int H, const char *L)

  Creates a new Fl_Valuator widget using the given position, size, and label string.
- void **handle_drag** (double newvalue)

  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.
- void **handle_push** ()

  Stores the current value in the previous value.
- void **handle_release** ()

  Called after an FL_WHEN_RELEASE event is received and before the callback.
- int **horizontal** () const

  Tells if the valuator is an FL_HORIZONTAL one.
- double **previous_value** () const

  Gets the previous floating point value before an event changed it.
- void **set_value** (double v)

  Sets the current floating point value.
- double **softclamp** (double)

  Clamps the value, but accepts v if the previous value is not already out of range.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.

- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.8.1 Detailed Description

The Fl_Adjuster widget was stolen from Prisms, and has proven to be very useful for values that need a large dynamic range.

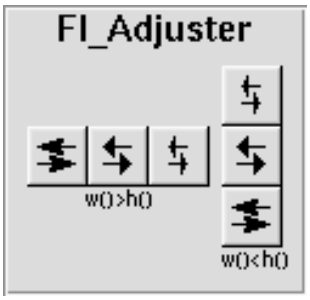


Figure 13.1 Fl_Adjuster

When you press a button and drag to the right the value increases. When you drag to the left it decreases. The largest button adjusts by 100 ∗ step(), the next by 10 ∗ step() and that smallest button by step(). Clicking on the buttons increments by 10 times the amount dragging by a pixel does. Shift + click decrements by 10 times the amount.

13.8.2 Constructor & Destructor Documentation

13.8.2.1 Fl_Adjuster()

```
Fl_Adjuster::Fl_Adjuster (
            int X,
            int Y,
            int W,
            int H,
            const char * l = 0 )
```

Creates a new Fl_Adjuster widget using the given position, size, and label string.
It looks best if one of the dimensions is 3 times the other.
Inherited destructor destroys the Valuator.

13.8.3 Member Function Documentation

13.8.3.1 draw()

```
void Fl_Adjuster::draw ( )  [override], [protected], [virtual]
```

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();                 // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

13.8.3.2 handle()

```
int Fl_Adjuster::handle (
            int event )  [override], [protected], [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to *override* the behavior of the base class. This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to *filter* event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In

such a case you must not call the handle() method of the base class and tell FLTK that you *consumed* the event by returning 1 even if you didn't *do* anything with it.

Parameters

in	*event*	the kind of event received

Return values

0	if the event was not used or understood
1	if the event was used and can be deleted

See also

Fl_Event

Reimplemented from Fl_Widget.

13.8.3.3 soft() [1/2]

```
int Fl_Adjuster::soft ( ) const  [inline]
```

If "soft" is turned on, the user is allowed to drag the value outside the range.
If they drag the value to one of the ends, let go, then grab again and continue to drag, they can get to any value.
Default is one.

13.8.3.4 soft() [2/2]

```
void Fl_Adjuster::soft (
            int s )  [inline]
```

If "soft" is turned on, the user is allowed to drag the value outside the range.
If they drag the value to one of the ends, let go, then grab again and continue to drag, they can get to any value.
Default is one.

13.8.3.5 value_damage()

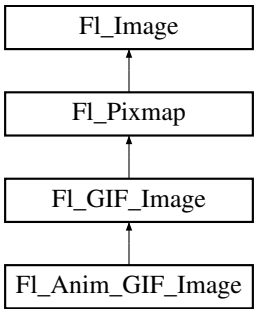
```
void Fl_Adjuster::value_damage ( )  [override], [protected], [virtual]
```

Asks for partial redraw.
Reimplemented from Fl_Valuator.
The documentation for this class was generated from the following files:

- Fl_Adjuster.H
- Fl_Adjuster.cxx

13.9 Fl_Anim_GIF_Image Class Reference

The Fl_Anim_GIF_Image class supports loading, caching, and drawing of animated Compuserve GIF[SM] images.
Inheritance diagram for Fl_Anim_GIF_Image:



Public Types

- enum Flags {
  DONT_START = 1 , DONT_RESIZE_CANVAS = 2 , DONT_SET_AS_IMAGE = 4 , OPTIMIZE_MEMORY = 8 ,
  LOG_FLAG = 64 , DEBUG_FLAG = 128 }

  When opening an Fl_Anim_GIF_Image there are some options that can be passed in a `flags` value.

Public Member Functions

- Fl_Widget * canvas () const

  Gets the current widget, that is used to display the frame images.
- void canvas (Fl_Widget *canvas, unsigned short flags=0)

  Link the image back to a widget for automated animation.
- int canvas_h () const

  Return the height of the animation canvas.
- int canvas_w () const

  Return the width of the animation canvas.
- void color_average (Fl_Color c, float i) override

  Applies a color average to all frames.
- Fl_Image * **copy** () const
- Fl_Image * copy (int W, int H) const override

  Copy and resize the animation frames.
- int **debug** () const
- void delay (int frame, double delay)

  Set the delay of frame `[0-frames() -1]` in seconds.
- double delay (int frame_) const

  Return the delay of frame `[0-frames() -1]` in seconds.
- void desaturate () override

  Desaturate to all frames of the animation.
- void draw (int x, int y, int w, int h, int cx=0, int cy=0) override

  Draw the current frame of the animation.
- **Fl_Anim_GIF_Image** ()

  Create an empty animated GIF image shell.
- Fl_Anim_GIF_Image (const char *filename, Fl_Widget *canvas=0, unsigned short flags=0)

  Load an animated GIF image from a file.
- Fl_Anim_GIF_Image (const char *imagename, const unsigned char *data, const size_t length, Fl_Widget *canvas=0, unsigned short flags=0)

  Load an animated GIF image from memory.
- int frame () const

  Return the current frame.
- void frame (int frame)

  Set the current frame.
- int frame_h (int frame) const

  Return the frame dimensions of a frame.
- bool frame_uncache () const

  Return the active frame_uncache() setting.
- void frame_uncache (bool uncache)

  Use frame_uncache() to set or forbid frame image uncaching.
- int frame_w (int frame) const

  Return the frame dimensions of a frame.
- int frame_x (int frame) const

Return the frame position of a frame.

- int frame_y (int frame) const

  Return the frame position of a frame.
- int frames () const

  Get the number of frames in the animation.
- Fl_Image * image () const

  Return the current frame image.
- Fl_Image * image (int frame) const

  Return the image of the given frame index.
- bool is_animated () const

  Check if this is a valid animation with more than one frame.
- bool load (const char *name, const unsigned char *imgdata=NULL, size_t imglength=0)

  Load an animation from a file or from a memory block.
- const char * name () const

  Return the name of the played file as specified in the constructor.
- bool next ()

  Show the next frame if the animation is stopped.
- bool playing () const

  Return if the animation is currently running or stopped.
- Fl_Anim_GIF_Image & resize (double scale)

  Resizes the image to the specified size, replacing the current image.
- Fl_Anim_GIF_Image & resize (int w, int h)

  Resizes the image to the specified size, replacing the current image.
- double speed () const

  Get the animation speed factor.
- void speed (double speed)

  Set the animation speed factor.
- bool start ()

  The start() method (re-)starts the playing of the frames.
- bool stop ()

  The stop() method stops the playing of the frames.
- void uncache () override

  Uncache all cached image data now.
- bool valid () const

  Check if animation is valid.
- ~Fl_Anim_GIF_Image () override

  Release the image and all cached data.

Public Member Functions inherited from Fl_GIF_Image

- Fl_GIF_Image (const char *filename)

  This constructor loads a GIF image from the given file.
- Fl_GIF_Image (const char *imagename, const unsigned char *data)

  This constructor loads a GIF image from memory (deprecated).
- Fl_GIF_Image (const char *imagename, const unsigned char *data, const size_t length)

  This constructor loads a GIF image from memory.

Public Member Functions inherited from Fl_Pixmap

- int **cache_h** () const
- int **cache_w** () const
- void color_average (Fl_Color c, float i) override

  The color_average() method averages the colors in the image with the provided FLTK color value.
- Fl_Image ∗ **copy** () const
- Fl_Image ∗ copy (int W, int H) const override

  Creates a resized copy of the image.
- void desaturate () override

  The desaturate() method converts an image to grayscale.
- void **draw** (int X, int Y)
- void draw (int X, int Y, int W, int H, int cx=0, int cy=0) override

  Draws the image to the current drawing surface with a bounding box.
- Fl_Pixmap (char ∗const ∗D)

  The constructors create a new pixmap from the specified XPM data.
- Fl_Pixmap (const char ∗const ∗D)

  The constructors create a new pixmap from the specified XPM data.
- Fl_Pixmap (const uchar ∗const ∗D)

  The constructors create a new pixmap from the specified XPM data.
- Fl_Pixmap (uchar ∗const ∗D)

  The constructors create a new pixmap from the specified XPM data.
- void label (Fl_Menu_Item ∗m) override

  This method is an obsolete way to set the image attribute of a menu item.
- void label (Fl_Widget ∗w) override

  This method is an obsolete way to set the image attribute of a widget or menu item.
- void uncache () override

  If the image has been cached for display, delete the cache data.
- virtual ∼**Fl_Pixmap** ()

  The destructor frees all memory and server resources that are used by the pixmap.

Public Member Functions inherited from Fl_Image

- virtual class Fl_Shared_Image ∗ as_shared_image ()

  Returns whether an image is an Fl_Shared_Image or not.
- Fl_Image ∗ copy () const

  Creates a copy of the image in the same size.
- int count () const

  Returns the number of data values associated with the image.
- int d () const

  Returns the image depth.
- const char ∗const ∗ data () const

  Returns a pointer to the current image data array.
- int **data_h** () const

  Returns the height of the image data.
- int **data_w** () const

  Returns the width of the image data.
- void draw (int X, int Y)

  Draws the image to the current drawing surface.
- int fail () const

  Returns a value that is not 0 if there is currently no image available.
- Fl_Image (int W, int H, int D)

The constructor creates an empty image with the specified width, height, and depth.

- int h () const

  Returns the current image drawing height in FLTK units.
- void inactive ()

  The inactive() method calls color_average(FL_BACKGROUND_COLOR, 0.33f) to produce an image that appears grayed out.
- int ld () const

  Returns the current line data size in bytes.
- virtual void release ()

  Releases an Fl_Image - the same as '`delete this`'.
- virtual void scale (int width, int height, int proportional=1, int can_expand=0)

  Sets the drawing size of the image.
- int w () const

  Returns the current image drawing width in FLTK units.
- virtual ~**Fl_Image** ()

  The destructor is a virtual method that frees all memory used by the image.

Static Public Member Functions

- static int frame_count (const char *name, const unsigned char *imgdata=NULL, size_t imglength=0)

  Get the number of frames in a GIF file or in a GIF compressed data block.

Static Public Member Functions inherited from Fl_GIF_Image

- static bool **is_animated** (const char *name_)

Static Public Member Functions inherited from Fl_Image

- static Fl_Labeltype **define_FL_IMAGE_LABEL** ()
- static Fl_RGB_Scaling **RGB_scaling** ()

  Returns the currently used RGB image scaling method.
- static void RGB_scaling (Fl_RGB_Scaling)

  Sets the RGB image scaling method used for copy(int, int).
- static Fl_RGB_Scaling **scaling_algorithm** ()

  Gets what algorithm is used when resizing a source image to draw it.
- static void scaling_algorithm (Fl_RGB_Scaling algorithm)

  Sets what algorithm is used when resizing a source image to draw it.

Static Public Attributes

- static bool loop = true

  The loop flag can be used to (dis-)allow loop count.
- static double min_delay = 0.

  The min_delay value can be used to set a minimum value for the frame delay for playback.

Static Public Attributes inherited from Fl_GIF_Image

- static bool animate = false

  Sets how the shared image core routine should treat animated GIF files.

Static Public Attributes inherited from Fl_Image

- static const int **ERR_FILE_ACCESS** = -2
- static const int **ERR_FORMAT** = -3
- static const int **ERR_MEMORY_ACCESS** = -4
- static const int **ERR_NO_IMAGE** = -1
- static bool **register_images_done** = false

  True after fl_register_images() was called, false before.

Protected Member Functions

- void **clear_frames** ()
- bool **next_frame** ()
- void on_extension_data (Fl_GIF_Image::GIF_FRAME &f) override
- void on_frame_data (Fl_GIF_Image::GIF_FRAME &f) override
- void **scale_frame** ()
- void **set_frame** ()
- void **set_frame** (int frame)

Protected Member Functions inherited from Fl_GIF_Image

- **Fl_GIF_Image** ()

  The default constructor creates an empty GIF image.
- **Fl_GIF_Image** (const char *filename, bool anim)
- **Fl_GIF_Image** (const char *imagename, const unsigned char *data, const size_t length, bool anim)
- void **load** (const char *filename, bool anim)

  The protected load() methods are used by Fl_Anim_GIF_Image to request loading of animated GIF's.
- void **load** (const char *imagename, const unsigned char *data, const size_t length, bool anim)
- void **load_gif_** (class Fl_Image_Reader &rdr, bool anim=false)

Protected Member Functions inherited from Fl_Pixmap

- void **measure** ()

Protected Member Functions inherited from Fl_Image

- void **d** (int D)

  Sets the current image depth.
- void data (const char *const *p, int c)

  Sets the current data pointer and count of pointers in the array.
- void draw_empty (int X, int Y)

  The protected method draw_empty() draws a box with an X in it.
- int draw_scaled (int X, int Y, int W, int H)

  Draw the image to the current drawing surface rescaled to a given width and height.
- void h (int H)

  Sets the height of the image data.
- void ld (int LD)

  Sets the current line data size in bytes.
- void w (int W)

  Sets the width of the image data.

Static Protected Member Functions

- static void **cb_animate** (void *d)

Static Protected Member Functions inherited from Fl_Image

- static void **labeltype** (const Fl_Label *lo, int lx, int ly, int lw, int lh, Fl_Align la)
- static void **measure** (const Fl_Label *lo, int &lw, int &lh)

Additional Inherited Members

Public Attributes inherited from Fl_Pixmap

- int **alloc_data**

13.9.1 Detailed Description

The Fl_Anim_GIF_Image class supports loading, caching, and drawing of animated Compuserve GIF^{SM} images. The class loads all images contained in the file and animates them by cycling through them as defined by the delay times in the image file.
The user must supply an FLTK widget as "container" in order to see the animation by specifying it in the constructor or later using the canvas() method.

13.9.2 Member Enumeration Documentation

13.9.2.1 Flags

`enum Fl_Anim_GIF_Image::Flags`
When opening an Fl_Anim_GIF_Image there are some options that can be passed in a `flags` value.

Enumerator

DONT_START	This flag indicates to the loader that it should not start the animation immediately after successful load, which is the default. It can be started later using the start() method.
DONT_RESIZE_CANVAS	This flag indicates to the loader that it should not resize the canvas widget of the animation to the dimensions of the animation, which is the default. Needed for special use cases.
DONT_SET_AS_IMAGE	This flag indicates to the loader that it should not set the animation as image() member of the canvas widget, which is the default. Needed for special use cases.
OPTIMIZE_MEMORY	Often frames change just a small area of the animation canvas. This flag indicates to the loader to try using less memory by storing frame data not as canvas-sized images but use the sizes defined in the GIF file. The drawbacks are higher cpu usage during playback and maybe minor artifacts when resized.
LOG_FLAG	This flag can be used to print informations about the decoding process to the console.
DEBUG_FLAG	This flag can be used to print even more informations about the decoding process to the console.

13.9.3 Constructor & Destructor Documentation

13.9.3.1 Fl_Anim_GIF_Image() [1/2]

```
Fl_Anim_GIF_Image::Fl_Anim_GIF_Image (
            const char * filename,
            Fl_Widget * canvas = 0,
            unsigned short flags = 0 )
```

Load an animated GIF image from a file.
This constructor creates an animated image from a GIF-formatted file. Optionally it applies the canvas() method after successful load. If DONT_START is not specified in the `flags` parameter it calls start() after successful load.

Parameters

in	*filename*	path and name of GIF file in the file system
in	*canvas*	a widget that will show and animate the GIF, or NULL
in	*flags*	see Flags for details, or 0

Note

The GIF image must be decoupled from the canvas by calling `myGif->canvas(NULL);` before deleting the canvas.

13.9.3.2 Fl_Anim_GIF_Image() [2/2]

```
Fl_Anim_GIF_Image::Fl_Anim_GIF_Image (
            const char * imagename,
            const unsigned char * data,
            const size_t length,
            Fl_Widget * canvas = 0,
            unsigned short flags = 0 )
```

Load an animated GIF image from memory.
This constructor creates an animated image from a GIF-formatted block in memory. Optionally it applies the canvas() method after successful load. If DONT_START is not specified in the `flags` parameter it calls start() after successful load.
`imagename` can be NULL. If a name is given, the image is added to the list of shared images and will be available by that name.

Parameters

in	*imagename*	a name given to this image or NULL
in	*data*	pointer to the start of the GIF image in memory
in	*length*	length of the GIF image in memory
in	*canvas*	a widget that will show and animate the GIF, or NULL
in	*flags*	see Flags for details, or 0

Note

The GIF image must be decoupled from the canvas by calling `myGif->canvas(NULL);` before deleting the canvas.

13.9.3.3 ~Fl_Anim_GIF_Image()

```
Fl_Anim_GIF_Image::~Fl_Anim_GIF_Image ( )  [override]
```

Release the image and all cached data.
Also removes the animation timer.

13.9.4 Member Function Documentation

13.9.4.1 canvas() [1/2]

```
Fl_Widget * Fl_Anim_GIF_Image::canvas ( ) const
```

Gets the current widget, that is used to display the frame images.

Returns

a pointer to a widget

13.9.4.2 canvas() [2/2]

```
void Fl_Anim_GIF_Image::canvas (
            Fl_Widget * canvas,
            unsigned short flags = 0 )
```

Link the image back to a widget for automated animation.
This method sets current widget, that is used to display the frame images. The `flags` parameter specifies whether the canvas widget is resized to the animation dimensions and/or its image() method will be used to set the current frame image during animation.

Parameters

in	*canvas*	a pointer to the widget that will show the animation
in	*flags*	see Flags

Note

The GIF image must be decoupled from the canvas by calling `myGif->canvas(NULL);` before deleting the canvas.

13.9.4.3 canvas_h()

```
int Fl_Anim_GIF_Image::canvas_h ( ) const
```

Return the height of the animation canvas.

Returns

the width in pixel units

13.9.4.4 canvas_w()

```
int Fl_Anim_GIF_Image::canvas_w ( ) const
```

Return the width of the animation canvas.

Returns

the width in pixel units

13.9.4.5 color_average()

```
void Fl_Anim_GIF_Image::color_average (
            Fl_Color c,
            float i ) [override], [virtual]
```

Applies a color average to all frames.
The color_average() method averages the colors in the image with the provided FLTK color value.

Parameters

in	*c*	blend color
in	*i*	a value between 0.0 and 1.0 where 0 results in the blend color, and 1 returns the original image

Reimplemented from Fl_Image.

13.9.4.6 copy()

```
Fl_Image * Fl_Anim_GIF_Image::copy (
            int W,
            int H ) const [override], [virtual]
```

Copy and resize the animation frames.
The virtual copy() method makes a copy of the animated image and resizes all of its frame images to W x H using the current resize method.

Parameters

in	*W,H*	new size in FLTK pixel units

Returns

the resized copy of the animation

Reimplemented from Fl_Image.

13.9.4.7 delay() [1/2]

```
void Fl_Anim_GIF_Image::delay (
            int frame,
            double delay )
```

Set the delay of frame [0-frames() -1] in seconds.

Parameters

in	*frame*	index into frame list
in	*delay*	to next frame in seconds

13.9.4.8 delay() [2/2]

```
double Fl_Anim_GIF_Image::delay (
            int frame ) const
```

Return the delay of frame [0-frames() -1] in seconds.

Parameters

in	*frame*	index into frame list

Returns

delay to next frame in seconds

13.9.4.9 desaturate()

```
void Fl_Anim_GIF_Image::desaturate ( ) [override], [virtual]
```

Desaturate to all frames of the animation.
Reimplemented from Fl_Image.

13.9.4.10 draw()

```
void Fl_Anim_GIF_Image::draw (
            int x,
            int y,
            int w,
            int h,
            int cx = 0,
            int cy = 0 ) [override], [virtual]
```

Draw the current frame of the animation.

Parameters

in	*x,y,w,h*	target rectangle
in	*cx,cy*	source offset

Reimplemented from Fl_Image.

13.9.4.11 frame() [1/2]

```
int Fl_Anim_GIF_Image::frame ( ) const
```

Return the current frame.

Returns

the current frame index in the range for 0 to `frames()-1`.

-1 if the image has no frames.

13.9.4.12 frame() [2/2]

```
void Fl_Anim_GIF_Image::frame (
            int frame )
```

Set the current frame.

Parameters

in	*frame*	index into list of frames

13.9.4.13 frame_count()

```
int Fl_Anim_GIF_Image::frame_count (
            const char * name,
            const unsigned char * imgdata = NULL,
            size_t imglength = 0 )  [static]
```

Get the number of frames in a GIF file or in a GIF compressed data block.
The static frame_count() method is just a convenience method for getting the number of images (frames) stored in a GIF file.
As this count is not readily available in the GIF header, the whole GIF file has be parsed (which is done here by using a temporary Fl_Anim_GIF_Image object for simplicity). So this call may be slow with large files.
If `imgdata` is `NULL`, the image will be read from the file. Otherwise, it will be read from memory.

Parameters

in	*name*	path and name of GIF file in the file system, ignored when reading from memeory
in	*imgdata*	pointer to the start of the GIF image in memory, or `NULL` to read from a file
in	*imglength*	length of the GIF image in memory, or `0`

Returns

the number of frames in the animation

13.9.4.14 frame_h()

```
int Fl_Anim_GIF_Image::frame_h (
            int frame ) const
```

Return the frame dimensions of a frame.
Usefull only if loaded with 'optimize_mem' and the animation also has size optimized frames.

Parameters

in	*frame*	index into frame list

Returns

height in FLTK pixle units

13.9.4.15 frame_uncache() [1/2]

```
bool Fl_Anim_GIF_Image::frame_uncache ( ) const
```

Return the active frame_uncache() setting.

Returns

true if caching is disabled

13.9.4.16 frame_uncache() [2/2]

```
void Fl_Anim_GIF_Image::frame_uncache (
            bool uncache )
```

Use frame_uncache() to set or forbid frame image uncaching.
If frame uncaching is set, frame images are not offscreen cached for re-use and will be re-created every time they are displayed. This saves a lot of memory on the expense of cpu usage and should be carefully considered. Per default frame caching will be done.

Parameters

in	*uncache*	true to disable caching

13.9.4.17 frame_w()

```
int Fl_Anim_GIF_Image::frame_w (
            int frame ) const
```

Return the frame dimensions of a frame.
Usefull only if loaded with 'optimize_mem' and the animation also has size optimized frames.

Parameters

in	*frame*	index into frame list

Returns

width in FLTK pixle units

13.9.4.18 frame_x()

```
int Fl_Anim_GIF_Image::frame_x (
            int frame ) const
```

Return the frame position of a frame.
Usefull only if loaded with 'optimize_mem' and the animation also has size optimized frames.

Parameters

in	*frame*	index into frame list

Returns

x position in FLTK pixle units

13.9.4.19 frame_y()

```
int Fl_Anim_GIF_Image::frame_y (
            int frame ) const
```

Return the frame position of a frame.
Usefull only if loaded with 'optimize_mem' and the animation also has size optimized frames.

Parameters

in	*frame*	index into frame list

Returns

y position in FLTK pixle units

13.9.4.20 frames()

```
int Fl_Anim_GIF_Image::frames ( ) const
```

Get the number of frames in the animation.

Returns

the number of frames

13.9.4.21 image() [1/2]

```
Fl_Image * Fl_Anim_GIF_Image::image ( ) const
```

Return the current frame image.

Returns

a pointer to the image or NULL if this is not an animation.

13.9.4.22 image() [2/2]

```
Fl_Image * Fl_Anim_GIF_Image::image (
            int frame_ ) const
```

Return the image of the given frame index.

Parameters

in	*frame_*	index into list of frames

Returns

image data or NULL if the frame number is not valid.

13.9.4.23 is_animated()

```
bool Fl_Anim_GIF_Image::is_animated ( ) const
```

Check if this is a valid animation with more than one frame.
The `is_animated()` method is just a convenience method for testing the valid flag and the frame count beeing greater 1.

Returns

true if the animation is valid and has multiple frames.

13.9.4.24 load()

```
bool Fl_Anim_GIF_Image::load (
            const char * name,
            const unsigned char * imgdata = NULL,
            size_t imglength = 0 )
```

Load an animation from a file or from a memory block.
The load() method is either used from the constructor to load the image from the given file, or to re-load an existing animation from another file.

Parameters

in	*name*	path and name of GIF file in the file system, or the image name when reading from memory
in	*imgdata*	pointer to the start of the GIF image in memory, or `NULL` to read from a file
in	*imglength*	length of the GIF image in memory, or `0`

Returns

true if the animation loaded correctly

13.9.4.25 name()

```
const char * Fl_Anim_GIF_Image::name ( ) const
```

Return the name of the played file as specified in the constructor.
If read from a memory block, this returns the name of the animation.

Returns

pointer to a C string

13.9.4.26 next()

```
bool Fl_Anim_GIF_Image::next ( )
```

Show the next frame if the animation is stopped.

Returns

true if the animation has frames

13.9.4.27 on_extension_data()

```
void Fl_Anim_GIF_Image::on_extension_data (
            Fl_GIF_Image::GIF_FRAME & f )  [override], [protected], [virtual]
```

Reimplemented from Fl_GIF_Image.

13.9.4.28 on_frame_data()

```
void Fl_Anim_GIF_Image::on_frame_data (
            Fl_GIF_Image::GIF_FRAME & f )  [override], [protected], [virtual]
```

Reimplemented from Fl_GIF_Image.

13.9.4.29 playing()

```
bool Fl_Anim_GIF_Image::playing ( ) const  [inline]
```

Return if the animation is currently running or stopped.

Returns

true if the animation is running

13.9.4.30 resize() [1/2]

```
Fl_Anim_GIF_Image & Fl_Anim_GIF_Image::resize (
            double scale )
```

Resizes the image to the specified size, replacing the current image.
If DONT_RESIZE_CANVAS is not set, the canvas widget will also be resized.

Parameters

in	*scale*	rescale factor in relation to current size

13.9.4.31 resize() [2/2]

```
Fl_Anim_GIF_Image & Fl_Anim_GIF_Image::resize (
            int w,
            int h )
```

Resizes the image to the specified size, replacing the current image.
If DONT_RESIZE_CANVAS is not set, the canvas widget will also be resized.

Parameters

in	*w,h*	new size of teh naimtion frames

13.9.4.32 speed() [1/2]

```
double Fl_Anim_GIF_Image::speed ( ) const
```

Get the animation speed factor.

Returns

the current speed factor

13.9.4.33 speed() [2/2]

```
void Fl_Anim_GIF_Image::speed (
            double speed )
```

Set the animation speed factor.
The speed() method changes the playing speed to `speed` x original speed. E.g. to play at half speed call it with 0.5, for double speed with 2.

Parameters

in	*speed*	floating point speed factor

13.9.4.34 start()

```
bool Fl_Anim_GIF_Image::start ( )
```

The start() method (re-)starts the playing of the frames.

Returns

true if the animation has frames

13.9.4.35 stop()

`bool Fl_Anim_GIF_Image::stop ( )`
The stop() method stops the playing of the frames.

Returns

true if the animation has frames

13.9.4.36 uncache()

`void Fl_Anim_GIF_Image::uncache ( ) [override], [virtual]`
Uncache all cached image data now.
Re-implemented from Fl_Pixmap.
Reimplemented from Fl_Image.

13.9.4.37 valid()

`bool Fl_Anim_GIF_Image::valid ( ) const`
Check if animation is valid.

Returns

true if the class has successfully loaded and the image has at least one frame.

13.9.5 Member Data Documentation

13.9.5.1 loop

`bool Fl_Anim_GIF_Image::loop = true [static]`
The loop flag can be used to (dis-)allow loop count.
If set (which is the default), the animation will be stopped after the number of repeats specified in the GIF file (typically this count is set to 'forever' anyway). If cleared the animation will always be 'forever', regardless of what is specified in the GIF file.

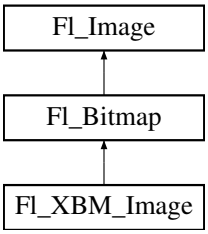
13.9.5.2 min_delay

`double Fl_Anim_GIF_Image::min_delay = 0. [static]`
The min_delay value can be used to set a minimum value for the frame delay for playback.
This is to prevent CPU hogs caused by images with very low delay rates. This is a global value for all Fl_Anim_GIF_Image objects.
The documentation for this class was generated from the following files:

- Fl_Anim_GIF_Image.H
- Fl_Anim_GIF_Image.cxx

13.10 Fl_Bitmap Class Reference

The Fl_Bitmap class supports caching and drawing of mono-color (bitmap) images.
`#include <Fl_Bitmap.H>`
Inheritance diagram for Fl_Bitmap:



Public Member Functions

- int **cache_h** () const
- int **cache_w** () const
- Fl_Image ∗ **copy** () const
- Fl_Image ∗ copy (int W, int H) const override

  Creates a resized copy of the image.
- void **draw** (int X, int Y)
- void draw (int X, int Y, int W, int H, int cx=0, int cy=0) override

  Draws the image to the current drawing surface with a bounding box.
- Fl_Bitmap (const char ∗bits, int bits_length, int W, int H)

  The constructors create a new bitmap from the specified bitmap data.
- Fl_Bitmap (const char ∗bits, int W, int H)

  The constructors create a new bitmap from the specified bitmap data.
- Fl_Bitmap (const uchar ∗bits, int bits_length, int W, int H)

  The constructors create a new bitmap from the specified bitmap data.
- Fl_Bitmap (const uchar ∗bits, int W, int H)

  The constructors create a new bitmap from the specified bitmap data.
- void label (Fl_Menu_Item ∗m) override

  This method is an obsolete way to set the image attribute of a menu item.
- void label (Fl_Widget ∗w) override

  This method is an obsolete way to set the image attribute of a widget or menu item.
- void uncache () override

  If the image has been cached for display, delete the cache data.
- virtual ∼**Fl_Bitmap** ()

  The destructor frees all memory and server resources that are used by the bitmap.

Public Member Functions inherited from Fl_Image

- virtual class Fl_Shared_Image ∗ as_shared_image ()

  Returns whether an image is an Fl_Shared_Image or not.
- virtual void color_average (Fl_Color c, float i)

  The color_average() method averages the colors in the image with the provided FLTK color value.
- Fl_Image ∗ copy () const

  Creates a copy of the image in the same size.
- int count () const

  Returns the number of data values associated with the image.
- int d () const

  Returns the image depth.
- const char ∗const ∗ data () const

  Returns a pointer to the current image data array.
- int **data_h** () const

  Returns the height of the image data.
- int **data_w** () const

Returns the width of the image data.

- virtual void desaturate ()

  The desaturate() method converts an image to grayscale.
- void draw (int X, int Y)

  Draws the image to the current drawing surface.
- int fail () const

  Returns a value that is not 0 if there is currently no image available.
- Fl_Image (int W, int H, int D)

  The constructor creates an empty image with the specified width, height, and depth.
- int h () const

  Returns the current image drawing height in FLTK units.
- void inactive ()

  The inactive() method calls color_average(FL_BACKGROUND_COLOR, 0.33f) to produce an image that appears grayed out.
- int ld () const

  Returns the current line data size in bytes.
- virtual void release ()

  Releases an Fl_Image - the same as '`delete this`'.
- virtual void scale (int width, int height, int proportional=1, int can_expand=0)

  Sets the drawing size of the image.
- int w () const

  Returns the current image drawing width in FLTK units.
- virtual ~**Fl_Image** ()

  The destructor is a virtual method that frees all memory used by the image.

Public Attributes

- int **alloc_array**

  Non-zero if array points to bitmap data allocated internally.
- const uchar * **array**

  pointer to raw bitmap data

Friends

- class **Fl_Graphics_Driver**

Additional Inherited Members

Static Public Member Functions inherited from Fl_Image

- static Fl_Labeltype **define_FL_IMAGE_LABEL** ()
- static Fl_RGB_Scaling **RGB_scaling** ()

  Returns the currently used RGB image scaling method.
- static void RGB_scaling (Fl_RGB_Scaling)

  Sets the RGB image scaling method used for copy(int, int).
- static Fl_RGB_Scaling **scaling_algorithm** ()

  Gets what algorithm is used when resizing a source image to draw it.
- static void scaling_algorithm (Fl_RGB_Scaling algorithm)

  Sets what algorithm is used when resizing a source image to draw it.

Static Public Attributes inherited from Fl_Image

- static const int **ERR_FILE_ACCESS** = -2
- static const int **ERR_FORMAT** = -3
- static const int **ERR_MEMORY_ACCESS** = -4
- static const int **ERR_NO_IMAGE** = -1
- static bool **register_images_done** = false

  True after fl_register_images() was called, false before.

Protected Member Functions inherited from Fl_Image

- void **d** (int D)

  Sets the current image depth.
- void data (const char ∗const ∗p, int c)

  Sets the current data pointer and count of pointers in the array.
- void draw_empty (int X, int Y)

  The protected method draw_empty() draws a box with an X in it.
- int draw_scaled (int X, int Y, int W, int H)

  Draw the image to the current drawing surface rescaled to a given width and height.
- void h (int H)

  Sets the height of the image data.
- void ld (int LD)

  Sets the current line data size in bytes.
- void w (int W)

  Sets the width of the image data.

Static Protected Member Functions inherited from Fl_Image

- static void **labeltype** (const Fl_Label ∗lo, int lx, int ly, int lw, int lh, Fl_Align la)
- static void **measure** (const Fl_Label ∗lo, int &lw, int &lh)

13.10.1 Detailed Description

The Fl_Bitmap class supports caching and drawing of mono-color (bitmap) images.
Images are drawn using the current color.

13.10.2 Constructor & Destructor Documentation

13.10.2.1 Fl_Bitmap() [1/4]

```
Fl_Bitmap::Fl_Bitmap (
            const uchar * bits,
            int W,
            int H )  [inline]
```

The constructors create a new bitmap from the specified bitmap data.

See also

Fl_Bitmap(const uchar ∗bits, int bits_length, int W, int H)

13.10.2.2 Fl_Bitmap() [2/4]

```
Fl_Bitmap::Fl_Bitmap (
            const char * bits,
            int W,
            int H )  [inline]
```

The constructors create a new bitmap from the specified bitmap data.

See also

Fl_Bitmap(const char *bits, int bits_length, int W, int H)

13.10.2.3 Fl_Bitmap() [3/4]

```
Fl_Bitmap::Fl_Bitmap (
            const uchar * bits,
            int bits_length,
            int W,
            int H )
```

The constructors create a new bitmap from the specified bitmap data.
If the provided array is too small to contain all the image data, the constructor will not generate the bitmap to avoid illegal memory read access and instead set `data` to NULL and `ld` to `ERR_MEMORY_ACCESS`.

Parameters

bits	bitmap data, one pixel per bit, rows are rounded to the next byte
bits_length	length of the `bits` array in bytes
W	image width in pixels
H	image height in pixels

See also

Fl_Bitmap(const char *bits, int bits_length, int W, int H), Fl_Bitmap(const uchar *bits, int W, int H)

13.10.2.4 Fl_Bitmap() [4/4]

```
Fl_Bitmap::Fl_Bitmap (
            const char * bits,
            int bits_length,
            int W,
            int H )
```

The constructors create a new bitmap from the specified bitmap data.
If the provided array is too small to contain all the image data, the constructor will not generate the bitmap to avoid illegal memory read access and instead set `data` to NULL and `ld` to `ERR_MEMORY_ACCESS`.

Parameters

bits	bitmap data, one pixel per bit, rows are rounded to the next byte
bits_length	length of the `bits` array in bytes
W	image width in pixels
H	image height in pixels

See also

Fl_Bitmap(const uchar *bits, int bits_length, int W, int H), Fl_Bitmap(const char *bits, int W, int H)

13.10.3 Member Function Documentation

13.10.3.1 copy()

```
Fl_Image * Fl_Bitmap::copy (
            int W,
            int H ) const  [override], [virtual]
```

Creates a resized copy of the image.

It is recommended not to call this member function to reduce the size of an image to the size of the area where this image will be drawn, and to use Fl_Image::scale() instead.
The new image should be released when you are done with it.
Note: since FLTK 1.4.0 you can use Fl_Image::release() for all types of images (i.e. all subclasses of Fl_Image) instead of operator *delete* for Fl_Image's and Fl_Image::release() for Fl_Shared_Image's.
The new image data will be converted to the requested size. RGB images are resized using the algorithm set by Fl_Image::RGB_scaling().
For the new image the following equations are true:

- w() == data_w() == `W`
- h() == data_h() == `H`

Parameters

in	*W,H*	Requested width and height of the new image

Note

The returned image can be safely cast to the same image type as that of the source image provided this type is one of Fl_RGB_Image, Fl_SVG_Image, Fl_Pixmap, Fl_Bitmap, Fl_Tiled_Image, Fl_Anim_GIF_Image and Fl_Shared_Image. Returned objects copied from images of other, derived, image classes belong to the parent class appearing in this list. For example, the copy of an Fl_GIF_Image is an object of class Fl_Pixmap.

Since FLTK 1.4.0 this method is 'const'. If you derive your own class from Fl_Image or any subclass your overridden methods of 'Fl_Image::copy() const' and 'Fl_Image::copy(int, int) const' **must** also be 'const' for inheritance to work properly. This is different than in FLTK 1.3.x and earlier where these methods have not been 'const'.

Reimplemented from Fl_Image.

13.10.3.2 draw()

```
void Fl_Bitmap::draw (
            int X,
            int Y,
            int W,
            int H,
            int cx = 0,
            int cy = 0 )  [override], [virtual]
```

Draws the image to the current drawing surface with a bounding box.
Arguments `X,Y,W,H` specify a bounding box for the image, with the origin (upper-left corner) of the image offset by the `cx` and `cy` arguments.
Another way to see what part of the image gets drawn and where, is to consider this alternative writing producing the same output:

```
fl_push_clip(X,Y,W,H);
this->draw(X - cx, Y - cy);
fl_pop_clip();
```

Repeated calls to this member function with the same image but varying `W, H, cx`, or `cy` arguments can be more efficiently processed using the above alternative writing.
Reimplemented from Fl_Image.

13.10.3.3 label() [1/2]

```
void Fl_Bitmap::label (
            Fl_Menu_Item * m )  [override], [virtual]
```

This method is an obsolete way to set the image attribute of a menu item.

Deprecated Please use Fl_Menu_Item::image() instead.

Reimplemented from Fl_Image.

13.10.3.4 label() [2/2]

```
void Fl_Bitmap::label (
            Fl_Widget * widget )  [override], [virtual]
```

This method is an obsolete way to set the image attribute of a widget or menu item.

Deprecated Please use Fl_Widget::image() or Fl_Widget::deimage() instead.

Reimplemented from Fl_Image.

13.10.3.5 uncache()

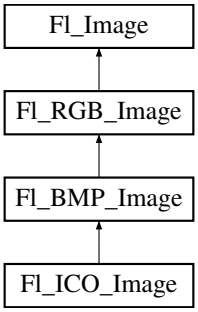
```
void Fl_Bitmap::uncache ( )  [override], [virtual]
```

If the image has been cached for display, delete the cache data.
This allows you to change the data used for the image and then redraw it without recreating an image object.
Reimplemented from Fl_Image.
The documentation for this class was generated from the following files:

- Fl_Bitmap.H
- Fl_Bitmap.cxx

13.11 Fl_BMP_Image Class Reference

The Fl_BMP_Image class supports loading, caching, and drawing of Windows Bitmap (BMP) image files.
`#include <Fl_BMP_Image.H>`
Inheritance diagram for Fl_BMP_Image:



Public Member Functions

- Fl_BMP_Image (const char *filename)

  This constructor loads the named BMP image from the given BMP filename.
- Fl_BMP_Image (const char *imagename, const unsigned char *data, const long length=-1)

  This constructor loads a BMP image from memory.

Public Member Functions inherited from Fl_RGB_Image

- virtual Fl_SVG_Image * as_svg_image ()

  Returns whether an RGB image is an Fl_SVG_Image or not.
- virtual const Fl_SVG_Image * as_svg_image () const

  Returns whether an RGB image is an Fl_SVG_Image or not.
- int **cache_h** () const
- int **cache_w** () const
- void color_average (Fl_Color c, float i) override

  The color_average() method averages the colors in the image with the provided FLTK color value.
- Fl_Image * **copy** () const
- Fl_Image * copy (int W, int H) const override

Creates a resized copy of the image.

- void desaturate () override

  The desaturate() method converts an image to grayscale.
- void **draw** (int X, int Y)
- void draw (int X, int Y, int W, int H, int cx=0, int cy=0) override

  Draws the image to the current drawing surface with a bounding box.
- Fl_RGB_Image (const Fl_Pixmap *pxm, Fl_Color bg=FL_GRAY)

  The constructor creates a new RGBA image from the specified Fl_Pixmap.
- Fl_RGB_Image (const uchar *bits, int bits_length, int W, int H, int D, int LD)

  The constructor creates a new image from the specified data.
- Fl_RGB_Image (const uchar *bits, int W, int H, int D=3, int LD=0)

  The constructor creates a new image from the specified data.
- void label (Fl_Menu_Item *m) override

  This method is an obsolete way to set the image attribute of a menu item.
- void label (Fl_Widget *w) override

  This method is an obsolete way to set the image attribute of a widget or menu item.
- virtual void normalize ()

  Makes sure the object is fully initialized.
- void uncache () override

  If the image has been cached for display, delete the cache data.
- ~**Fl_RGB_Image** () override

  The destructor frees all memory and server resources that are used by the image.

Public Member Functions inherited from Fl_Image

- virtual class Fl_Shared_Image * as_shared_image ()

  Returns whether an image is an Fl_Shared_Image or not.
- Fl_Image * copy () const

  Creates a copy of the image in the same size.
- int count () const

  Returns the number of data values associated with the image.
- int d () const

  Returns the image depth.
- const char *const * data () const

  Returns a pointer to the current image data array.
- int **data_h** () const

  Returns the height of the image data.
- int **data_w** () const

  Returns the width of the image data.
- void draw (int X, int Y)

  Draws the image to the current drawing surface.
- int fail () const

  Returns a value that is not 0 if there is currently no image available.
- Fl_Image (int W, int H, int D)

  The constructor creates an empty image with the specified width, height, and depth.
- int h () const

  Returns the current image drawing height in FLTK units.
- void inactive ()

  The inactive() method calls color_average(FL_BACKGROUND_COLOR, 0.33f) to produce an image that appears grayed out.
- int ld () const

Returns the current line data size in bytes.

- virtual void release ()

  Releases an Fl_Image - the same as '`delete this`'.

- virtual void scale (int width, int height, int proportional=1, int can_expand=0)

  Sets the drawing size of the image.

- int w () const

  Returns the current image drawing width in FLTK units.

- virtual ~**Fl_Image** ()

  The destructor is a virtual method that frees all memory used by the image.

Protected Member Functions

- void **load_bmp_** (class Fl_Image_Reader &rdr, int ico_height=0, int ico_width=0)

Protected Member Functions inherited from Fl_Image

- void **d** (int D)

  Sets the current image depth.

- void data (const char *const *p, int c)

  Sets the current data pointer and count of pointers in the array.

- void draw_empty (int X, int Y)

  The protected method draw_empty() draws a box with an X in it.

- int draw_scaled (int X, int Y, int W, int H)

  Draw the image to the current drawing surface rescaled to a given width and height.

- void h (int H)

  Sets the height of the image data.

- void ld (int LD)

  Sets the current line data size in bytes.

- void w (int W)

  Sets the width of the image data.

Additional Inherited Members

Static Public Member Functions inherited from Fl_RGB_Image

- static size_t max_size ()

  Returns the maximum allowed image size in bytes when creating an Fl_RGB_Image object.

- static void max_size (size_t size)

  Sets the maximum allowed image size in bytes when creating an Fl_RGB_Image object.

Static Public Member Functions inherited from Fl_Image

- static Fl_Labeltype **define_FL_IMAGE_LABEL** ()
- static Fl_RGB_Scaling **RGB_scaling** ()

  Returns the currently used RGB image scaling method.

- static void RGB_scaling (Fl_RGB_Scaling)

  Sets the RGB image scaling method used for copy(int, int).

- static Fl_RGB_Scaling **scaling_algorithm** ()

  Gets what algorithm is used when resizing a source image to draw it.

- static void scaling_algorithm (Fl_RGB_Scaling algorithm)

  Sets what algorithm is used when resizing a source image to draw it.

Public Attributes inherited from Fl_RGB_Image

- int **alloc_array**

  If non-zero, the object's data array is delete[]'d when deleting the object.
- const uchar * array

  Points to the start of the object's data array.

Static Public Attributes inherited from Fl_Image

- static const int **ERR_FILE_ACCESS** = -2
- static const int **ERR_FORMAT** = -3
- static const int **ERR_MEMORY_ACCESS** = -4
- static const int **ERR_NO_IMAGE** = -1
- static bool **register_images_done** = false

  True after fl_register_images() was called, false before.

Static Protected Member Functions inherited from Fl_Image

- static void **labeltype** (const Fl_Label *lo, int lx, int ly, int lw, int lh, Fl_Align la)
- static void **measure** (const Fl_Label *lo, int &lw, int &lh)

13.11.1 Detailed Description

The Fl_BMP_Image class supports loading, caching, and drawing of Windows Bitmap (BMP) image files.

13.11.2 Constructor & Destructor Documentation

13.11.2.1 Fl_BMP_Image() [1/2]

```
Fl_BMP_Image::Fl_BMP_Image (
            const char * filename )
```

This constructor loads the named BMP image from the given BMP filename.
The destructor frees all memory and server resources that are used by the image.
Use Fl_Image::fail() to check if Fl_BMP_Image failed to load. fail() returns ERR_FILE_ACCESS if the file could not be opened or read, ERR_FORMAT if the BMP format could not be decoded, and ERR_NO_IMAGE if the image could not be loaded for another reason.

Parameters

in	*filename*	a full path and name pointing to a BMP file.

See also

Fl_BMP_Image::Fl_BMP_Image(const char* imagename, const unsigned char *data, const long length = -1);

13.11.2.2 Fl_BMP_Image() [2/2]

```
Fl_BMP_Image::Fl_BMP_Image (
            const char * imagename,
            const unsigned char * data,
            const long length = -1 )
```

This constructor loads a BMP image from memory.
Construct an image from a block of memory inside the application. Fluid offers "binary data" chunks as a great way to add image data into the C++ source code. `imagename` can be NULL. If a name is given, the image is added to the list of shared images and will be available by that name.
The destructor frees all memory and server resources that are used by the image.
The (new and optional) third parameter `length` **should** be used so buffer overruns (i.e. truncated images) can be checked. See note below.

If `length` is not used

- it defaults to -1 (unlimited size)
- buffer overruns will not be checked.

Note

The optional parameter `length` is available since FLTK 1.4.0. Not using it is deprecated and old code should be modified to use it. This parameter will likely become mandatory in a future FLTK version.

Use Fl_Image::fail() to check if Fl_BMP_Image failed to load. fail() returns ERR_FILE_ACCESS if the image could not be read from memory, ERR_FORMAT if the BMP format could not be decoded, and ERR_NO_IMAGE if the image could not be loaded for another reason.

Parameters

in	*imagename*	A name given to this image or NULL
in	*data*	Pointer to the start of the BMP image in memory.
in	*length*	Length of the BMP image in memory.

See also

Fl_BMP_Image::Fl_BMP_Image(const char ∗filename)

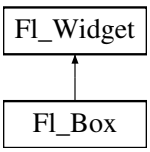
Fl_Shared_Image

The documentation for this class was generated from the following files:

- Fl_BMP_Image.H
- Fl_BMP_Image.cxx

13.12 Fl_Box Class Reference

This widget simply draws its box, and possibly its label.
`#include <Fl_Box.H>`
Inheritance diagram for Fl_Box:



Public Member Functions

- Fl_Box (Fl_Boxtype B, int X, int Y, int W, int H, const char ∗L)

  Creates a new Fl_Box widget with the given boxtype, coordinates, size, and label.
- Fl_Box (int X, int Y, int W, int H, const char ∗L=0)

  Creates a new Fl_Box widget with the given coordinates, size, and label.
- int handle (int) override

  Handles the specified event.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

Activates the widget.

- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

Marks the value of the widget as unchanged.

- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.

- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

Sets whether this widget needs a keyboard.

- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group ∗ parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group ∗p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.

- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void draw () override

  Draws the widget.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 , AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.12.1 Detailed Description

This widget simply draws its box, and possibly its label.
Putting it before some other widgets and making it big enough to surround them will let you draw a frame around them.

13.12.2 Constructor & Destructor Documentation

13.12.2.1 Fl_Box() [1/2]

```
Fl_Box::Fl_Box (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

Creates a new Fl_Box widget with the given coordinates, size, and label.
This constructor sets box() to FL_NO_BOX, which means it is invisible.
However such widgets are useful as placeholders or Fl_Group::resizable() values. To change the box to something visible, use box(Fl_Boxtype).
The destructor removes the box from its parent group.

Parameters

in	*X,Y*	the position of the widget relative to the enclosing window
in	*W,H*	size of the widget in pixels
in	*L*	optional text for the widget label (default: no label)

See also

Fl_Box(Fl_Boxtype b, int X, int Y, int W, int H, const char *L)

13.12.2.2 Fl_Box() [2/2]

```
Fl_Box::Fl_Box (
            Fl_Boxtype B,
            int X,
            int Y,
            int W,
```

```
int H,
const char * L )
```

Creates a new Fl_Box widget with the given boxtype, coordinates, size, and label.
This constructor sets box() to the given Fl_Boxtype `B`.
You must also specify a label but it can be `nullptr` (0, NULL) if you don't want or need a visible label.
The destructor removes the box from its parent group.

Parameters

in	*B*	boxtype of the widget
in	*X,Y*	the position of the widget relative to the enclosing window
in	*W,H*	size of the widget in pixels
in	*L*	optional text for the widget label (see description)

See also

Fl_Box(int X, int Y, int W, int H, const char *L)

13.12.3 Member Function Documentation

13.12.3.1 draw()

```
void Fl_Box::draw ( ) [override], [protected], [virtual]
```

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();                 // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

13.12.3.2 handle()

```
int Fl_Box::handle (
            int event ) [override], [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to *override* the behavior of the base class. This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to *filter* event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you *consumed* the event by returning 1 even if you didn't *do* anything with it.

Parameters

in	*event*	the kind of event received

Return values

0	if the event was not used or understood
1	if the event was used and can be deleted

See also

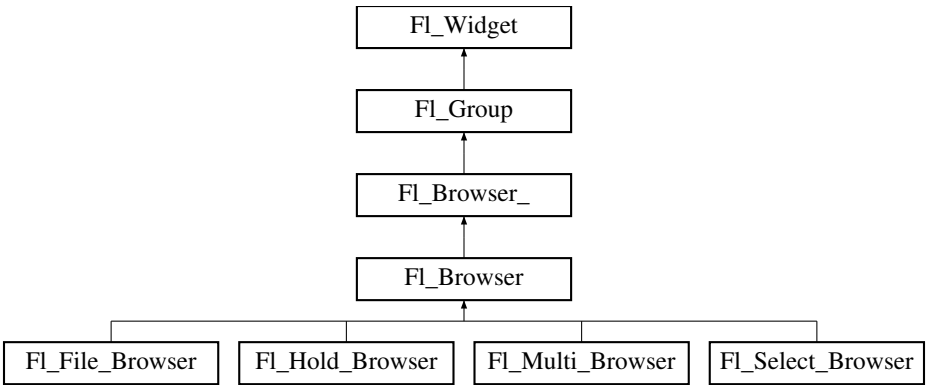
Fl_Event

Reimplemented from Fl_Widget.
The documentation for this class was generated from the following files:

- Fl_Box.H
- Fl_Box.cxx

13.13 Fl_Browser Class Reference

The Fl_Browser widget displays a scrolling list of text lines, and manages all the storage for the text.
`#include <Fl_Browser.H>`
Inheritance diagram for Fl_Browser:



Public Types

- enum Fl_Line_Position { **TOP** , **BOTTOM** , **MIDDLE** }

  For internal use only?

Public Types inherited from Fl_Browser_

- enum {
  HORIZONTAL = 1 , VERTICAL = 2 , BOTH = 3 , ALWAYS_ON = 4 ,
  HORIZONTAL_ALWAYS = 5 , VERTICAL_ALWAYS = 6 , BOTH_ALWAYS = 7 }

  Values for has_scrollbar().

Public Member Functions

- void add (const char ∗newtext, void ∗d=0)

  Adds a new line to the end of the browser.
- void bottomline (int line)

  Scrolls the browser so the bottom item in the browser is showing the specified `line`.
- void clear ()

  Removes all the lines in the browser.
- char column_char () const

  Gets the current column separator character.
- void column_char (char c)

  Sets the column separator to c.
- const int ∗ column_widths () const

  Gets the current column width array.
- void column_widths (const int ∗arr)

Sets the current array to `arr`.

- void * data (int line) const

  Returns the user data() for specified `line`.
- void data (int line, void *d)

  Sets the user data for specified `line` to `d`.
- void display (int line, int val=1)

  For back compatibility.
- int displayed (int line) const

  Returns non-zero if `line` has been scrolled to a position where it is being displayed.
- Fl_Browser (int X, int Y, int W, int H, const char *L=0)

  The constructor makes an empty browser.
- char format_char () const

  Gets the current format code prefix character, which by default is '@'.
- void format_char (char c)

  Sets the current format code prefix character to `c`.
- void hide () override

  Hides the entire Fl_Browser widget – opposite of show().
- void hide (int line)

  Makes `line` invisible, preventing selection by the user.
- Fl_Image * icon (int line) const

  Returns the icon currently defined for `line`.
- void icon (int line, Fl_Image *icon)

  Set the image icon for `line` to the value `icon`.
- void insert (int line, const char *newtext, void *d=0)

  Insert a new entry whose label is `newtext` above given `line`, optional data `d`.
- void lineposition (int line, Fl_Line_Position pos)

  Updates the browser so that `line` is shown at position `pos`.
- int load (const char *filename)

  Clears the browser and reads the file, adding each line from the file to the browser.
- void make_visible (int line)

  Make the item at the specified `line` visible().
- void middleline (int line)

  Scrolls the browser so the middle item in the browser is showing the specified `line`.
- void move (int to, int from)

  Line `from` is removed and reinserted at `to`.
- void remove (int line)

  Remove entry for given `line` number, making the browser one line shorter.
- void remove_icon (int line)

  Removes the icon for `line`.
- void **replace** (int a, const char *b)

  For back compatibility only.
- int select (int line, int val=1)

  Sets the selection state of the item at `line` to the value `val`.
- int selected (int line) const

  Returns 1 if specified `line` is selected, 0 if not.
- void show () override

  Shows the entire Fl_Browser widget – opposite of hide().
- void show (int line)

  Makes `line` visible, and available for selection by user.
- int size () const

  Returns how many lines are in the browser.

- void **size** (int W, int H)
- void swap (int a, int b)

  Swaps two browser lines `a` and `b`.
- const char * text (int line) const

  Returns the label text for the specified `line`.
- void text (int line, const char *newtext)

  Sets the text for the specified `line` to `newtext`.
- Fl_Fontsize **textsize** () const

  Gets the default text size (in pixels) for the lines in the browser.
- void textsize (Fl_Fontsize newSize)

  Sets the default text size (in pixels) for the lines in the browser to `newSize`.
- int topline () const

  Returns the line that is currently visible at the top of the browser.
- void topline (int line)

  Scrolls the browser so the top item in the browser is showing the specified `line`.
- int value () const

  Returns the line number of the currently selected line, or 0 if none selected.
- void value (int line)

  Sets the browser's value(), which selects the specified `line`.
- int visible (int line) const

  Returns non-zero if the specified `line` is visible, 0 if hidden.
- ~**Fl_Browser** ()

  The destructor deletes all list items and destroys the browser.

Public Member Functions inherited from Fl_Browser_

- int deselect (int docallbacks=0)

  Deselects all items in the list and returns 1 if the state changed or 0 if it did not.
- void display (void *item)

  Displays the `item`, scrolling the list as necessary.
- int handle (int event) override

  Handles the `event` within the normal widget bounding box.
- uchar **has_scrollbar** () const

  Returns the current scrollbar mode, see Fl_Browser_::has_scrollbar(uchar)
- void has_scrollbar (uchar mode)

  Sets whether the widget should have scrollbars or not (default Fl_Browser_::BOTH).
- int hposition () const

  Gets the horizontal scroll position of the list as a pixel position `pos`.
- void hposition (int)

  Sets the horizontal scroll position of the list to pixel position `pos`.
- int linespacing () const

  Return the height of additional spacing between browser lines.
- void linespacing (int pixels)

  Add some space between browser lines.
- int position () const
- void position (int pos)
- void **position** (int x, int y)
- void resize (int X, int Y, int W, int H) override

  Repositions and/or resizes the browser.
- void scrollbar_left ()

  Moves the vertical scrollbar to the lefthand side of the list.

- void scrollbar_right ()

  Moves the vertical scrollbar to the righthand side of the list.
- int scrollbar_size () const

  Gets the current size of the scrollbars' troughs, in pixels.
- void scrollbar_size (int newSize)

  Sets the pixel size of the scrollbars' troughs to `newSize`, in pixels.
- int select (void *item, int val=1, int docallbacks=0)

  Sets the selection state of `item` to `val`, and returns 1 if the state changed or 0 if it did not.
- int select_only (void *item, int docallbacks=0)

  Selects `item` and returns 1 if the state changed or 0 if it did not.
- void sort (int flags=0)

  Sort the items in the browser based on `flags`.
- Fl_Color **textcolor** () const

  Gets the default text color for the lines in the browser.
- void **textcolor** (Fl_Color col)

  Sets the default text color for the lines in the browser to color `col`.
- Fl_Font textfont () const

  Gets the default text font for the lines in the browser.
- void **textfont** (Fl_Font font)

  Sets the default text font for the lines in the browser to `font`.
- Fl_Fontsize **textsize** () const

  Gets the default text size (in pixels) for the lines in the browser.
- void **textsize** (Fl_Fontsize newSize)

  Sets the default text size (in pixels) for the lines in the browser to `size`.
- int vposition () const

  Gets the vertical scroll position of the list as a pixel position `pos`.
- void vposition (int pos)

  Sets the vertical scroll position of the list to pixel position `pos`.

Public Member Functions inherited from Fl_Group

- Fl_Widget *& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget *o)

  See void Fl_Group::add(Fl_Widget &w)
- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget *const * array () const

  Returns a pointer to the internal array of children.
- Fl_Group const * as_group () const override
- Fl_Group * as_group () override

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const

  Returns the n'th child.
- int **children** () const

  Returns how many child widgets the group has.
- void clear ()

Deletes all child widgets from memory recursively.

- unsigned int clip_children () const

  Returns the current clipping mode.
- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)

  Removes the widget at `index` from the group and deletes it.
- void end ()

  Exactly the same as current(this->parent()).
- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*
- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.
- void **forms_end** ()

  This is for forms compatibility only.
- int handle (int) override

  Handles the specified event.
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

  This does insert(w, find(before)).
- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.
- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.
- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- void resize (int, int, int, int) override

  Resizes the Fl_Group widget and all of its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

Returns whether the widget and all of its parents are active.

- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ **as_gl_window** () const
- virtual Fl_Window ∗ as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ **as_window** () const
- void bind_deimage (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗cb, void ∗p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.

- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

Gets the image that is used as part of the widget label when in the active state.

- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.

- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

Sets the user data for this widget.

- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- FL_BLINE ∗ _remove (int line)

  Removes the item at the specified `line`.
- const void ∗ **bline_data** (const FL_BLINE ∗b) const

  Get FL_BLINE data.
- void ∗& **bline_data** (FL_BLINE ∗b) const

  Get writable reference to FL_BLINE data.
- char **bline_flags** (const FL_BLINE ∗b) const

  Get FL_BLINE flags.
- char & **bline_flags** (FL_BLINE ∗b) const

  Get writable reference to FL_BLINE flags.
- const Fl_Image ∗ **bline_icon** (const FL_BLINE ∗b) const

  Get FL_BLINE icon.
- Fl_Image ∗& **bline_icon** (FL_BLINE ∗b) const

  Get writable reference to FL_BLINE icon.
- short bline_length (const FL_BLINE ∗b) const

  Get FL_BLINE allocated text buffer size (excl.
- const char ∗ **bline_txt** (const FL_BLINE ∗b) const

  Get FL_BLINE text.
- char ∗ **bline_txt** (FL_BLINE ∗b) const

  Get writable reference to FL_BLINE text.
- FL_BLINE ∗ find_line (int line) const

Returns the item for specified `line`.

- int full_height () const override

  The height of the entire list of all visible() items in pixels.
- int incr_height () const override

  The default 'average' item height (including inter-item spacing) in pixels.
- void insert (int line, FL_BLINE *item)

  Insert specified `item` above `line`.
- void * item_at (int line) const override

  Return the item at specified `line`.
- void item_draw (void *item, int X, int Y, int W, int H) const override

  Draws `item` at the position specified by `X Y W H`.
- void * item_first () const override

  Returns the very first item in the list.
- int item_height (void *item) const override

  Returns height of `item` in pixels.
- void * item_last () const override

  Returns the very last item in the list.
- void * item_next (void *item) const override

  Returns the next item after `item`.
- void * item_prev (void *item) const override

  Returns the previous item before `item`.
- void item_select (void *item, int val) override

  Change the selection state of `item` to the value `val`.
- int item_selected (void *item) const override

  See if `item` is selected.
- void item_swap (void *a, void *b) override

  Swap the items `a` and `b`.
- const char * item_text (void *item) const override

  Returns the label text for `item`.
- int item_width (void *item) const override

  Returns width of `item` in pixels.
- int lineno (void *item) const

  Returns line number corresponding to `item`, or zero if not found.
- void swap (FL_BLINE *a, FL_BLINE *b)

  Swap the two items `a` and `b`.

Protected Member Functions inherited from Fl_Browser_

- void bbox (int &X, int &Y, int &W, int &H) const

  Returns the bounding box for the interior of the list's display window, inside the scrollbars.
- void deleting (void *item)

  This method should be used when `item` is being deleted from the list.
- int displayed (void *item) const

  Returns non-zero if `item` has been scrolled to a position where it is being displayed.
- void draw () override

  Draws the list within the normal widget bounding box.
- void * find_item (int ypos)

  This method returns the item under mouse y position `ypos`.
- Fl_Browser_ (int X, int Y, int W, int H, const char *L=0)

  The constructor makes an empty browser.
- virtual int full_width () const

This method may be provided by the subclass to indicate the full width of the item list, in pixels.

- void inserting (void *a, void *b)

  This method should be used when an item is in the process of being inserted into the list.
- virtual int item_quick_height (void *item) const

  This method may be provided by the subclass to return the height of the `item`, in pixels.
- int leftedge () const

  This method returns the X position of the left edge of the list area after adjusting for the scrollbar and border, if any.
- void new_list ()

  This method should be called when the list data is completely replaced or cleared.
- void redraw_line (void *item)

  This method should be called when the contents of `item` has changed, but not its height.
- void redraw_lines ()

  This method will cause the entire list to be redrawn.
- void replacing (void *a, void *b)

  This method should be used when item `a` is being replaced by item `b`.
- void * selection () const

  Returns the item currently selected, or NULL if there is no selection.
- void swapping (void *a, void *b)

  This method should be used when two items `a` and `b` are being swapped.
- void * **top** () const

  Returns the item that appears at the top of the list.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()

  Returns the internal array of widget sizes and positions.
- void draw () override

  Draws the widget.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget *, int)

  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)

  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)

  Allow derived groups to act when a child widget is removed from the group.
- int * sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Static Protected Attributes

- static constexpr char **BLINE_NOTDISPLAYED** = 2
- static constexpr char **BLINE_SELECTED** = 1

Additional Inherited Members

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()

  Returns the currently active group.
- static void current (Fl_Group *g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Public Attributes inherited from Fl_Browser_

- Fl_Scrollbar hscrollbar

  Horizontal scrollbar.
- Fl_Scrollbar scrollbar

  Vertical scrollbar.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.13.1 Detailed Description

The Fl_Browser widget displays a scrolling list of text lines, and manages all the storage for the text.
This is not a text editor or spreadsheet! But it is useful for showing a vertical list of named objects to the user.

Figure 13.2 Fl_Hold_Browser

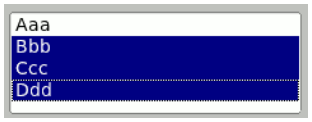


Figure 13.3 Fl_Multi_Browser

Each line in the browser is identified by number. *The numbers start at one* (this is so that zero can be reserved for "no line" in the selective browsers). *Unless otherwise noted, the methods do not check to see if the passed line number is in range and legal. It must always be greater than zero and <= size().*
Each line contains a null-terminated string of text and a void * data pointer. The text string is displayed, the void * pointer can be used by the callbacks to reference the object the text describes.
The base class does nothing when the user clicks on it. The subclasses Fl_Select_Browser, Fl_Hold_Browser, and Fl_Multi_Browser react to user clicks to select lines in the browser and do callbacks.
The base class Fl_Browser_ provides the scrolling and selection mechanisms of this and all the subclasses, but the dimensions and appearance of each item are determined by the subclass. You can use Fl_Browser_ to display information other than text, or text that is dynamically produced from your own data structures. If you find that loading the browser is a lot of work or is inefficient, you may want to make a subclass of Fl_Browser_.
Some common coding patterns used for working with Fl_Browser:

```
// How to loop through all the items in the browser
for ( int t=1; t<=browser->size(); t++ ) {       // index 1 based..!
    printf("item #%d, label='%s'\n", t, browser->text(t));
}
```

Note: If you are *subclassing* Fl_Browser, it's more efficient to use the protected methods item_first() and item_next(), since Fl_Browser internally uses linked lists to manage the browser's items. For more info, see find_item(int).

13.13.2 Constructor & Destructor Documentation

13.13.2.1 Fl_Browser()

```
Fl_Browser::Fl_Browser (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

The constructor makes an empty browser.

Parameters

in	*X,Y,W,H*	position and size.
in	*L*	label string, may be NULL.

13.13.3 Member Function Documentation

13.13.3.1 _remove()

```
FL_BLINE * Fl_Browser::_remove (
            int line )  [protected]
```

Removes the item at the specified `line`.
Caveat: See efficiency note in find_line(). You must call redraw() to make any changes visible.

Parameters

in	*line*	The line number to be removed. (1 based) Must be in range!

Returns

Pointer to browser item that was removed (and is no longer valid).

See also

add(), insert(), remove(), swap(int,int), clear()

13.13.3.2 add()

```
void Fl_Browser::add (
            const char * newtext,
            void * d = 0 )
```

Adds a new line to the end of the browser.
The text string `newtext` may contain format characters; see format_char() for details. `newtext` is copied using the strdup() function, and can be NULL to make a blank line.
The optional void* argument `d` will be the data() for the new item.

Parameters

in	*newtext*	The label text used for the added item
in	*d*	Optional user data() for the item (0 if unspecified)

See also

add(), insert(), remove(), swap(int,int), clear()

13.13.3.3 bline_length()

```
short Fl_Browser::bline_length (
            const FL_BLINE * b ) const  [protected]
```

Get FL_BLINE allocated text buffer size (excl.
null terminator).

13.13.3.4 bottomline()

```
void Fl_Browser::bottomline (
            int line )  [inline]
```

Scrolls the browser so the bottom item in the browser is showing the specified `line`.

Parameters

in	*line*	The line to be displayed at the bottom.

See also

topline(), middleline(), bottomline(), displayed(), lineposition()

13.13.3.5 clear()

```
void Fl_Browser::clear ( )
```

Removes all the lines in the browser.

See also

add(), insert(), remove(), swap(int,int), clear()

13.13.3.6 column_char() [1/2]

```
char Fl_Browser::column_char ( ) const  [inline]
```

Gets the current column separator character.
The default is '\t' (tab).

See also

column_char(), column_widths()

13.13.3.7 column_char() [2/2]

```
void Fl_Browser::column_char (
            char c )  [inline]
```

Sets the column separator to c.
This will only have an effect if you also set column_widths(). The default is '\t' (tab).

See also

column_char(), column_widths()

13.13.3.8 column_widths() [1/2]

```
const int * Fl_Browser::column_widths ( ) const  [inline]
```

Gets the current column width array.
This array is zero-terminated and specifies the widths in pixels of each column. The text is split at each column_char() and each part is formatted into it's own column. After the last column any remaining text is formatted into the space between the last column and the right edge of the browser, even if the text contains instances of column_char() . The default value is a one-element array of just a zero, which means there are no columns.
Example:

```
Fl_Browser *b = new Fl_Browser(..);
static int widths[] = { 50, 50, 50, 70, 70, 40, 40, 70, 70, 50, 0 };  // widths for each column
b->column_widths(widths); // assign array to widget
b->column_char('\t');     // use tab as the column character
b->add("USER\tPID\tCPU\tMEM\tVSZ\tRSS\tTTY\tSTAT\tSTART\tTIME\tCOMMAND");
b->add("root\t2888\t0.0\t0.0\t1352\t0\ttty3\tSW\tAug15\t0:00\t@b@f/sbin/mingetty tty3");
b->add("root\t13115\t0.0\t0.0\t1352\t0\ttty2\tSW\tAug30\t0:00\t@b@f/sbin/mingetty tty2");
[..]
```

See also

column_char(), column_widths()

13.13.3.9 column_widths() [2/2]

```
void Fl_Browser::column_widths (
            const int * arr )  [inline]
```

Sets the current array to `arr`.
Make sure the last entry is zero.

See also

column_char(), column_widths()

13.13.3.10 data() [1/2]

```
void * Fl_Browser::data (
            int line ) const
```

Returns the user data() for specified `line`.
Return value can be NULL if `line` is out of range or no user data() was defined. The parameter `line` is 1 based (1 will be the first item in the list).

Parameters

in	*line*	The line number of the item whose data() is returned. (1 based)

Returns

The user data pointer (can be NULL)

13.13.3.11 data() [2/2]

```
void Fl_Browser::data (
            int line,
            void * d )
```

Sets the user data for specified `line` to `d`.
Does nothing if `line` is out of range.

Parameters

in	*line*	The line of the item whose data() is to be changed. (1 based)
in	*d*	The new data to be assigned to the item. (can be NULL)

13.13.3.12 display()

```
void Fl_Browser::display (
            int line,
            int val = 1 )
```

For back compatibility.
This calls show(line) if `val` is true, and hide(line) otherwise. If `val` is not specified, the default is 1 (makes the line visible).

See also

show(int), hide(int), display(), visible(), make_visible()

13.13.3.13 displayed()

```
int Fl_Browser::displayed (
            int line ) const  [inline]
```

Returns non-zero if `line` has been scrolled to a position where it is being displayed.
Checks to see if the item's vertical position is within the top and bottom edges of the display window. This does NOT take into account the hide()/show() status of the widget or item.

Parameters

in	*line*	The line to be checked

Returns

1 if visible, 0 if not visible.

See also

topline(), middleline(), bottomline(), displayed(), lineposition()

13.13.3.14 find_line()

```
FL_BLINE * Fl_Browser::find_line (
            int line ) const  [protected]
```

Returns the item for specified `line`.
Note: This call is slow. It's fine for e.g. responding to user clicks, but slow if called often, such as in a tight sorting loop. Finding an item 'by line' involves a linear lookup on the internal linked list. The performance hit can be

significant if the browser's contents is large, and the method is called often (e.g. during a sort). If you're writing a subclass, use the protected methods item_first(), item_next(), etc. to access the internal linked list more efficiently.

Parameters

in	*line*	The line number of the item to return. (1 based)

Return values

item	that was found.
NULL	if line is out of range.

See also

item_at(), find_line(), lineno()

13.13.3.15 format_char() [1/2]

`char Fl_Browser::format_char ( ) const [inline]`

Gets the current format code prefix character, which by default is '@'.
A string of formatting codes at the start of each column are stripped off and used to modify how the rest of the line is printed:

- `'@.'` Print rest of line, don't look for more '@' signs
- `'@@'` Doubling the format character prints the format character once, followed by the rest of line
- `'@l'` Use a LARGE (24 point) font
- `'@m'` Use a medium large (18 point) font
- `'@s'` Use a small (11 point) font
- `'@b'` Use a **bold** font (adds FL_BOLD to font)
- `'@i'` Use an *italic* font (adds FL_ITALIC to font)
- `'@f' or '@t'` Use a fixed-pitch font (sets font to FL_COURIER)
- `'@c'` Center the line horizontally
- `'@r'` Right-justify the text
- `'@N'` Use fl_inactive_color() to draw the text
- `'@B0', '@B1', ... '@B255'` Fill the background with fl_color(n)
- `'@C0', '@C1', ... '@C255'` Use fl_color(n) to draw the text
- `'@F0', '@F1', ...` Use fl_font(n) to draw the text
- `'@S1', '@S2', ...` Use point size n to draw the text
- `'@u' or '@_'` Underline the text.
- `'@-'` draw an engraved line through the middle.

Notice that the '@.' command can be used to reliably terminate the parsing. To print a random string in a random color, use `sprintf("@C%d@.%s", color, string)` and it will work even if the string starts with a digit or has the format character in it.

13.13.3.16 format_char() [2/2]

```
void Fl_Browser::format_char (
            char c ) [inline]
```

Sets the current format code prefix character to `c`.
The default prefix is '@'. Set the prefix to 0 to disable formatting.

See also

format_char() for list of '@' codes

13.13.3.17 full_height()

```
int Fl_Browser::full_height ( ) const [override], [protected], [virtual]
```

The height of the entire list of all visible() items in pixels.
This returns the accumulated height of *all* the items in the browser that are not hidden with hide(), including items scrolled off screen.

Returns

The accumulated size of all the visible items in pixels.

See also

item_height(), item_width(),
incr_height(), full_height()

Reimplemented from Fl_Browser_.

13.13.3.18 hide() [1/2]

```
void Fl_Browser::hide ( ) [inline], [override], [virtual]
```

Hides the entire Fl_Browser widget – opposite of show().
Reimplemented from Fl_Widget.

13.13.3.19 hide() [2/2]

```
void Fl_Browser::hide (
            int line )
```

Makes `line` invisible, preventing selection by the user.
The line can still be selected under program control. This changes the full_height() if the state was changed. When a line is made invisible, lines below it are moved up in the display. redraw() is called automatically if a change occurred.

Parameters

in	*line*	The line to be hidden. (1 based)

See also

show(int), hide(int), display(), visible(), make_visible()

13.13.3.20 icon() [1/2]

```
Fl_Image * Fl_Browser::icon (
            int line ) const
```

Returns the icon currently defined for `line`.
If no icon is defined, NULL is returned.

Parameters

in	*line*	The line whose icon is returned.

Returns

The icon defined, or NULL if none.

13.13.3.21 icon() [2/2]

```
void Fl_Browser::icon (
            int line,
            Fl_Image * icon )
```

Set the image icon for `line` to the value `icon`.
Caller is responsible for keeping the icon allocated. The `line` is automatically redrawn.

Parameters

in	*line*	The line to be modified. If out of range, nothing is done.
in	*icon*	The image icon to be assigned to the `line`. If NULL, any previous icon is removed.

13.13.3.22 incr_height()

```
int Fl_Browser::incr_height ( ) const  [override], [protected], [virtual]
```

The default 'average' item height (including inter-item spacing) in pixels.
This currently returns textsize() + 2.

Returns

The value in pixels.

See also

item_height(), item_width(),
incr_height(), full_height()

Reimplemented from Fl_Browser_.

13.13.3.23 insert() [1/2]

```
void Fl_Browser::insert (
            int line,
            const char * newtext,
            void * d = 0 )
```

Insert a new entry whose label is `newtext` *above* given `line`, optional data `d`.
Text may contain format characters; see format_char() for details. `newtext` is copied using the strdup() function, and can be NULL to make a blank line.
The optional void * argument `d` will be the data() of the new item.

Parameters

in	*line*	Line position for insert. (1 based) If `line` > size(), the entry will be added at the end.
in	*newtext*	The label text for the new line.
in	*d*	Optional pointer to user data to be associated with the new line.

13.13.3.24 insert() [2/2]

```
void Fl_Browser::insert (
            int line,
            FL_BLINE * item )  [protected]
```

Insert specified `item` above `line`.
If `line` > size() then the line is added to the end.
Caveat: See efficiency note in find_line().

Parameters

in	*line*	The new line will be inserted above this line (1 based).
in	*item*	The item to be added.

13.13.3.25 item_at()

```
void * Fl_Browser::item_at (
            int line ) const  [inline], [override], [protected], [virtual]
```

Return the item at specified `line`.

Parameters

in	*line*	The line of the item to return. (1 based)

Returns

The item, or NULL if line out of range.

See also

item_at(), find_line(), lineno()

Reimplemented from Fl_Browser_.

13.13.3.26 item_draw()

```
void Fl_Browser::item_draw (
            void * item,
            int X,
            int Y,
            int W,
            int H ) const  [override], [protected], [virtual]
```

Draws `item` at the position specified by `X Y W H`.
The `W` and `H` values are used for clipping. Should only be called within the context of an FLTK draw().

Parameters

in	*item*	The item to be drawn
in	*X,Y,W,H*	position and size.

Implements Fl_Browser_.

13.13.3.27 item_first()

```
void * Fl_Browser::item_first ( ) const  [override], [protected], [virtual]
```

Returns the very first item in the list.
Example of use:

```
// Walk the browser from beginning to end
```

```
for ( void *i=item_first(); i; i=item_next(i) ) {
    printf("item label='%s'\n", item_text(i));
}
```

Returns

The first item, or NULL if list is empty.

See also

item_first(), item_last(), item_next(), item_prev()

Implements Fl_Browser_.

13.13.3.28 item_height()

```
int Fl_Browser::item_height (
            void * item ) const  [override], [protected], [virtual]
```

Returns height of `item` in pixels.
This takes into account embedded @ codes within the text() label.

Parameters

in	*item*	The item whose height is returned.

Returns

The height of the item in pixels.

See also

item_height(), item_width(),
incr_height(), full_height()

Implements Fl_Browser_.

13.13.3.29 item_last()

```
void * Fl_Browser::item_last ( ) const  [override], [protected], [virtual]
```

Returns the very last item in the list.
Example of use:

```
// Walk the browser in reverse, from end to start
for ( void *i=item_last(); i; i=item_prev(i) ) {
    printf("item label='%s'\n", item_text(i));
}
```

Returns

The last item, or NULL if list is empty.

See also

item_first(), item_last(), item_next(), item_prev()

Reimplemented from Fl_Browser_.

13.13.3.30 item_next()

```
void * Fl_Browser::item_next (
            void * item ) const  [override], [protected], [virtual]
```

Returns the next item after `item`.

Parameters

in	*item*	The 'current' item

Returns

The next item after `item`, or NULL if there are none after this one.

See also

item_first(), item_last(), item_next(), item_prev()

Implements Fl_Browser_.

13.13.3.31 item_prev()

```
void * Fl_Browser::item_prev (
            void * item ) const  [override], [protected], [virtual]
```

Returns the previous item before `item`.

Parameters

in	*item*	The 'current' item

Returns

The previous item before `item`, or NULL if there are none before this one.

See also

item_first(), item_last(), item_next(), item_prev()

Implements Fl_Browser_.

13.13.3.32 item_select()

```
void Fl_Browser::item_select (
            void * item,
            int val )  [override], [protected], [virtual]
```

Change the selection state of `item` to the value `val`.

Parameters

in	*item*	The item to be changed.
in	*val*	The new selection state: 1 selects, 0 de-selects.

See also

select(), selected(), value(), item_select(), item_selected()

Reimplemented from Fl_Browser_.

13.13.3.33 item_selected()

```
int Fl_Browser::item_selected (
            void * item ) const  [override], [protected], [virtual]
```

See if `item` is selected.

Parameters

in	*item*	The item whose selection state is to be checked.

Returns

1 if selected, 0 if not.

See also

select(), selected(), value(), item_select(), item_selected()

Reimplemented from Fl_Browser_.

13.13.3.34 item_swap()

```
void Fl_Browser::item_swap (
            void * a,
            void * b )  [inline], [override], [protected], [virtual]
```

Swap the items `a` and `b`.
You must call redraw() to make any changes visible.

Parameters

in	*a,b*	the items to be swapped.

See also

swap(int,int), item_swap()

Reimplemented from Fl_Browser_.

13.13.3.35 item_text()

```
const char * Fl_Browser::item_text (
            void * item ) const  [override], [protected], [virtual]
```

Returns the label text for `item`.

Parameters

in	*item*	The item whose label text is returned.

Returns

The item's text string. (Can be NULL)

Reimplemented from Fl_Browser_.

13.13.3.36 item_width()

```
int Fl_Browser::item_width (
            void * item ) const  [override], [protected], [virtual]
```

Returns width of `item` in pixels.
This takes into account embedded @ codes within the text() label.

Parameters

in	*item*	The item whose width is returned.

Returns

The width of the item in pixels.

See also

item_height(), item_width(),
incr_height(), full_height()

Implements Fl_Browser_.

13.13.3.37 lineno()

```
int Fl_Browser::lineno (
            void * item ) const  [protected]
```

Returns line number corresponding to `item`, or zero if not found.
Caveat: See efficiency note in find_line().

Parameters

in	*item*	The item to be found

Returns

The line number of the item, or 0 if not found.

See also

item_at(), find_line(), lineno()

13.13.3.38 lineposition()

```
void Fl_Browser::lineposition (
            int line,
            Fl_Line_Position pos )
```

Updates the browser so that `line` is shown at position `pos`.

Parameters

in	*line*	line number. (1 based)
in	*pos*	position.

See also

topline(), middleline(), bottomline()

13.13.3.39 load()

```
int Fl_Browser::load (
            const char * filename )
```

Clears the browser and reads the file, adding each line from the file to the browser.
If the filename is NULL or a zero-length string then this just clears the browser. This returns zero if there was any error in opening or reading the file, in which case errno is set to the system error. The data() of each line is set to NULL.

Parameters

in	*filename*	The filename to load

Returns

1 if OK, 0 on error (errno has reason)

See also

add()

13.13.3.40 make_visible()

```
void Fl_Browser::make_visible (
            int line )  [inline]
```

Make the item at the specified `line` visible().
Functionally similar to show(int line). If `line` is out of range, redisplay top or bottom of list as appropriate.

Parameters

in	*line*	The line to be made visible.

See also

show(int), hide(int), display(), visible(), make_visible()

13.13.3.41 middleline()

```
void Fl_Browser::middleline (
            int line )  [inline]
```

Scrolls the browser so the middle item in the browser is showing the specified `line`.

Parameters

in	*line*	The line to be displayed in the middle.

See also

topline(), middleline(), bottomline(), displayed(), lineposition()

13.13.3.42 move()

```
void Fl_Browser::move (
            int to,
            int from )
```

Line `from` is removed and reinserted at `to`.
Note: `to` is calculated *after* line `from` gets removed.

Parameters

in	*to*	Destination line number (calculated *after* line `from` is removed)
in	*from*	Line number of item to be moved

13.13.3.43 remove()

```
void Fl_Browser::remove (
            int line )
```

Remove entry for given `line` number, making the browser one line shorter.

You must call redraw() to make any changes visible.

Parameters

in	*line*	Line to be removed. (1 based) If `line` is out of range, no action is taken.

See also

add(), insert(), remove(), swap(int,int), clear()

13.13.3.44 remove_icon()

```
void Fl_Browser::remove_icon (
            int line )
```

Removes the icon for `line`.
It's ok to remove an icon if none has been defined.

Parameters

in	*line*	The line whose icon is to be removed.

13.13.3.45 select()

```
int Fl_Browser::select (
            int line,
            int val = 1 )
```

Sets the selection state of the item at `line` to the value `val`.
If `val` is not specified, the default is 1 (selects the item).

Parameters

in	*line*	The line number of the item to be changed. (1 based)
in	*val*	The new selection state (1=select, 0=de-select).

Returns

1 if the state changed, 0 if not.

See also

select(), selected(), value(), item_select(), item_selected()

13.13.3.46 selected()

```
int Fl_Browser::selected (
            int line ) const
```

Returns 1 if specified `line` is selected, 0 if not.

Parameters

in	*line*	The line being checked (1 based)

Returns

1 if item selected, 0 if not.

See also

select(), selected(), value(), item_select(), item_selected()

13.13.3.47 show() [1/2]

```
void Fl_Browser::show ( ) [inline], [override], [virtual]
```

Shows the entire Fl_Browser widget – opposite of hide().
Reimplemented from Fl_Widget.

13.13.3.48 show() [2/2]

```
void Fl_Browser::show (
            int line )
```

Makes `line` visible, and available for selection by user.
Opposite of hide(int). This changes the full_height() if the state was changed. redraw() is called automatically if a change occurred.

Parameters

in	*line*	The line to be shown. (1 based)

See also

show(int), hide(int), display(), visible(), make_visible()

13.13.3.49 size()

```
int Fl_Browser::size ( ) const [inline]
```

Returns how many lines are in the browser.
The last line number is equal to this. Returns 0 if browser is empty.

13.13.3.50 swap() [1/2]

```
void Fl_Browser::swap (
            FL_BLINE * a,
            FL_BLINE * b ) [protected]
```

Swap the two items `a` and `b`.
Uses swapping() to ensure list updates correctly.

Parameters

in	*a,b*	The two items to be swapped.

See also

swap(int,int), item_swap()

13.13.3.51 swap() [2/2]

```
void Fl_Browser::swap (
            int a,
            int b )
```

Swaps two browser lines `a` and `b`.
You must call redraw() to make any changes visible.

Parameters

in	*a,b*	The two lines to be swapped. (both 1 based)

See also

swap(int,int), item_swap()

13.13.3.52 text() [1/2]

```
const char * Fl_Browser::text (
            int line ) const
```

Returns the label text for the specified `line`.
Return value can be NULL if `line` is out of range or unset. The parameter `line` is 1 based.

Parameters

in	*line*	The line number of the item whose text is returned. (1 based)

Returns

The text string (can be NULL)

13.13.3.53 text() [2/2]

```
void Fl_Browser::text (
            int line,
            const char * newtext )
```

Sets the text for the specified `line` to `newtext`.
Text may contain format characters; see format_char() for details. `newtext` is copied using the strdup() function, and can be NULL to make a blank line.
Does nothing if `line` is out of range.

Parameters

in	*line*	The line of the item whose text will be changed. (1 based)
in	*newtext*	The new string to be assigned to the item.

13.13.3.54 textsize()

```
void Fl_Browser::textsize (
            Fl_Fontsize newSize )
```

Sets the default text size (in pixels) for the lines in the browser to `newSize`.
This method recalculates all item heights and caches the total height internally for optimization of later item changes.
This can be slow if there are many items in the browser.
It returns immediately (w/o recalculation) if `newSize` equals the current textsize().
You *may* need to call redraw() to see the effect and to have the scrollbar positions recalculated.
You should set the text size *before* populating the browser with items unless you really need to *change* the size later.

13.13.3.55 topline() [1/2]

```
int Fl_Browser::topline ( ) const
```

Returns the line that is currently visible at the top of the browser.
If there is no vertical scrollbar then this will always return 1.

Returns

The lineno() of the top() of the browser.

13.13.3.56 topline() [2/2]

```
void Fl_Browser::topline (
            int line )  [inline]
```

Scrolls the browser so the top item in the browser is showing the specified `line`.

Parameters

in	*line*	The line to be displayed at the top.

See also

topline(), middleline(), bottomline(), displayed(), lineposition()

13.13.3.57 value() [1/2]

```
int Fl_Browser::value ( ) const
```

Returns the line number of the currently selected line, or 0 if none selected.

Returns

The line number of current selection, or 0 if none selected.

See also

select(), selected(), value(), item_select(), item_selected()

13.13.3.58 value() [2/2]

```
void Fl_Browser::value (
            int line )  [inline]
```

Sets the browser's value(), which selects the specified `line`.
This is the same as calling select(line).

See also

select(), selected(), value(), item_select(), item_selected()

13.13.3.59 visible()

```
int Fl_Browser::visible (
            int line ) const
```

Returns non-zero if the specified `line` is visible, 0 if hidden.
Use show(int), hide(int), or make_visible(int) to change an item's visible state.

Parameters

in	*line*	The line in the browser to be tested. (1 based)

See also

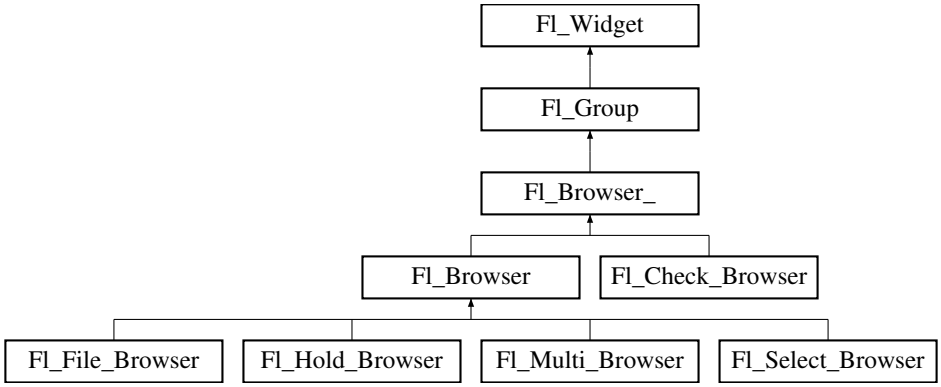
show(int), hide(int), display(), visible(), make_visible()

The documentation for this class was generated from the following files:

- Fl_Browser.H
- Fl_Browser.cxx
- Fl_Browser_load.cxx

13.14 Fl_Browser_ Class Reference

This is the base class for browsers.
`#include <Fl_Browser_.H>`
Inheritance diagram for Fl_Browser_:



Public Types

- enum {
  HORIZONTAL = 1 , VERTICAL = 2 , BOTH = 3 , ALWAYS_ON = 4 ,
  HORIZONTAL_ALWAYS = 5 , VERTICAL_ALWAYS = 6 , BOTH_ALWAYS = 7 }

  Values for has_scrollbar().

Public Member Functions

- int deselect (int docallbacks=0)

  Deselects all items in the list and returns 1 if the state changed or 0 if it did not.
- void display (void *item)

  Displays the `item`, scrolling the list as necessary.
- int handle (int event) override

  Handles the `event` within the normal widget bounding box.
- uchar **has_scrollbar** () const

  Returns the current scrollbar mode, see Fl_Browser_::has_scrollbar(uchar)
- void has_scrollbar (uchar mode)

  Sets whether the widget should have scrollbars or not (default Fl_Browser_::BOTH).
- int hposition () const

  Gets the horizontal scroll position of the list as a pixel position `pos`.
- void hposition (int)

  Sets the horizontal scroll position of the list to pixel position `pos`.
- int linespacing () const

  Return the height of additional spacing between browser lines.
- void linespacing (int pixels)

Add some space between browser lines.

- int position () const
- void position (int pos)
- void **position** (int x, int y)
- void resize (int X, int Y, int W, int H) override

  Repositions and/or resizes the browser.
- void scrollbar_left ()

  Moves the vertical scrollbar to the lefthand side of the list.
- void scrollbar_right ()

  Moves the vertical scrollbar to the righthand side of the list.
- int scrollbar_size () const

  Gets the current size of the scrollbars' troughs, in pixels.
- void scrollbar_size (int newSize)

  Sets the pixel size of the scrollbars' troughs to `newSize`, in pixels.
- int select (void *item, int val=1, int docallbacks=0)

  Sets the selection state of `item` to `val`, and returns 1 if the state changed or 0 if it did not.
- int select_only (void *item, int docallbacks=0)

  Selects `item` and returns 1 if the state changed or 0 if it did not.
- void sort (int flags=0)

  Sort the items in the browser based on `flags`.
- Fl_Color **textcolor** () const

  Gets the default text color for the lines in the browser.
- void **textcolor** (Fl_Color col)

  Sets the default text color for the lines in the browser to color `col`.
- Fl_Font textfont () const

  Gets the default text font for the lines in the browser.
- void **textfont** (Fl_Font font)

  Sets the default text font for the lines in the browser to `font`.
- Fl_Fontsize **textsize** () const

  Gets the default text size (in pixels) for the lines in the browser.
- void **textsize** (Fl_Fontsize newSize)

  Sets the default text size (in pixels) for the lines in the browser to `size`.
- int vposition () const

  Gets the vertical scroll position of the list as a pixel position `pos`.
- void vposition (int pos)

  Sets the vertical scroll position of the list to pixel position `pos`.

Public Member Functions inherited from Fl_Group

- Fl_Widget *& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget *o)

  See void Fl_Group::add(Fl_Widget &w)
- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget *const * array () const

  Returns a pointer to the internal array of children.
- Fl_Group const * as_group () const override
- Fl_Group * as_group () override

Returns an Fl_Group pointer if this widget is an Fl_Group.

- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const

  Returns the n'th child.
- int **children** () const

  Returns how many child widgets the group has.
- void clear ()

  Deletes all child widgets from memory recursively.
- unsigned int clip_children () const

  Returns the current clipping mode.
- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)

  Removes the widget at `index` from the group and deletes it.
- void end ()

  Exactly the same as current(this->parent()).
- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*
- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.
- void **forms_end** ()

  This is for forms compatibility only.
- int handle (int) override

  Handles the specified event.
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

  This does insert(w, find(before)).
- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.
- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.
- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- void resize (int, int, int, int) override

  Resizes the Fl_Group widget and all of its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ **as_gl_window** () const
- virtual Fl_Window ∗ as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ **as_window** () const
- void bind_deimage (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗cb, void ∗p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

Marks the widget as inactive without sending events or changing focus.

- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

Returns whether this widget needs a keyboard.

- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.

- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Public Attributes

- Fl_Scrollbar hscrollbar

  Horizontal scrollbar.
- Fl_Scrollbar scrollbar

  Vertical scrollbar.

Protected Member Functions

- void bbox (int &X, int &Y, int &W, int &H) const

  Returns the bounding box for the interior of the list's display window, inside the scrollbars.
- void deleting (void *item)

  This method should be used when `item` is being deleted from the list.
- int displayed (void *item) const

  Returns non-zero if `item` has been scrolled to a position where it is being displayed.
- void draw () override

  Draws the list within the normal widget bounding box.
- void * find_item (int ypos)

  This method returns the item under mouse y position `ypos`.
- Fl_Browser_ (int X, int Y, int W, int H, const char *L=0)

  The constructor makes an empty browser.
- virtual int full_height () const

  This method may be provided by the subclass to indicate the full height of the item list, in pixels.
- virtual int full_width () const

  This method may be provided by the subclass to indicate the full width of the item list, in pixels.
- virtual int incr_height () const

  This method may be provided to return the average height of all items to be used for scrolling.
- void inserting (void *a, void *b)

  This method should be used when an item is in the process of being inserted into the list.
- virtual void * item_at (int index) const

  This method must be provided by the subclass to return the item for the specified `index`.
- virtual void item_draw (void *item, int X, int Y, int W, int H) const =0

  This method must be provided by the subclass to draw the `item` in the area indicated by `X`, `Y`, `W`, `H`.
- virtual void * item_first () const =0

  This method must be provided by the subclass to return the first item in the list.
- virtual int item_height (void *item) const =0

  This method must be provided by the subclass to return the height of `item` in pixels.
- virtual void * item_last () const

  This method must be provided by the subclass to return the last item in the list.
- virtual void * item_next (void *item) const =0

  This method must be provided by the subclass to return the item in the list after `item`.
- virtual void * item_prev (void *item) const =0

  This method must be provided by the subclass to return the item in the list before `item`.
- virtual int item_quick_height (void *item) const

  This method may be provided by the subclass to return the height of the `item`, in pixels.
- virtual void item_select (void *item, int val=1)

  This method must be implemented by the subclass if it supports multiple selections; sets the selection state to `val` for the `item`.
- virtual int item_selected (void *item) const

  This method must be implemented by the subclass if it supports multiple selections; returns the selection state for `item`.
- virtual void item_swap (void *a, void *b)

  This optional method should be provided by the subclass to efficiently swap browser items `a` and `b`, such as for sorting.
- virtual const char * item_text (void *item) const

  This optional method returns a string (label) that may be used for sorting.
- virtual int item_width (void *item) const =0

  This method must be provided by the subclass to return the width of the `item` in pixels.
- int leftedge () const

This method returns the X position of the left edge of the list area after adjusting for the scrollbar and border, if any.

- void new_list ()

  This method should be called when the list data is completely replaced or cleared.
- void redraw_line (void *item)

  This method should be called when the contents of `item` has changed, but not its height.
- void redraw_lines ()

  This method will cause the entire list to be redrawn.
- void replacing (void *a, void *b)

  This method should be used when item `a` is being replaced by item `b`.
- void * selection () const

  Returns the item currently selected, or NULL if there is no selection.
- void swapping (void *a, void *b)

  This method should be used when two items `a` and `b` are being swapped.
- void * **top** () const

  Returns the item that appears at the top of the list.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()

  Returns the internal array of widget sizes and positions.
- void draw () override

  Draws the widget.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget *, int)

  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)

  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)

  Allow derived groups to act when a child widget is removed from the group.
- int * sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

Draws a focus rectangle around the widget.

- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()

  Returns the currently active group.
- static void current (Fl_Group *g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,

NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 , AUTO_DELETE_USER_DATA = 1<<23 ,
MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
USERFLAG1 = 1<<31 }

flags possible values enumeration.

13.14.1 Detailed Description

This is the base class for browsers.
To be useful it must be subclassed and several virtual functions defined. The Forms-compatible browser and the file chooser's browser are subclassed off of this.
This has been designed so that the subclass has complete control over the storage of the data, although because next() and prev() functions are used to index, it works best as a linked list or as a large block of characters in which the line breaks must be searched for.
A great deal of work has been done so that the "height" of a data object does not need to be determined until it is drawn. This is useful if actually figuring out the size of an object requires accessing image data or doing stat() on a file or doing some other slow operation.
Callbacks are called when the value changes with `FL_REASON_CHANGED`. If `FL_WHEN_RELEASE` is set, callbacks are called when the mouse button is released with `FL_REASON_CHANGED` or `FL_REASON_RESELECTED` if the selection did not change. If `FL_WHEN_ENTER_KEY` is set, callbacks are also called when key presses or double clicks change the selection.

Keyboard navigation of browser items

The keyboard navigation of browser items is only possible if visible_focus() is enabled. If disabled, the widget rejects keyboard focus; Tab and Shift-Tab focus navigation will skip the widget.
In 'Select' and 'Normal' mode, the widget rejects keyboard focus; no navigation keys are supported (other than scrollbar positioning).
In 'Hold' mode, the widget accepts keyboard focus, and Up/Down arrow keys can navigate the selected item.
In 'Multi' mode, the widget accepts keyboard focus, and Up/Down arrow keys navigate the focus box; Space toggles the current item's selection, Enter selects only the current item (deselects all others). If Shift (or Ctrl) is combined with Up/Down arrow keys, the current item's selection state is extended to the next item. In this way one can extend a selection or de-selection.

13.14.2 Member Enumeration Documentation

13.14.2.1 anonymous enum

`anonymous enum`
Values for has_scrollbar().
Anonymous enum bit flags for has_scrollbar().

- bit 0: horizontal
- bit 1: vertical
- bit 2: 'always' (to be combined with bits 0 and 1)
- bit 3-31: reserved for future use

Enumerator

HORIZONTAL	Only show horizontal scrollbar.
VERTICAL	Only show vertical scrollbar.
BOTH	Show both scrollbars. (default)
ALWAYS_ON	Specified scrollbar(s) should 'always' be shown (to be used with HORIZONTAL/VERTICAL)
HORIZONTAL_ALWAYS	Horizontal scrollbar always on.
VERTICAL_ALWAYS	Vertical scrollbar always on.
BOTH_ALWAYS	Both scrollbars always on.

13.14.3 Constructor & Destructor Documentation

13.14.3.1 Fl_Browser_()

```
Fl_Browser_::Fl_Browser_ (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )  [protected]
```

The constructor makes an empty browser.

Parameters

in	*X,Y,W,H*	position and size.
in	*L*	The label string, may be NULL.

13.14.4 Member Function Documentation

13.14.4.1 bbox()

```
void Fl_Browser_::bbox (
            int & X,
            int & Y,
            int & W,
            int & H ) const  [protected]
```

Returns the bounding box for the interior of the list's display window, inside the scrollbars.

Parameters

out	*X,Y,W,H*	The returned bounding box. (The original contents of these parameters are overwritten)

13.14.4.2 deleting()

```
void Fl_Browser_::deleting (
            void * item )  [protected]
```

This method should be used when `item` is being deleted from the list.
It allows the Fl_Browser_ to discard any cached data it has on the item. This method does not actually delete the item, but handles the follow up bookkeeping after the item has just been deleted.

Parameters

in	*item*	The item being deleted.

13.14.4.3 deselect()

```
int Fl_Browser_::deselect (
            int docallbacks = 0 )
```

Deselects all items in the list and returns 1 if the state changed or 0 if it did not.
If the optional `docallbacks` parameter is non-zero, deselect tries to call the callback function for the widget.

Parameters

in	*docallbacks*	If non-zero, invokes widget callback if item changed. If 0, doesn't do callback (default).

13.14.4.4 display()

```
void Fl_Browser_::display (
            void * item )
```

Displays the `item`, scrolling the list as necessary.

Parameters

in	*item*	The item to be displayed.

See also

display(), displayed()

13.14.4.5 displayed()

```
int Fl_Browser_::displayed (
            void * item ) const [protected]
```

Returns non-zero if `item` has been scrolled to a position where it is being displayed.
Checks to see if the item's vertical position is within the top and bottom edges of the display window. This does NOT take into account the hide()/show() status of the widget or item.

Parameters

in	*item*	The item to check

Returns

1 if visible, 0 if not visible.

See also

display(), displayed()

13.14.4.6 draw()

```
void Fl_Browser_::draw (
            void ) [override], [protected], [virtual]
```

Draws the list within the normal widget bounding box.
Implements Fl_Widget.

13.14.4.7 find_item()

```
void * Fl_Browser_::find_item (
            int ypos ) [protected]
```

This method returns the item under mouse y position `ypos`.
NULL is returned if no item is displayed at that position.

Parameters

in	*ypos*	The y position (eg. Fl::event_y()) to find an item under.

Returns

The item, or NULL if not found

13.14.4.8 full_height()

```
int Fl_Browser_::full_height ( ) const  [protected], [virtual]
```

This method may be provided by the subclass to indicate the full height of the item list, in pixels.
The default implementation computes the full height from the item heights. Includes the items that are scrolled off screen.

Returns

The height of the entire list, in pixels.

Reimplemented in Fl_Browser.

13.14.4.9 full_width()

```
int Fl_Browser_::full_width ( ) const  [protected], [virtual]
```

This method may be provided by the subclass to indicate the full width of the item list, in pixels.
The default implementation computes the full width from the item widths.

Returns

The maximum width of all the items, in pixels.

13.14.4.10 handle()

```
int Fl_Browser_::handle (
            int event )  [override], [virtual]
```

Handles the `event` within the normal widget bounding box.

Parameters

in	*event*	The event to process.

Returns

1 if event was processed, 0 if not.

Reimplemented from Fl_Widget.
Reimplemented in Fl_Check_Browser.

13.14.4.11 has_scrollbar()

```
void Fl_Browser_::has_scrollbar (
            uchar mode )  [inline]
```

Sets whether the widget should have scrollbars or not (default Fl_Browser_::BOTH).
By default you can scroll in both directions, and the scrollbars disappear if the data will fit in the widget. has_scrollbar() changes this based on the value of `mode`:

- 0 - No scrollbars.
- Fl_Browser_::HORIZONTAL - Only a horizontal scrollbar.
- Fl_Browser_::VERTICAL - Only a vertical scrollbar.
- Fl_Browser_::BOTH - The default is both scrollbars.
- Fl_Browser_::HORIZONTAL_ALWAYS - Horizontal scrollbar always on, vertical always off.
- Fl_Browser_::VERTICAL_ALWAYS - Vertical scrollbar always on, horizontal always off.
- Fl_Browser_::BOTH_ALWAYS - Both always on.

13.14.4.12 hposition() [1/2]

```
int Fl_Browser_::hposition ( ) const  [inline]
```

Gets the horizontal scroll position of the list as a pixel position `pos`.
The position returned is how many pixels of the list are scrolled off the left edge of the screen. Example: A position of '18' indicates the left 18 pixels of the list are scrolled off the left edge of the screen.

See also

position(), hposition()

13.14.4.13 hposition() [2/2]

```
void Fl_Browser_::hposition (
            int pos )
```

Sets the horizontal scroll position of the list to pixel position `pos`.
The position is how many pixels of the list are scrolled off the left edge of the screen. Example: A position of '18' scrolls the left 18 pixels of the list off the left edge of the screen.

Parameters

in	*pos*	The horizontal position (in pixels) to scroll the browser to.

See also

vposition(), hposition()

13.14.4.14 incr_height()

```
int Fl_Browser_::incr_height ( ) const  [protected], [virtual]
```

This method may be provided to return the average height of all items to be used for scrolling.
The default implementation uses the height of the first item.

Returns

The average height of items, in pixels.

Reimplemented in Fl_Browser.

13.14.4.15 inserting()

```
void Fl_Browser_::inserting (
            void * a,
            void * b )  [protected]
```

This method should be used when an item is in the process of being inserted into the list.
It allows the Fl_Browser_ to update its cache data as needed, scheduling a redraw for the affected lines. This method does not actually insert items, but handles the follow up bookkeeping after items have been inserted.

Parameters

in	*a*	The starting item position
in	*b*	The new item being inserted

13.14.4.16 item_at()

```
virtual void * Fl_Browser_::item_at (
            int index ) const  [inline], [protected], [virtual]
```

This method must be provided by the subclass to return the item for the specified `index`.

Parameters

in	*index*	The `index` of the item to be returned

Returns

The item at the specified `index`.

Reimplemented in Fl_Check_Browser, and Fl_Browser.

13.14.4.17 item_draw()

```
virtual void Fl_Browser_::item_draw (
            void * item,
            int X,
            int Y,
            int W,
            int H ) const [protected], [pure virtual]
```

This method must be provided by the subclass to draw the `item` in the area indicated by `X`, `Y`, `W`, `H`.
Implemented in Fl_Check_Browser, and Fl_Browser.

13.14.4.18 item_first()

```
virtual void * Fl_Browser_::item_first ( ) const [protected], [pure virtual]
```

This method must be provided by the subclass to return the first item in the list.

See also

item_first(), item_next(), item_last(), item_prev()

Implemented in Fl_Browser, and Fl_Check_Browser.

13.14.4.19 item_height()

```
virtual int Fl_Browser_::item_height (
            void * item ) const [protected], [pure virtual]
```

This method must be provided by the subclass to return the height of `item` in pixels.
Allow for two additional pixels for the list selection box.

Parameters

in	*item*	The item whose height is returned.

Returns

The height of the specified `item` in pixels.

See also

item_height(), item_width(), item_quick_height()

Implemented in Fl_Check_Browser, and Fl_Browser.

13.14.4.20 item_last()

```
virtual void * Fl_Browser_::item_last ( ) const [inline], [protected], [virtual]
```

This method must be provided by the subclass to return the last item in the list.

See also

item_first(), item_next(), item_last(), item_prev()

Reimplemented in Fl_Browser.

13.14.4.21 item_next()

```
virtual void * Fl_Browser_::item_next (
            void * item ) const  [protected], [pure virtual]
```

This method must be provided by the subclass to return the item in the list after `item`.

See also

item_first(), item_next(), item_last(), item_prev()

Implemented in Fl_Check_Browser, and Fl_Browser.

13.14.4.22 item_prev()

```
virtual void * Fl_Browser_::item_prev (
            void * item ) const  [protected], [pure virtual]
```

This method must be provided by the subclass to return the item in the list before `item`.

See also

item_first(), item_next(), item_last(), item_prev()

Implemented in Fl_Check_Browser, and Fl_Browser.

13.14.4.23 item_quick_height()

```
int Fl_Browser_::item_quick_height (
            void * item ) const  [protected], [virtual]
```

This method may be provided by the subclass to return the height of the `item`, in pixels.
Allow for two additional pixels for the list selection box. This method differs from item_height in that it is only called for selection and scrolling operations. The default implementation calls item_height.

Parameters

in	*item*	The item whose height to return.

Returns

The height, in pixels.

13.14.4.24 item_select()

```
void Fl_Browser_::item_select (
            void * item,
            int val = 1 )  [protected], [virtual]
```

This method must be implemented by the subclass if it supports multiple selections; sets the selection state to `val` for the `item`.
Sets the selection state for `item`, where optional `val` is 1 (select, the default) or 0 (de-select).

Parameters

in	*item*	The item to be selected
in	*val*	The optional selection state; 1=select, 0=de-select. The default is to select the item (1).

Reimplemented in Fl_Check_Browser, and Fl_Browser.

13.14.4.25 item_selected()

```
int Fl_Browser_::item_selected (
```

```
            void * item ) const  [protected], [virtual]
```

This method must be implemented by the subclass if it supports multiple selections; returns the selection state for `item`.
The method should return 1 if `item` is selected, or 0 otherwise.

Parameters

in	*item*	The item to test.

Reimplemented in Fl_Check_Browser, and Fl_Browser.

13.14.4.26 item_swap()

```
virtual void Fl_Browser_::item_swap (
            void * a,
            void * b )  [inline], [protected], [virtual]
```

This optional method should be provided by the subclass to efficiently swap browser items `a` and `b`, such as for sorting.

Parameters

in	*a,b*	The two items to be swapped.

Reimplemented in Fl_Browser, and Fl_Check_Browser.

13.14.4.27 item_text()

```
virtual const char * Fl_Browser_::item_text (
            void * item ) const  [inline], [protected], [virtual]
```

This optional method returns a string (label) that may be used for sorting.

Parameters

in	*item*	The item whose label text is returned.

Returns

The item's text label. (Can be NULL if blank)

Reimplemented in Fl_Browser, and Fl_Check_Browser.

13.14.4.28 item_width()

```
virtual int Fl_Browser_::item_width (
            void * item ) const  [protected], [pure virtual]
```

This method must be provided by the subclass to return the width of the `item` in pixels.
Allow for two additional pixels for the list selection box.

Parameters

in	*item*	The item whose width is returned.

Returns

The width of the item in pixels.

Implemented in Fl_Check_Browser, and Fl_Browser.

13.14.4.29 leftedge()

```
int Fl_Browser_::leftedge ( ) const [protected]
```

This method returns the X position of the left edge of the list area after adjusting for the scrollbar and border, if any.

Returns

The X position of the left edge of the list, in pixels.

See also

Fl_Browser_::bbox()

13.14.4.30 linespacing() [1/2]

```
int Fl_Browser_::linespacing ( ) const [inline]
```

Return the height of additional spacing between browser lines.

Returns

spacing height in pixel units.

13.14.4.31 linespacing() [2/2]

```
void Fl_Browser_::linespacing (
            int pixels ) [inline]
```

Add some space between browser lines.

Parameters

in	*pixels*	number of additional pixels between lines.

13.14.4.32 new_list()

```
void Fl_Browser_::new_list ( ) [protected]
```

This method should be called when the list data is completely replaced or cleared.
It informs the Fl_Browser_ widget that any cached information it has concerning the items is invalid. This method does not clear the list, it just handles the follow up bookkeeping after the list has been cleared.

13.14.4.33 position() [1/2]

```
int Fl_Browser_::position ( ) const [inline]
```

Deprecated "since 1.4.0 - use vposition() instead"

13.14.4.34 position() [2/2]

```
void Fl_Browser_::position (
            int pos ) [inline]
```

Deprecated "since 1.4.0 - use vposition(pos) instead"

13.14.4.35 redraw_line()

```
void Fl_Browser_::redraw_line (
            void * item )  [protected]
```

This method should be called when the contents of `item` has changed, but not its height.

Parameters

in	*item*	The item that needs to be redrawn.

See also

redraw_lines(), redraw_line()

13.14.4.36 redraw_lines()

```
void Fl_Browser_::redraw_lines ( )  [inline], [protected]
```

This method will cause the entire list to be redrawn.

See also

redraw_lines(), redraw_line()

13.14.4.37 replacing()

```
void Fl_Browser_::replacing (
            void * a,
            void * b )  [protected]
```

This method should be used when item `a` is being replaced by item `b`.
It allows the Fl_Browser_ to update its cache data as needed, schedules a redraw for the item being changed, and tries to maintain the selection. This method does not actually replace the item, but handles the follow up bookkeeping after the item has just been replaced.

Parameters

in	*a*	Item being replaced
in	*b*	Item to replace 'a'

13.14.4.38 resize()

```
void Fl_Browser_::resize (
            int X,
            int Y,
            int W,
            int H )  [override], [virtual]
```

Repositions and/or resizes the browser.

Parameters

in	*X,Y,W,H*	The new position and size for the browser, in pixels.

Reimplemented from Fl_Widget.

13.14.4.39 scrollbar_left()

```
void Fl_Browser_::scrollbar_left ( )  [inline]
```

Moves the vertical scrollbar to the lefthand side of the list.

For back compatibility.

13.14.4.40 scrollbar_right()

```
void Fl_Browser_::scrollbar_right ( ) [inline]
```

Moves the vertical scrollbar to the righthand side of the list.
For back compatibility.

13.14.4.41 scrollbar_size() [1/2]

```
int Fl_Browser_::scrollbar_size ( ) const [inline]
```

Gets the current size of the scrollbars' troughs, in pixels.
If this value is zero (default), this widget will use the Fl::scrollbar_size() value as the scrollbar's width.

Returns

Scrollbar size in pixels, or 0 if the global Fl::scrollbar_size() is being used.

See also

Fl::scrollbar_size(int)

13.14.4.42 scrollbar_size() [2/2]

```
void Fl_Browser_::scrollbar_size (
            int newSize ) [inline]
```

Sets the pixel size of the scrollbars' troughs to `newSize`, in pixels.
Normally you should not need this method, and should use Fl::scrollbar_size(int) instead to manage the size of ALL your widgets' scrollbars. This ensures your application has a consistent UI, is the default behavior, and is normally what you want.
Only use **this** method if you really need to override the global scrollbar size. The need for this should be rare.
Setting `newSize` to the special value of 0 causes the widget to track the global Fl::scrollbar_size(), which is the default.

Parameters

in	*newSize*	Sets the scrollbar size in pixels. If 0 (default), scrollbar size tracks the global Fl::scrollbar_size()

See also

Fl::scrollbar_size()

13.14.4.43 select()

```
int Fl_Browser_::select (
            void * item,
            int val = 1,
            int docallbacks = 0 )
```

Sets the selection state of `item` to `val`, and returns 1 if the state changed or 0 if it did not.
If `docallbacks` is non-zero, select tries to call the callback function for the widget.

Parameters

in	*item*	The item whose selection state is to be changed
in	*val*	The new selection state (1=select, 0=de-select)
in	*docallbacks*	If non-zero, invokes widget callback if item changed. If 0, doesn't do callback (default).

Returns

1 if state was changed, 0 if not.

13.14.4.44 select_only()

```
int Fl_Browser_::select_only (
            void * item,
            int docallbacks = 0 )
```

Selects `item` and returns 1 if the state changed or 0 if it did not.
Any other items in the list are deselected.

Parameters

in	*item*	The `item` to select.
in	*docallbacks*	If non-zero, invokes widget callback if item changed. If 0, doesn't do callback (default).

13.14.4.45 selection()

```
void * Fl_Browser_::selection ( ) const  [inline], [protected]
```

Returns the item currently selected, or NULL if there is no selection.
For multiple selection browsers this call returns the currently focused item, even if it is not selected. To find all selected items, call Fl_Multi_Browser::selected() for every item in question.

13.14.4.46 sort()

```
void Fl_Browser_::sort (
            int flags = 0 )
```

Sort the items in the browser based on `flags`.
item_swap(void∗, void∗) and item_text(void∗) must be implemented for this call.

Parameters

in	*flags*	FL_SORT_ASCENDING – sort in ascending order FL_SORT_DESCENDING – sort in descending order FL_SORT_CASEINSENSITIVE – add this to sort case-insensitively Values other than the above will cause undefined behavior Other flags may appear in the future.

13.14.4.47 swapping()

```
void Fl_Browser_::swapping (
            void * a,
            void * b )  [protected]
```

This method should be used when two items `a` and `b` are being swapped.
It allows the Fl_Browser_ to update its cache data as needed, schedules a redraw for the two items, and tries to maintain the current selection. This method does not actually swap items, but handles the follow up bookkeeping after items have been swapped.

Parameters

in	*a,b*	Items being swapped.

13.14.4.48 textfont()

`Fl_Font Fl_Browser_::textfont ( ) const [inline]`
Gets the default text font for the lines in the browser.

See also

textfont(), textsize(), textcolor()

13.14.4.49 vposition() [1/2]

`int Fl_Browser_::vposition ( ) const [inline]`
Gets the vertical scroll position of the list as a pixel position `pos`.
The position returned is how many pixels of the list are scrolled off the top edge of the screen. Example: A position of '3' indicates the top 3 pixels of the list are scrolled off the top edge of the screen.

See also

position(), hposition()

13.14.4.50 vposition() [2/2]

```
void Fl_Browser_::vposition (
            int pos )
```
Sets the vertical scroll position of the list to pixel position `pos`.
The position is how many pixels of the list are scrolled off the top edge of the screen. Example: A position of '3' scrolls the top three pixels of the list off the top edge of the screen.

Parameters

in	*pos*	The vertical position (in pixels) to scroll the browser to.

See also

vposition(), hposition()

13.14.5 Member Data Documentation

13.14.5.1 hscrollbar

`Fl_Scrollbar Fl_Browser_::hscrollbar`
Horizontal scrollbar.
Public, so that it can be accessed directly.

13.14.5.2 scrollbar

`Fl_Scrollbar Fl_Browser_::scrollbar`
Vertical scrollbar.
Public, so that it can be accessed directly.
Use `scrollbar_left()` or `scrollbar_right()` to change what side the vertical scrollbar is drawn on.
Use `scrollbar.align(int)` (see `Fl_Widget::align(Fl_Align)`) to change what side either of the scrollbars is drawn on.
If the `FL_ALIGN_LEFT` bit is on, the vertical scrollbar is on the left. If the `FL_ALIGN_TOP` bit is on, the horizontal scrollbar is on the top. Note that only the alignment flags in scrollbar are considered. The flags in hscrollbar however are ignored.
The documentation for this class was generated from the following files:

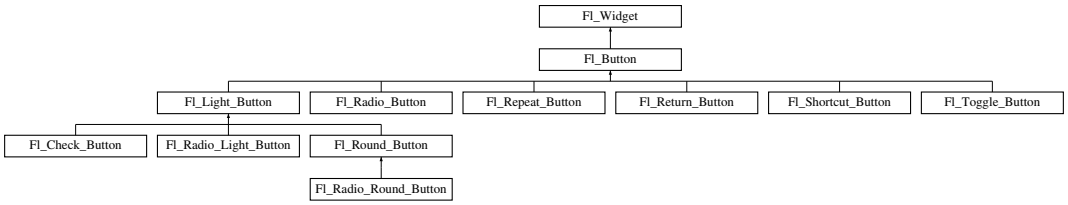
- Fl_Browser_.H
- Fl_Browser_.cxx

13.15 Fl_Button Class Reference

Buttons generate callbacks when they are clicked by the user.

`#include <Fl_Button.H>`

Inheritance diagram for Fl_Button:



Public Member Functions

- int clear ()

  Same as `value(0)`.
- uchar compact () const

  Return true if buttons are rendered as compact buttons.
- void compact (uchar v)

  Decide if buttons should be rendered in compact mode.
- Fl_Boxtype down_box () const

  Returns the current down box type, which is drawn when value() is non-zero.
- void down_box (Fl_Boxtype b)

  Sets the down box type.
- Fl_Color **down_color** () const

  (for backwards compatibility)
- void **down_color** (unsigned c)

  (for backwards compatibility)
- Fl_Button (int X, int Y, int W, int H, const char ∗L=0)

  The constructor creates the button using the given position, size, and label.
- int handle (int) override

  Handles the specified event.
- int set ()

  Same as `value(1)`.
- void **setonly** ()

  Turns on this button and turns off all other radio buttons in the group (calling `value(1)` or `set()` does not do this).
- int shortcut () const

  Returns the current shortcut key for the button.
- void **shortcut** (const char ∗s)

  (for backwards compatibility)
- void shortcut (int s)

  Sets the shortcut key to `s`.
- char **value** () const

  Returns the current value of the button (0 or 1).
- int value (int v)

  Sets the current value of the button.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ **as_gl_window** () const
- virtual Fl_Group ∗ as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const ∗ **as_group** () const
- virtual Fl_Window ∗ as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ **as_window** () const
- void bind_deimage (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗cb, void ∗p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

Checks if the widget value changed since the last callback.

- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.

- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

Sets width ww and height hh accordingly with the label size.

- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.

- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void draw () override

  Draws the widget.
- void **simulate_key_action** ()

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Static Protected Member Functions

- static void **key_release_timeout** (void *)

Static Protected Attributes

- static Fl_Widget_Tracker * **key_release_tracker** = 0

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 , AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.15.1 Detailed Description

Buttons generate callbacks when they are clicked by the user.
You control exactly when and how by changing the values for type(uchar) and when(uchar). Buttons can also generate callbacks in response to `FL_SHORTCUT` events. The button can either have an explicit shortcut(int s) value or a letter shortcut can be indicated in the label() with an '&' character before it. For the label shortcut it does not matter if *Alt* is held down, but if you have an input field in the same window, the user will have to hold down the *Alt* key so that the input field does not eat the event first as an `FL_KEYBOARD` event.

See also

Fl_Widget::shortcut_label(int)

For an Fl_Button object, the type() call returns one of:

- `FL_NORMAL_BUTTON` (0): value() remains unchanged after button press.
- `FL_TOGGLE_BUTTON`: value() is inverted after button press.
- `FL_RADIO_BUTTON`: value() is set to 1 after button press, and all other buttons in the current group with `type() == FL_RADIO_BUTTON` are set to zero.

For an Fl_Button object, the following when() values are useful, the default being `FL_WHEN_RELEASE`:

- `0`: The callback is not done, instead changed() is turned on.
- `FL_WHEN_RELEASE`: The callback is done after the user successfully clicks the button, or when a shortcut is typed. The reason is `FL_REASON_RELEASED`.
- `FL_WHEN_CHANGED`: The callback is done each time the value() changes (when the user pushes and releases the button, and as the mouse is dragged around in and out of the button). The reason is set to `FL_REASON_CHANGED`
- `FL_WHEN_NOT_CHANGED`: The callback is done when the mouse button is released, but the value did not changed. The reason is set to `FL_REASON_SELECTED`

13.15.2 Constructor & Destructor Documentation

13.15.2.1 Fl_Button()

```
Fl_Button::Fl_Button (
            int X,
            int Y,
            int W,
```

```
int H,
const char * L = 0 )
```

The constructor creates the button using the given position, size, and label.
The default box type is box(FL_UP_BOX).
You can control how the button is drawn when ON by setting down_box(). The default is FL_NO_BOX (0) which will select an appropriate box type using the normal (OFF) box type by using fl_down(box()).
Derived classes may handle this differently.
Calling `when()` will tell the button widget when to call the callback.
Setting `FL_WHEN_RELEASE` will call the callback only if the button value changed. It's called during `FL_RELEASE` and `FL_KEYBOARD` events with `FL_REASON_RELEASED` set as the callback reason.
Setting `FL_WHEN_CHANGED` will call the callback with `FL_REASON_CHANGED` every time the value of the button changes during `FL_DRAG`, `FL_RELEASE`, and `FL_KEYBOARD` events.
Setting `FL_WHEN_NOT_CHANGED` will trigger a callback during `FL_RELEASE` events, even if the value of the button die *not* change. For radio buttons, this is also true during `FL_KEYBOARD` events.

Parameters

in	*X,Y,W,H*	position and size of the widget
in	*L*	widget label, default is no label

13.15.3 Member Function Documentation

13.15.3.1 clear()

```
int Fl_Button::clear ( ) [inline]
```

Same as `value(0)`.

See also

value(int v)

13.15.3.2 compact() [1/2]

```
uchar Fl_Button::compact ( ) const [inline]
```

Return true if buttons are rendered as compact buttons.

Returns

0 if compact mode is off, 1 if it is on

See also

compact(bool)

13.15.3.3 compact() [2/2]

```
void Fl_Button::compact (
            uchar v )
```

Decide if buttons should be rendered in compact mode.

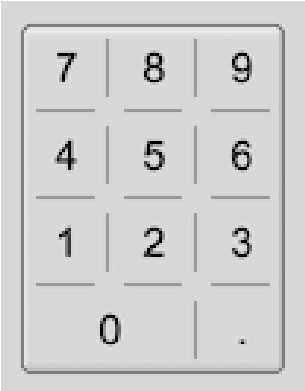


Figure 13.4 compact button keypad using GTK+ Scheme

Figure 13.5 compact buttons in Gleam

In compact mode, the button's surrounding border is altered to visually signal that multiple buttons are functionally linked together. To ensure the correct rendering of buttons in compact mode, all buttons must be part of the same group, positioned close to each other, and aligned with the edges of the group. Any button outlines not in contact with the parent group's outline will be displayed as separators.

Parameters

in	*v*	switch compact mode on (1) or off (0)

13.15.3.4 down_box() [1/2]

```
Fl_Boxtype Fl_Button::down_box ( ) const  [inline]
```

Returns the current down box type, which is drawn when value() is non-zero.

Return values

Fl_Boxtype	

13.15.3.5 down_box() [2/2]

```
void Fl_Button::down_box (
            Fl_Boxtype b )  [inline]
```

Sets the down box type.

The default value of 0 causes FLTK to figure out the correct matching down version of box().

Some derived classes (e.g. Fl_Round_Button and Fl_Light_Button use down_box() for special purposes. See docs

of these classes.

Parameters

in	*b*	down box type

13.15.3.6 draw()

```
void Fl_Button::draw ( ) [override], [protected], [virtual]
```

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();                 // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.
Reimplemented in Fl_Light_Button, Fl_Return_Button, and Fl_Shortcut_Button.

13.15.3.7 handle()

```
int Fl_Button::handle (
            int event ) [override], [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to *override* the behavior of the base class. This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to *filter* event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you *consumed* the event by returning 1 even if you didn't *do* anything with it.

Parameters

in	*event*	the kind of event received

Return values

0	if the event was not used or understood
1	if the event was used and can be deleted

See also

Fl_Event

Reimplemented from Fl_Widget.
Reimplemented in Fl_Light_Button, Fl_Repeat_Button, Fl_Return_Button, and Fl_Shortcut_Button.

13.15.3.8 set()

```
int Fl_Button::set ( ) [inline]
```

Same as `value(1)`.

See also

value(int v)

13.15.3.9 shortcut() [1/2]

```
int Fl_Button::shortcut ( ) const  [inline]
```

Returns the current shortcut key for the button.

Return values

int	

13.15.3.10 shortcut() [2/2]

```
void Fl_Button::shortcut (
            int s )  [inline]
```

Sets the shortcut key to `s`.
Setting this overrides the use of '&' in the label(). The value is a bitwise OR of a key and a set of shift flags, for example: `FL_ALT | 'a'`, or `FL_ALT | (FL_F + 10)`, or just `'a'`. A value of 0 disables the shortcut.
The key can be any value returned by Fl::event_key(), but will usually be an ASCII letter. Use a lower-case letter unless you require the shift key to be held down.
The shift flags can be any set of values accepted by Fl::event_state(). If the bit is on, that shift key must be pushed. Meta, Alt, Ctrl, and Shift must be off if they are not in the shift flags (zero for the other bits indicates a "don't care" setting).

Parameters

in	*s*	bitwise OR of key and shift flags

13.15.3.11 value()

```
int Fl_Button::value (
            int v )
```

Sets the current value of the button.
A non-zero value sets the button to 1 (ON), and zero sets it to 0 (OFF).

Parameters

in	*v*	button value.

See also

set(), clear()

The documentation for this class was generated from the following files:

- Fl_Button.H
- Fl_Button.cxx

13.16 Fl_Cairo_State Class Reference

Contains all the necessary info on the current cairo context.
`#include <Fl_Cairo.H>`

Public Member Functions

- bool **autolink** () const

  Gets the autolink option. See Fl::cairo_autolink_context(bool)
- void **autolink** (bool b)

  Sets the autolink option, only available with –enable-cairoext.
- cairo_t ∗ **cc** () const

  Gets the current cairo context.
- void cc (cairo_t ∗c, bool own=true)

  Sets the current cairo context.
- void ∗ **gc** () const

  Gets the last gc attached to a cc.
- void **gc** (void ∗c)

  Sets the gc `c` *to keep track on.*
- void ∗ **window** () const

  Gets the last window attached to a cc.
- void **window** (void ∗w)

  Sets the window `w` *to keep track on.*

13.16.1 Detailed Description

Contains all the necessary info on the current cairo context.
A private internal & unique corresponding object is created to permit cairo context state handling while keeping it opaque. For internal use only.

Note

Only available if one or both of the CMake options FLTK_OPTION_CAIRO_WINDOW or FLTK_OPTION_↩ CAIRO_EXT is set (ON).

13.16.2 Member Function Documentation

13.16.2.1 cc()

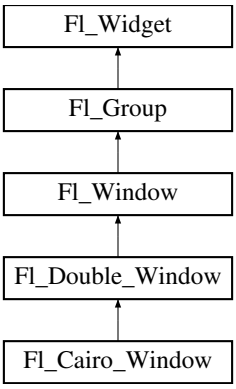
```
void Fl_Cairo_State::cc (
            cairo_t * c,
            bool own = true )  [inline]
```

Sets the current cairo context.
`own` == *true* (the default) indicates that the cairo context `c` will be deleted by FLTK internally when another cc is set later.
`own` == *false* indicates cc deletion is handled externally by the user program.
The documentation for this class was generated from the following files:

- Fl_Cairo.H
- Fl_Cairo.cxx

13.17 Fl_Cairo_Window Class Reference

This defines an FLTK window with Cairo support.
`#include <Fl_Cairo_Window.H>`
Inheritance diagram for Fl_Cairo_Window:



Public Types

- typedef void(∗ **cairo_draw_cb**) (Fl_Cairo_Window ∗self, cairo_t ∗def)

  The Cairo draw callback prototype you need to implement.

Public Types inherited from Fl_Window

- typedef struct HICON__ ∗ **HICON**

Public Member Functions

- **Fl_Cairo_Window** (int W, int H, const char ∗L=0)
- **Fl_Cairo_Window** (int X, int Y, int W, int H, const char ∗L=0)
- void set_draw_cb (cairo_draw_cb cb)

  You must provide a draw callback that implements your Cairo rendering.

Public Member Functions inherited from Fl_Double_Window

- Fl_Double_Window ∗ as_double_window () override

  Return non-null if this is an Fl_Double_Window object.
- **Fl_Double_Window** (int W, int H, const char ∗l=0)

  Creates a new Fl_Double_Window widget using the given position, size, and label (title) string.
- **Fl_Double_Window** (int X, int Y, int W, int H, const char ∗l=0)

  See Fl_Double_Window::Fl_Double_Window(int w, int h, const char ∗label = 0)
- void flush () override

  Forces the window to be drawn, this window is also made current and calls draw().
- void hide () override

  Makes a widget invisible.
- void resize (int, int, int, int) override

  Changes the size or position of the widget.
- void show () override

  Makes a widget visible.
- void **show** (int a, char ∗∗b)

  Same as Fl_Window::show(int a, char ∗∗b)
- ∼Fl_Double_Window ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Window

- void allow_expand_outside_parent ()

  Allow this subwindow to expand outside the area of its parent window.
- virtual class Fl_Overlay_Window ∗ as_overlay_window ()

  Return non-null if this is an Fl_Overlay_Window object.
- Fl_Window const ∗ as_window () const override
- Fl_Window ∗ as_window () override

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- unsigned int **border** () const

  Returns whether the window possesses a border.
- void border (int b)

  Sets whether or not the window manager border is around the window.
- void clear_border ()

  Fast inline function to turn the window manager border off.
- void clear_modal_states ()

  Clears the "modal" flags and converts a "modal" or "non-modal" window back into a "normal" window.
- void **copy_label** (const char ∗a)

  Sets the window titlebar label to a copy of a character string.
- void cursor (const Fl_RGB_Image ∗, int, int)

  Changes the cursor for this window using the provided image as cursor's shape.
- void cursor (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)

  For back compatibility only.
- void cursor (Fl_Cursor)

  Changes the cursor for this window.
- int decorated_h () const

  Returns the window height including any window title bar and any frame added by the window manager.
- int decorated_w () const

  Returns the window width including any frame added by the window manager.
- void default_cursor (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)

  For back compatibility only.
- void default_cursor (Fl_Cursor)

  Sets the default window cursor.
- void **draw_backdrop** ()

  Draw the background image if one is set and is aligned inside.
- Fl_Window (int w, int h, const char ∗title=0)

  Creates a window from the given width `w`, height `h`, and `title`.
- Fl_Window (int x, int y, int w, int h, const char ∗title=0)

  Creates a window from the given position (x, y), size (w, h) and title.
- void free_position ()

  Undoes the effect of a previous resize() or show() so that the next time show() is called the window manager is free to position the window.
- void fullscreen ()

  Makes the window completely fill one or more screens, without any window manager border visible.
- unsigned int **fullscreen_active** () const

  Returns non zero if FULLSCREEN flag is set, 0 otherwise.
- void **fullscreen_off** ()

  Turns off any side effects of fullscreen()
- void **fullscreen_off** (int X, int Y, int W, int H)

  Turns off any side effects of fullscreen() and does resize(X,Y,W,H).
- void fullscreen_screens (int top, int bottom, int left, int right)

Sets which screens should be used when this window is in fullscreen mode.

- uchar get_size_range (int *minw, int *minh, int *maxw=NULL, int *maxh=NULL, int *dw=NULL, int *dh=NULL, int *aspect=NULL) const

  Gets the allowable range to which the user can resize this window.
- int handle (int) override

  Handles the specified event.
- void hide () override

  Removes the window from the screen.
- void **hotspot** (const Fl_Widget &p, int offscreen=0)

  See void Fl_Window::hotspot(int x, int y, int offscreen = 0)
- void **hotspot** (const Fl_Widget *, int offscreen=0)

  See void Fl_Window::hotspot(int x, int y, int offscreen = 0)
- void hotspot (int x, int y, int offscreen=0)

  Positions the window so that the mouse is pointing at the given position, or at the center of the given widget, which may be the window itself.
- const void * icon () const

  Gets the current icon window target dependent data.
- void icon (const Fl_RGB_Image *)

  Sets or resets a single window icon.
- void icon (const void *ic)

  Platform-specific method to set the window icon usable on Windows and X11 only.
- void iconize ()

  Iconifies the window.
- const char * **iconlabel** () const

  See void Fl_Window::iconlabel(const char)*
- void **iconlabel** (const char *)

  Sets the icon label.
- void icons (const Fl_RGB_Image *[], int)

  Sets the window icons.
- void icons (HICON big_icon, HICON small_icon)

  Sets the window icons using HICON handles (Windows platform only).
- const char * **label** () const

  See void Fl_Window::label(const char)*
- void **label** (const char *)

  Sets the window title bar label.
- void **label** (const char *label, const char *iconlabel)

  Sets the icon label.
- void make_current ()

  Sets things up so that the drawing functions in <FL/fl_draw.H> will go into this window.
- void maximize ()

  Maximizes a top-level window to its current screen.
- unsigned int **maximize_active** () const

  Returns whether the window is currently maximized.
- unsigned int **menu_window** () const

  Returns true if this window is a menu window.
- unsigned int modal () const

  Returns true if this window is modal.
- unsigned int **non_modal** () const

  Returns true if this window is modal or non-modal.

- fl_uintptr_t os_id () const

  Returns a platform-specific identification of a shown window, or 0 if not shown.
- unsigned int **override** () const

  Returns non zero if OVERRIDE flag is set, 0 otherwise.
- void resize (int X, int Y, int W, int H) override

  Changes the size and position of the window.
- int screen_num () const

  The number of the screen containing the mapped window.
- void screen_num (int screen_num)

  Set the number of the screen where to map the window.
- void set_menu_window ()

  Marks the window as a menu window.
- void set_modal ()

  A "modal" window, when shown(), will prevent any events from being delivered to other windows in the same program, and will also remain on top of the other windows (if the X window manager supports the "transient for" property).
- void set_non_modal ()

  A "non-modal" window (terminology borrowed from Microsoft Windows) acts like a modal() one in that it remains on top, but it has no effect on event delivery.
- void **set_override** ()

  Activates the flags NOBORDER|OVERRIDE.
- void set_tooltip_window ()

  Marks the window as a tooltip window.
- const Fl_Image * shape () const

  Returns the image controlling the window shape or NULL.
- void shape (const Fl_Image &b)

  Set the window's shape with an Fl_Image.
- void shape (const Fl_Image *img)

  Assigns a non-rectangular shape to the window.
- void show () override

  Puts the window on the screen.
- void show (int argc, char **argv)

  Puts the window on the screen with show() and parses command-line arguments.
- int shown () const

  Returns non-zero if show() has been called (but not hide()).
- void size_range (int minw, int minh, int maxw=0, int maxh=0, int dw=0, int dh=0, int aspect=0)

  Sets the allowable range to which the user can resize this window.
- unsigned int **tooltip_window** () const

  Returns true if this window is a tooltip window.
- void un_maximize ()

  Returns a previously maximized top-level window to its previous size.
- void wait_for_expose ()

  Waits for the window to be displayed after calling show().
- int **x_root** () const

  Gets the x position of the window on the screen.
- const char * xclass () const

  Returns the xclass for this window, or a default.
- void xclass (const char *c)

  Sets the xclass for this window.
- int **y_root** () const

  Gets the y position of the window on the screen.
- virtual ~Fl_Window ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Group

- Fl_Widget *& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget *o)

  See void Fl_Group::add(Fl_Widget &w)
- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget *const * array () const

  Returns a pointer to the internal array of children.
- Fl_Group const * as_group () const override
- Fl_Group * as_group () override

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const

  Returns the n'th child.
- int **children** () const

  Returns how many child widgets the group has.
- void clear ()

  Deletes all child widgets from memory recursively.
- unsigned int clip_children () const

  Returns the current clipping mode.
- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)

  Removes the widget at `index` from the group and deletes it.
- void end ()

  Exactly the same as current(this->parent()).
- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*
- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.
- void **forms_end** ()

  This is for forms compatibility only.
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

  This does insert(w, find(before)).
- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.
- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.

- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

Sets the current callback function for the widget.

- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.

- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

Sets the label type.

- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.

- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void draw () override

  Overloaded to provide Cairo callback support.

Protected Member Functions inherited from Fl_Window

- void default_size_range ()

  Protected method to calculate the default size range of a window.
- void draw () override

  Draws the widget.
- int force_position () const

  Returns the internal state of the window's `FORCE_POSITION` flag.
- void force_position (int force)

  Sets an internal flag that tells FLTK and the window manager to honor position requests.
- void free_icons ()

  Deletes all icons previously attached to the window.
- int is_resizable ()

  Protected method to determine whether a window is resizable.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()

  Returns the internal array of widget sizes and positions.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget *, int)

  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)

  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)

  Allow derived groups to act when a child widget is removed from the group.
- int * sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.

- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Window

- static Fl_Window * current ()

  Returns the last window that was made current.
- static void **default_callback** (Fl_Window *, void *v)

  Back compatibility: Sets the default callback v for win to call on close event.
- static void default_icon (const Fl_RGB_Image *)

  Sets a single default window icon.
- static void default_icons (const Fl_RGB_Image *[ ], int)

  Sets the default window icons.
- static void default_icons (HICON big_icon, HICON small_icon)

  Sets the default window icons (Windows platform only).
- static const char * default_xclass ()

  Returns the default xclass.
- static void default_xclass (const char *)

  Sets the default window xclass.
- static bool is_a_rescale ()

  Returns true when a window is being rescaled.
- static char show_next_window_iconic ()

  Returns the static flag whether the next window should be opened iconified.
- static void show_next_window_iconic (char stat)

  Sets a static flag whether the next window should be opened iconified.

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()

  Returns the currently active group.
- static void current (Fl_Group *g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 , AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Static Protected Attributes inherited from Fl_Window

- static Fl_Window * current_

  Stores the last window that was made current.

13.17.1 Detailed Description

This defines an FLTK window with Cairo support.
This class overloads the virtual draw() method for you, so that the only thing you have to do is to provide your Cairo code. All Cairo context handling is achieved transparently.
The default coordinate system for Cairo drawing commands within Fl_Cairo_Window is FLTK's coordinate system, where the `x,y,w,h` values are relative to the top/left corner of the Fl_Cairo_Window, as one would expect with regular FLTK drawing commands, e.g.: `(0 x w-1), (0 y h-1)`. **Example:**

```
static void my_cairo_draw_cb(Fl_Cairo_Window *window, cairo_t *cr) {
  // Draw an "X"
  const double xmax = (window->w() - 1);
  const double ymax = (window->h() - 1);
  cairo_set_line_width(cr, 1.00);                                  // line width for drawing
  cairo_set_source_rgb(cr, 1.0, 0.5, 0.0);                         // orange
  cairo_move_to(cr, 0.0, 0.0);  cairo_line_to(cr, xmax, ymax);     // draw diagonal "\"
  cairo_move_to(cr, 0.0, ymax); cairo_line_to(cr, xmax, 0.0);      // draw diagonal "/"
  cairo_stroke(cr);                                                // stroke the lines
}
```

The FLTK coordinate system differs from the default native Cairo coordinate system which uses normalized `(0.0 ... 1.0)` values for x and y, e.g.: `(0 x 1.0), (0 y 1.0)`. So beware of this when copy/pasting Cairo example programs that assume normalized values. If need be, you can revert to the Cairo coordinate system by simply calling `cairo_scale()` with the widget's `w()` and `h()` values. **Example:**

```
static void my_cairo_draw_cb(Fl_Cairo_Window *window, cairo_t *cr) {
  cairo_scale(cr, window->w(), window->h());    // use Cairo's default coordinate system
  [..use 0.0 to 1.0 values from here on..]
}
```

See also

examples/cairo-draw-x.cxx

test/cairo_test.cxx

Note

Class Fl_Cairo_Window requires the FLTK library to have been built with CMake option FLTK_OPTION_↩ CAIRO_WINDOW or FLTK_OPTION_CAIRO_EXT.

You can alternatively define your custom Cairo FLTK window, and thus at least override the draw() method to provide custom Cairo support. In this case you will probably use Fl::cairo_make_current(Fl_Window∗) to attach a context to your window. You should do this only when your window is the current window.

See also

Fl_Window::current()

13.17.2 Member Function Documentation

13.17.2.1 draw()

```
void Fl_Cairo_Window::draw (
            void ) [inline], [override], [protected], [virtual]
```

Overloaded to provide Cairo callback support.
Implements Fl_Widget.

13.17.2.2 set_draw_cb()

```
void Fl_Cairo_Window::set_draw_cb (
            cairo_draw_cb cb ) [inline]
```

You must provide a draw callback that implements your Cairo rendering.
This method permits you to set your Cairo callback to `cb`.
The documentation for this class was generated from the following file:

- Fl_Cairo_Window.H

13.18 Fl_Callback_User_Data Class Reference

A class prototype that allows for additional data in callbacks.
`#include <Fl_Widget.H>`

Public Member Functions

- virtual ∼**Fl_Callback_User_Data** ()
  Destructor.

Protected Member Functions

- **Fl_Callback_User_Data** ()
  Protected constructor.

13.18.1 Detailed Description

A class prototype that allows for additional data in callbacks.
Users can derive this class and pass objects of such derived classes to widget callbacks. Widgets can take ownership of the callback data, deleting the data when the widget itself is deleted.
The destructor of this class is virtual, allowing for additional code to deallocate resources when the user data is deleted.

See also

FL_FUNCTION_CALLBACK_3, FL_METHOD_CALLBACK_1, FL_INLINE_CALLBACK_2

Fl_Widget::callback(Fl_Callback∗, Fl_Callback_User_Data∗, bool)

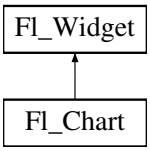
Fl_Widget::user_data(Fl_Callback_User_Data∗, bool)

The documentation for this class was generated from the following file:

- Fl_Widget.H

13.19 Fl_Chart Class Reference

Fl_Chart displays simple charts.
`#include <Fl_Chart.H>`
Inheritance diagram for Fl_Chart:



Public Member Functions

- void add (double val, const char *str=0, unsigned col=0)

  Adds the data value `val` with optional label `str` and color `col` to the chart.
- uchar autosize () const

  Gets whether the chart will automatically adjust the bounds of the chart.
- void autosize (uchar n)

  Sets whether the chart will automatically adjust the bounds of the chart.
- void bounds (double *a, double *b) const

  Gets the lower and upper bounds of the chart values.
- void bounds (double a, double b)

  Sets the lower and upper bounds of the chart values.
- void **clear** ()

  Removes all values from the chart.
- Fl_Chart (int X, int Y, int W, int H, const char *L=0)

  Create a new Fl_Chart widget using the given position, size and label string.
- void insert (int ind, double val, const char *str=0, unsigned col=0)

  Inserts a data value `val` at the given position `ind`.
- int **maxsize** () const

  Gets the maximum number of data values for a chart.
- void maxsize (int m)

  Sets the maximum number of data values for a chart.
- void replace (int ind, double val, const char *str=0, unsigned col=0)

  Replaces a data value `val` at the given position `ind`.
- int **size** () const

  Returns the number of data values in the chart.
- void size (int W, int H)

  Sets the widget size (width, height).
- Fl_Color **textcolor** () const

  Gets the chart's text color.
- void **textcolor** (Fl_Color n)

  Sets the chart's text color to `n`.
- Fl_Font **textfont** () const

  Gets the chart's text font.
- void **textfont** (Fl_Font s)

  Sets the chart's text font to `s`.
- Fl_Fontsize **textsize** () const

  Gets the chart's text size.
- void **textsize** (Fl_Fontsize s)

  Sets the chart's text size to `s`.
- ~**Fl_Chart** ()

  Destroys the Fl_Chart widget and all of its data.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ **as_gl_window** () const
- virtual Fl_Group ∗ as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const ∗ **as_group** () const
- virtual Fl_Window ∗ as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ **as_window** () const
- void bind_deimage (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗cb, void ∗p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

Checks if the widget value changed since the last callback.

- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.

- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual int handle (int event)

  Handles the specified event.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

Sets the label type.

- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.

- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void draw () override

  Draws the Fl_Chart widget.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Static Protected Member Functions

- static void draw_barchart (int x, int y, int w, int h, int numb, FL_CHART_ENTRY entries[ ], double min, double max, int autosize, int maxnumb, Fl_Color textcolor)

  Draws a bar chart.
- static void draw_horbarchart (int x, int y, int w, int h, int numb, FL_CHART_ENTRY entries[ ], double min, double max, int autosize, int maxnumb, Fl_Color textcolor)

  Draws a horizontal bar chart.
- static void draw_linechart (int type, int x, int y, int w, int h, int numb, FL_CHART_ENTRY entries[ ], double min, double max, int autosize, int maxnumb, Fl_Color textcolor)

  Draws a line chart.
- static void draw_piechart (int x, int y, int w, int h, int numb, FL_CHART_ENTRY entries[ ], int special, Fl_Color textcolor)

  Draws a pie chart.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.19.1 Detailed Description

Fl_Chart displays simple charts.
It is provided for Forms compatibility.

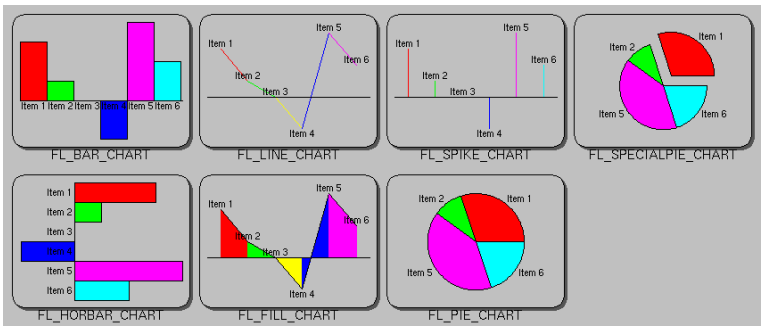


Figure 13.6 Fl_Chart

Todo Refactor Fl_Chart::type() information.

The type of an Fl_Chart object can be set using type(uchar t) to:

Chart Type	Description
FL_BAR_CHART	Each sample value is drawn as a vertical bar.
FL_FILLED_CHART	The chart is filled from the bottom of the graph to the sample values.
FL_HORBAR_CHART	Each sample value is drawn as a horizontal bar.
FL_LINE_CHART	The chart is drawn as a polyline with vertices at each sample value.
FL_PIE_CHART	A pie chart is drawn with each sample value being drawn as a proportionate slice in the circle.

Chart Type	Description
FL_SPECIALPIE_CHART	Like `FL_PIE_CHART`, but the first slice is separated from the pie.
FL_SPIKE_CHART	Each sample value is drawn as a vertical line.

13.19.2 Constructor & Destructor Documentation

13.19.2.1 Fl_Chart()

```
Fl_Chart::Fl_Chart (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

Create a new Fl_Chart widget using the given position, size and label string.
The default boxstyle is `FL_NO_BOX`.

Parameters

in	*X,Y,W,H*	position and size of the widget
in	*L*	widget label, default is no label

13.19.3 Member Function Documentation

13.19.3.1 add()

```
void Fl_Chart::add (
            double val,
            const char * str = 0,
            unsigned col = 0 )
```

Adds the data value `val` with optional label `str` and color `col` to the chart.

Parameters

in	*val*	data value
in	*str*	optional data label
in	*col*	optional data color

13.19.3.2 autosize() [1/2]

```
uchar Fl_Chart::autosize ( ) const  [inline]
```

Gets whether the chart will automatically adjust the bounds of the chart.

Returns

non-zero if auto-sizing is enabled and zero if disabled.

13.19.3.3 autosize() [2/2]

```
void Fl_Chart::autosize (
            uchar n )  [inline]
```

Sets whether the chart will automatically adjust the bounds of the chart.

Parameters

in	*n*	non-zero to enable automatic resizing, zero to disable.

13.19.3.4 bounds() [1/2]

```
void Fl_Chart::bounds (
            double * a,
            double * b ) const  [inline]
```

Gets the lower and upper bounds of the chart values.

Parameters

out	*a,b*	are set to lower, upper

13.19.3.5 bounds() [2/2]

```
void Fl_Chart::bounds (
            double a,
            double b )
```

Sets the lower and upper bounds of the chart values.

Parameters

in	*a,b*	are used to set lower, upper

13.19.3.6 draw()

```
void Fl_Chart::draw (
            void  )  [override], [protected], [virtual]
```

Draws the Fl_Chart widget.
Implements Fl_Widget.

13.19.3.7 draw_barchart()

```
void Fl_Chart::draw_barchart (
            int x,
            int y,
            int w,
            int h,
            int numb,
            FL_CHART_ENTRY entries[],
            double min,
            double max,
            int autosize,
            int maxnumb,
            Fl_Color textcolor )  [static], [protected]
```

Draws a bar chart.
`x`, `y`, `w`, `h` is the bounding box, `entries` the array of `numb` entries, and `min` and `max` the boundaries.

Parameters

in	*x,y,w,h*	Widget position and size
in	*numb*	Number of values
in	*entries*	Array of values
in	*min*	Lower boundary
in	*max*	Upper boundary
in	*autosize*	Whether the chart autosizes
in	*maxnumb*	Maximal number of entries
in	*textcolor*	Text color

13.19.3.8 draw_horbarchart()

```
void Fl_Chart::draw_horbarchart (
            int x,
            int y,
            int w,
            int h,
            int numb,
            FL_CHART_ENTRY entries[],
            double min,
            double max,
            int autosize,
            int maxnumb,
            Fl_Color textcolor )  [static], [protected]
```

Draws a horizontal bar chart.

`x`, `y`, `w`, `h` is the bounding box, `entries` the array of `numb` entries, and `min` and `max` the boundaries.

Parameters

in	*x,y,w,h*	Widget position and size
in	*numb*	Number of values
in	*entries*	Array of values
in	*min*	Lower boundary
in	*max*	Upper boundary
in	*autosize*	Whether the chart autosizes
in	*maxnumb*	Maximal number of entries
in	*textcolor*	Text color

13.19.3.9 draw_linechart()

```
void Fl_Chart::draw_linechart (
            int type,
            int x,
            int y,
            int w,
            int h,
            int numb,
            FL_CHART_ENTRY entries[],
            double min,
            double max,
            int autosize,
            int maxnumb,
            Fl_Color textcolor )  [static], [protected]
```

Draws a line chart.

`x`, `y`, `w`, `h` is the bounding box, `entries` the array of `numb` entries, and `min` and `max` the boundaries.

Parameters

in	*type*	Chart type
in	*x,y,w,h*	Widget position and size
in	*numb*	Number of values
in	*entries*	Array of values
in	*min*	Lower boundary
in	*max*	Upper boundary
in	*autosize*	Whether the chart autosizes
in	*maxnumb*	Maximal number of entries

Parameters

in	*textcolor*	Text color

13.19.3.10 draw_piechart()

```
void Fl_Chart::draw_piechart (
            int x,
            int y,
            int w,
            int h,
            int numb,
            FL_CHART_ENTRY entries[],
            int special,
            Fl_Color textcolor )  [static], [protected]
```

Draws a pie chart.

Parameters

in	*x,y,w,h*	bounding box
in	*numb*	number of chart entries
in	*entries*	array of chart entries
in	*special*	special (?)
in	*textcolor*	text color

13.19.3.11 insert()

```
void Fl_Chart::insert (
            int ind,
            double val,
            const char * str = 0,
            unsigned col = 0 )
```

Inserts a data value `val` at the given position `ind`.
Position 1 is the first data value.

Parameters

in	*ind*	insertion position
in	*val*	data value
in	*str*	optional data label
in	*col*	optional data color

13.19.3.12 maxsize()

```
void Fl_Chart::maxsize (
            int m )
```

Sets the maximum number of data values for a chart.
If you do not call this method then the chart will be allowed to grow to any size depending on available memory.

Parameters

in	*m*	maximum number of data values allowed.

13.19.3.13 replace()

```
void Fl_Chart::replace (
            int ind,
            double val,
            const char * str = 0,
            unsigned col = 0 )
```

Replaces a data value `val` at the given position `ind`.
Position 1 is the first data value.

Parameters

in	*ind*	insertion position
in	*val*	data value
in	*str*	optional data label
in	*col*	optional data color

13.19.3.14 size()

```
void Fl_Chart::size (
            int W,
            int H )  [inline]
```

Sets the widget size (width, height).
This is the same as calling Fl_Widget::size(int W, int H);

Parameters

in	*W,H*	new width and height of the widget

The documentation for this class was generated from the following files:

- Fl_Chart.H
- Fl_Chart.cxx

13.20 FL_CHART_ENTRY Struct Reference

For internal use only.
`#include <Fl_Chart.H>`

Public Attributes

- unsigned **col**

  For internal use only.
- char **str** [FL_CHART_LABEL_MAX+1]

  For internal use only.
- float **val**

  For internal use only.

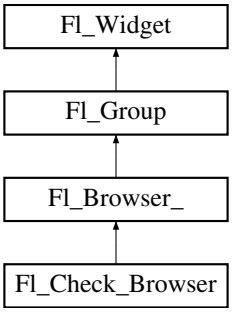
13.20.1 Detailed Description

For internal use only.
The documentation for this struct was generated from the following file:

- Fl_Chart.H

13.21 Fl_Check_Browser Class Reference

The Fl_Check_Browser widget displays a scrolling list of text lines that may be selected and/or checked by the user.
`#include <Fl_Check_Browser.H>`
Inheritance diagram for Fl_Check_Browser:



Public Member Functions

- int add (char ∗s)

  Add a new unchecked line to the end of the browser.
- int add (char ∗s, int b)

  Add a new line to the end of the browser.
- int **add** (const char ∗s)

  See int Fl_Check_Browser::add(char ∗s)
- int **add** (const char ∗s, int b)

  See int Fl_Check_Browser::add(char ∗s)
- void **check_all** ()

  Sets all the items checked.
- void **check_none** ()

  Sets all the items unchecked.
- int **checked** (int item) const

  Gets the current status of item item.
- void **checked** (int item, int b)

  Sets the check status of item item to b.
- void **clear** ()

  Remove every item from the browser.
- **Fl_Check_Browser** (int x, int y, int w, int h, const char ∗l=0)

  The constructor makes an empty browser.
- void ∗ item_at (int index) const override

  This method must be provided by the subclass to return the item for the specified `index`.
- void **item_swap** (int ia, int ib)
- void item_swap (void ∗a, void ∗b) override

  This optional method should be provided by the subclass to efficiently swap browser items `a` and `b`, such as for sorting.
- int **nchecked** () const

  Returns how many items are currently checked.
- int nitems () const

  Returns how many lines are in the browser.
- int remove (int item)

  Remove line n and make the browser one line shorter.
- void **set_checked** (int item)

  Equivalent to Fl_Check_Browser::checked(item, 1).

- char * **text** (int item) const

  Return a pointer to an internal buffer holding item item's text.
- int **value** () const

  Returns the index of the currently selected item.
- ~**Fl_Check_Browser** ()

  The destructor deletes all list items and destroys the browser.

Public Member Functions inherited from Fl_Browser_

- int deselect (int docallbacks=0)

  Deselects all items in the list and returns 1 if the state changed or 0 if it did not.
- void display (void *item)

  Displays the `item`, scrolling the list as necessary.
- uchar **has_scrollbar** () const

  Returns the current scrollbar mode, see Fl_Browser_::has_scrollbar(uchar)
- void has_scrollbar (uchar mode)

  Sets whether the widget should have scrollbars or not (default Fl_Browser_::BOTH).
- int hposition () const

  Gets the horizontal scroll position of the list as a pixel position `pos`.
- void hposition (int)

  Sets the horizontal scroll position of the list to pixel position `pos`.
- int linespacing () const

  Return the height of additional spacing between browser lines.
- void linespacing (int pixels)

  Add some space between browser lines.
- int position () const
- void position (int pos)
- void **position** (int x, int y)
- void resize (int X, int Y, int W, int H) override

  Repositions and/or resizes the browser.
- void scrollbar_left ()

  Moves the vertical scrollbar to the lefthand side of the list.
- void scrollbar_right ()

  Moves the vertical scrollbar to the righthand side of the list.
- int scrollbar_size () const

  Gets the current size of the scrollbars' troughs, in pixels.
- void scrollbar_size (int newSize)

  Sets the pixel size of the scrollbars' troughs to `newSize`, in pixels.
- int select (void *item, int val=1, int docallbacks=0)

  Sets the selection state of `item` to `val`, and returns 1 if the state changed or 0 if it did not.
- int select_only (void *item, int docallbacks=0)

  Selects `item` and returns 1 if the state changed or 0 if it did not.
- void sort (int flags=0)

  Sort the items in the browser based on `flags`.
- Fl_Color **textcolor** () const

  Gets the default text color for the lines in the browser.
- void **textcolor** (Fl_Color col)

  Sets the default text color for the lines in the browser to color `col`.
- Fl_Font textfont () const

  Gets the default text font for the lines in the browser.
- void **textfont** (Fl_Font font)

Sets the default text font for the lines in the browser to `font`.

- Fl_Fontsize **textsize** () const

  Gets the default text size (in pixels) for the lines in the browser.

- void **textsize** (Fl_Fontsize newSize)

  Sets the default text size (in pixels) for the lines in the browser to `size`.

- int vposition () const

  Gets the vertical scroll position of the list as a pixel position `pos`.

- void vposition (int pos)

  Sets the vertical scroll position of the list to pixel position `pos`.

Public Member Functions inherited from Fl_Group

- Fl_Widget *& **_ddfdesign_kludge** ()

  This is for forms compatibility only.

- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.

- void **add** (Fl_Widget *o)

  See void Fl_Group::add(Fl_Widget &w)

- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.

- Fl_Widget *const * array () const

  Returns a pointer to the internal array of children.

- Fl_Group const * as_group () const override
- Fl_Group * as_group () override

  Returns an Fl_Group pointer if this widget is an Fl_Group.

- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.

- Fl_Widget * child (int n) const

  Returns the n'th child.

- int **children** () const

  Returns how many child widgets the group has.

- void clear ()

  Deletes all child widgets from memory recursively.

- unsigned int clip_children () const

  Returns the current clipping mode.

- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.

- virtual int delete_child (int n)

  Removes the widget at `index` from the group and deletes it.

- void end ()

  Exactly the same as current(this->parent()).

- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*

- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.

- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.

- void **forms_end** ()

  This is for forms compatibility only.

- int handle (int) override

  Handles the specified event.

- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

  This does insert(w, find(before)).
- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.
- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.
- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- void resize (int, int, int, int) override

  Resizes the Fl_Group widget and all of its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

Sets the damage bits for an area inside the widget.

- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.

- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

Makes the widget visible.

- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.

- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- int handle (int) override

  Handles the `event` within the normal widget bounding box.
- void item_draw (void *, int, int, int, int) const override

  This method must be provided by the subclass to draw the `item` in the area indicated by `X`, `Y`, `W`, `H`.
- void * item_first () const override

  This method must be provided by the subclass to return the first item in the list.
- int item_height (void *) const override

  This method must be provided by the subclass to return the height of `item` in pixels.
- void * item_next (void *) const override

  This method must be provided by the subclass to return the item in the list after `item`.
- void * item_prev (void *) const override

  This method must be provided by the subclass to return the item in the list before `item`.
- void item_select (void *, int) override

  This method must be implemented by the subclass if it supports multiple selections; sets the selection state to `val` for the `item`.
- int item_selected (void *) const override

  This method must be implemented by the subclass if it supports multiple selections; returns the selection state for `item`.
- const char * item_text (void *item) const override

  This optional method returns a string (label) that may be used for sorting.
- int item_width (void *) const override

  This method must be provided by the subclass to return the width of the `item` in pixels.

Protected Member Functions inherited from Fl_Browser_

- void bbox (int &X, int &Y, int &W, int &H) const

  Returns the bounding box for the interior of the list's display window, inside the scrollbars.
- void deleting (void *item)

  This method should be used when `item` is being deleted from the list.
- int displayed (void *item) const

  Returns non-zero if `item` has been scrolled to a position where it is being displayed.
- void draw () override

  Draws the list within the normal widget bounding box.
- void * find_item (int ypos)

  This method returns the item under mouse y position `ypos`.
- Fl_Browser_ (int X, int Y, int W, int H, const char *L=0)

  The constructor makes an empty browser.

- virtual int full_height () const

  This method may be provided by the subclass to indicate the full height of the item list, in pixels.
- virtual int full_width () const

  This method may be provided by the subclass to indicate the full width of the item list, in pixels.
- virtual int incr_height () const

  This method may be provided to return the average height of all items to be used for scrolling.
- void inserting (void *a, void *b)

  This method should be used when an item is in the process of being inserted into the list.
- virtual void * item_last () const

  This method must be provided by the subclass to return the last item in the list.
- virtual int item_quick_height (void *item) const

  This method may be provided by the subclass to return the height of the `item`, in pixels.
- int leftedge () const

  This method returns the X position of the left edge of the list area after adjusting for the scrollbar and border, if any.
- void new_list ()

  This method should be called when the list data is completely replaced or cleared.
- void redraw_line (void *item)

  This method should be called when the contents of `item` has changed, but not its height.
- void redraw_lines ()

  This method will cause the entire list to be redrawn.
- void replacing (void *a, void *b)

  This method should be used when item `a` is being replaced by item `b`.
- void * selection () const

  Returns the item currently selected, or NULL if there is no selection.
- void swapping (void *a, void *b)

  This method should be used when two items `a` and `b` are being swapped.
- void * **top** () const

  Returns the item that appears at the top of the list.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()

  Returns the internal array of widget sizes and positions.
- void draw () override

  Draws the widget.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget *, int)

  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)

  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)

  Allow derived groups to act when a child widget is removed from the group.
- int * sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Public Types inherited from Fl_Browser_

- enum {
  HORIZONTAL = 1 , VERTICAL = 2 , BOTH = 3 , ALWAYS_ON = 4 ,
  HORIZONTAL_ALWAYS = 5 , VERTICAL_ALWAYS = 6 , BOTH_ALWAYS = 7 }

  Values for has_scrollbar().

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()

  Returns the currently active group.
- static void current (Fl_Group *g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` *contains the entered '&x' shortcut.*

Public Attributes inherited from Fl_Browser_

- Fl_Scrollbar hscrollbar

  Horizontal scrollbar.
- Fl_Scrollbar scrollbar

  Vertical scrollbar.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.21.1 Detailed Description

The Fl_Check_Browser widget displays a scrolling list of text lines that may be selected and/or checked by the user.

13.21.2 Member Function Documentation

13.21.2.1 add() [1/2]

```
int Fl_Check_Browser::add (
            char * s )
```

Add a new unchecked line to the end of the browser.

See also

add(char *s, int b)

13.21.2.2 add() [2/2]

```
int Fl_Check_Browser::add (
            char * s,
            int b )
```

Add a new line to the end of the browser.
The text is copied using the strdup() function. It may also be NULL to make a blank line. It can set the item checked if `b` is not 0.

13.21.2.3 handle()

```
int Fl_Check_Browser::handle (
            int event ) [override], [protected], [virtual]
```

Handles the `event` within the normal widget bounding box.

Parameters

in	*event*	The event to process.

Returns

1 if event was processed, 0 if not.

Reimplemented from Fl_Browser_.

13.21.2.4 item_at()

```
void * Fl_Check_Browser::item_at (
            int index ) const [override], [virtual]
```

This method must be provided by the subclass to return the item for the specified `index`.

Parameters

in	*index*	The `index` of the item to be returned

Returns

The item at the specified `index`.

Reimplemented from Fl_Browser_.

13.21.2.5 item_draw()

```
void Fl_Check_Browser::item_draw (
            void * item,
            int X,
            int Y,
            int W,
            int H ) const [override], [protected], [virtual]
```

This method must be provided by the subclass to draw the `item` in the area indicated by `X`, `Y`, `W`, `H`.
Implements Fl_Browser_.

13.21.2.6 item_first()

```
void * Fl_Check_Browser::item_first ( ) const [override], [protected], [virtual]
```

This method must be provided by the subclass to return the first item in the list.

See also

item_first(), item_next(), item_last(), item_prev()

Implements Fl_Browser_.

13.21.2.7 item_height()

```
int Fl_Check_Browser::item_height (
            void * item ) const [override], [protected], [virtual]
```

This method must be provided by the subclass to return the height of `item` in pixels.
Allow for two additional pixels for the list selection box.

Parameters

in	*item*	The item whose height is returned.

Returns

The height of the specified `item` in pixels.

See also

item_height(), item_width(), item_quick_height()

Implements Fl_Browser_.

13.21.2.8 item_next()

```
void * Fl_Check_Browser::item_next (
            void * item ) const  [override], [protected], [virtual]
```

This method must be provided by the subclass to return the item in the list after `item`.

See also

item_first(), item_next(), item_last(), item_prev()

Implements Fl_Browser_.

13.21.2.9 item_prev()

```
void * Fl_Check_Browser::item_prev (
            void * item ) const  [override], [protected], [virtual]
```

This method must be provided by the subclass to return the item in the list before `item`.

See also

item_first(), item_next(), item_last(), item_prev()

Implements Fl_Browser_.

13.21.2.10 item_select()

```
void Fl_Check_Browser::item_select (
            void * item,
            int val )  [override], [protected], [virtual]
```

This method must be implemented by the subclass if it supports multiple selections; sets the selection state to `val` for the `item`.
Sets the selection state for `item`, where optional `val` is 1 (select, the default) or 0 (de-select).

Parameters

in	*item*	The item to be selected
in	*val*	The optional selection state; 1=select, 0=de-select. The default is to select the item (1).

Reimplemented from Fl_Browser_.

13.21.2.11 item_selected()

```
int Fl_Check_Browser::item_selected (
            void * item ) const  [override], [protected], [virtual]
```

This method must be implemented by the subclass if it supports multiple selections; returns the selection state for `item`.

The method should return 1 if `item` is selected, or 0 otherwise.

Parameters

in	*item*	The item to test.

Reimplemented from Fl_Browser_.

13.21.2.12 item_swap()

```
void Fl_Check_Browser::item_swap (
            void * a,
            void * b )  [override], [virtual]
```

This optional method should be provided by the subclass to efficiently swap browser items `a` and `b`, such as for sorting.

Parameters

in	*a,b*	The two items to be swapped.

Reimplemented from Fl_Browser_.

13.21.2.13 item_text()

```
const char * Fl_Check_Browser::item_text (
            void * item ) const  [override], [protected], [virtual]
```

This optional method returns a string (label) that may be used for sorting.

Parameters

in	*item*	The item whose label text is returned.

Returns

The item's text label. (Can be NULL if blank)

Reimplemented from Fl_Browser_.

13.21.2.14 item_width()

```
int Fl_Check_Browser::item_width (
            void * item ) const  [override], [protected], [virtual]
```

This method must be provided by the subclass to return the width of the `item` in pixels.
Allow for two additional pixels for the list selection box.

Parameters

in	*item*	The item whose width is returned.

Returns

The width of the item in pixels.

Implements Fl_Browser_.

13.21.2.15 nitems()

```
int Fl_Check_Browser::nitems ( ) const  [inline]
```

Returns how many lines are in the browser.

The last line number is equal to this.

13.21.2.16 remove()

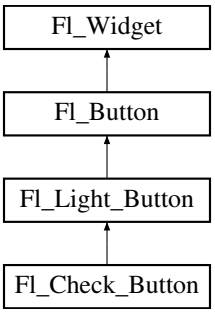
```
int Fl_Check_Browser::remove (
            int item )
```

Remove line n and make the browser one line shorter.
Returns the number of lines left in the browser.
The documentation for this class was generated from the following files:

- Fl_Check_Browser.H
- Fl_Check_Browser.cxx

13.22 Fl_Check_Button Class Reference

A button with a "checkmark" to show its status.
Inheritance diagram for Fl_Check_Button:



Public Member Functions

- Fl_Check_Button (int X, int Y, int W, int H, const char *L=0)

  Creates a new Fl_Check_Button widget using the given position, size, and label string.

Public Member Functions inherited from Fl_Light_Button

- Fl_Light_Button (int x, int y, int w, int h, const char *l=0)

  Creates a new Fl_Light_Button widget using the given position, size, and label string.
- int handle (int) override

  Handles the specified event.

Public Member Functions inherited from Fl_Button

- int clear ()

  Same as `value(0)`.
- uchar compact () const

  Return true if buttons are rendered as compact buttons.
- void compact (uchar v)

  Decide if buttons should be rendered in compact mode.
- Fl_Boxtype down_box () const

  Returns the current down box type, which is drawn when value() is non-zero.
- void down_box (Fl_Boxtype b)

  Sets the down box type.
- Fl_Color **down_color** () const

  (for backwards compatibility)

- void **down_color** (unsigned c)

  (for backwards compatibility)
- Fl_Button (int X, int Y, int W, int H, const char *L=0)

  The constructor creates the button using the given position, size, and label.
- int set ()

  Same as `value(1)`.
- void **setonly** ()

  Turns on this button and turns off all other radio buttons in the group (calling `value(1)` *or* `set()` *does not do this).*
- int shortcut () const

  Returns the current shortcut key for the button.
- void **shortcut** (const char *s)

  (for backwards compatibility)
- void shortcut (int s)

  Sets the shortcut key to `s`.
- char **value** () const

  Returns the current value of the button (0 or 1).
- int value (int v)

  Sets the current value of the button.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

Sets the image to use as part of the widget label when in the active state.

- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.

- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

Return the gap size between the label and the image.

- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.

- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

Gets the widget width.

- Fl_When when () const

  Returns the conditions under which the callback is called.

- void when (uchar i)

  Sets the flags used to decide when a callback is called.

- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.

- int x () const

  Gets the widget position in its window.

- int y () const

  Gets the widget position in its window.

- virtual ~Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.

- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.

- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Light_Button

- void draw () override

  Draws the widget.

Protected Member Functions inherited from Fl_Button

- void **simulate_key_action** ()

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Static Protected Member Functions inherited from Fl_Button

- static void **key_release_timeout** (void *)

Static Protected Attributes inherited from Fl_Button

- static Fl_Widget_Tracker * **key_release_tracker** = 0

13.22.1 Detailed Description

A button with a "checkmark" to show its status.

Figure 13.7 Fl_Check_Button

Buttons generate callbacks when they are clicked by the user. You control exactly when and how by changing the values for type() and when().
The Fl_Check_Button subclass displays its "ON" state by showing a "checkmark" rather than drawing itself pushed in.

13.22.2 Constructor & Destructor Documentation

13.22.2.1 Fl_Check_Button()

```
Fl_Check_Button::Fl_Check_Button (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

Creates a new Fl_Check_Button widget using the given position, size, and label string.
The default box type is FL_NO_BOX, which draws the label w/o a box right of the checkmark.
The selection_color() sets the color of the checkmark. Default is FL_FOREGROUND_COLOR (usually black).
You can use down_box() to change the box type of the checkmark. Default is FL_DOWN_BOX.

Parameters

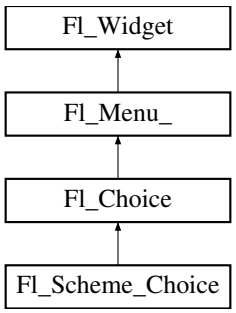
in	*X,Y,W,H*	position and size of the widget
in	*L*	widget label, default is no label

The documentation for this class was generated from the following files:

- Fl_Check_Button.H
- Fl_Check_Button.cxx

13.23 Fl_Choice Class Reference

A button that is used to pop up a menu.
`#include <Fl_Choice.H>`
Inheritance diagram for Fl_Choice:



Public Member Functions

- Fl_Choice (int X, int Y, int W, int H, const char ∗L=0)

  Create a new Fl_Choice widget using the given position, size and label string.
- int handle (int) override

  Handles the specified event.
- int value () const

  Gets the index of the last item chosen by the user.
- int value (const Fl_Menu_Item ∗v)

Sets the currently selected value using a pointer to menu item.

- int value (int v)

  Sets the currently selected value using the index into the menu item array.

Public Member Functions inherited from Fl_Menu_

- int add (const char *)

  This is a Forms (and SGI GL library) compatible add function, it adds many menu items, with '|' separating the menu items, and tab separating the menu item names from an optional shortcut string.
- int add (const char *, int shortcut, Fl_Callback *, void *=0, int=0)

  Adds a new menu item.
- int **add** (const char *a, const char *b, Fl_Callback *c, void *d=0, int e=0)

  See int Fl_Menu_::add(const char label, int shortcut, Fl_Callback*, void *user_data=0, int flags=0)*
- void clear ()

  Same as menu(NULL), set the array pointer to null, indicating a zero-length menu.
- int clear_submenu (int index)

  Clears the specified submenu pointed to by `index` of all menu items.
- void copy (const Fl_Menu_Item *m, void *user_data=0)

  Sets the menu array pointer with a copy of m that will be automatically deleted.
- Fl_Boxtype **down_box** () const

  This box type is used to surround the currently-selected items in the menus.
- void down_box (Fl_Boxtype b)

  Sets the box type used to surround the currently-selected items in the menus.
- Fl_Color **down_color** () const

  For back compatibility, same as selection_color()
- void **down_color** (unsigned c)

  For back compatibility, same as selection_color()
- int find_index (const char *name) const

  Find the menu item index for a given menu `pathname`, such as "Edit/Copy".
- int find_index (const Fl_Menu_Item *item) const

  Find the index into the menu array for a given `item`.
- int find_index (Fl_Callback *cb) const

  Find the index into the menu array for a given callback `cb`.
- const Fl_Menu_Item * find_item (const char *name)

  Find the menu item for a given menu `pathname`, such as "Edit/Copy".
- const Fl_Menu_Item * find_item (Fl_Callback *)

  Find the menu item for the given callback `cb`.
- const Fl_Menu_Item * find_item_with_argument (long)

  Find the menu item for the given user argument `v`.
- const Fl_Menu_Item * find_item_with_user_data (void *)

  Find the menu item for the given user data `v`.
- Fl_Menu_ (int, int, int, int, const char *=0)

  Creates a new Fl_Menu_ widget using the given position, size, and label string.
- void global ()

  Make the shortcuts for this menu work no matter what window has the focus when you type it.
- int insert (int index, const char *, int shortcut, Fl_Callback *, void *=0, int=0)

  Inserts a new menu item at the specified `index` position.
- int **insert** (int index, const char *a, const char *b, Fl_Callback *c, void *d=0, int e=0)

  See int Fl_Menu_::insert(const char label, int shortcut, Fl_Callback*, void *user_data=0, int flags=0)*
- int item_pathname (char *name, int namelen, const Fl_Menu_Item *finditem=0) const

  Get the menu 'pathname' for the specified menuitem.

- const Fl_Menu_Item * menu () const

  Returns a pointer to the array of Fl_Menu_Items.
- void menu (const Fl_Menu_Item *m)

  Sets the menu array pointer directly.
- Fl_Boxtype menu_box () const

  Get the box type for the menu popup windows.
- void menu_box (Fl_Boxtype b)

  Set the box type for the menu popup windows.
- const Fl_Menu_Item * menu_end ()

  Finishes menu modifications and returns menu().
- int mode (int i) const

  Get the flags of item i.
- void mode (int i, int fl)

  Set the flags of item i.
- const Fl_Menu_Item * mvalue () const

  Return a pointer to the last menu item that was picked.
- const Fl_Menu_Item * picked (const Fl_Menu_Item *)

  When user picks a menu item, call this.
- const Fl_Menu_Item * prev_mvalue () const

  Return a pointer to the menu item that was picked before the current one was picked.
- void remove (int)

  Deletes item `i` from the menu.
- void replace (int, const char *)

  Changes the text of item `i`.
- void **setonly** (Fl_Menu_Item *item)

  Turns the radio item "on" for the menu item and turns "off" adjacent radio items of the same group.
- void **shortcut** (int i, int s)

  Change the shortcut of item `i` to `s`.
- int size () const

  This returns the number of Fl_Menu_Item structures that make up the menu, correctly counting submenus.
- void **size** (int W, int H)
- const Fl_Menu_Item * test_shortcut ()

  Returns the menu item with the entered shortcut (key value).
- const char * text () const

  Returns the title of the last item chosen.
- const char * text (int i) const

  Returns the title of item i.
- Fl_Color textcolor () const

  Get the current color of menu item labels.
- void **textcolor** (Fl_Color c)

  Sets the current color of menu item labels.
- Fl_Font textfont () const

  Gets the current font of menu item labels.
- void textfont (Fl_Font c)

  Sets the current font of menu item labels.
- Fl_Fontsize textsize () const

  Gets the font size of menu item labels.
- void textsize (Fl_Fontsize c)

  Sets the font size of menu item labels.
- int value () const

  Return the index into the menu() of the last item chosen by the user.

- int value (const Fl_Menu_Item *)

  Set the value of a menu to the menu item `m`.
- int value (int i)

  Set the value of the menu to index `i`.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

Sets the current callback function and data for the widget.

- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.

- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

Sets the font size in pixels.

- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.

- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void draw () override

  Draws the widget.

Protected Member Functions inherited from Fl_Menu_

- int **item_pathname_** (char *name, int namelen, const Fl_Menu_Item *finditem, const Fl_Menu_Item *menu=0) const

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Attributes inherited from Fl_Menu_

- uchar **alloc**
- uchar **down_box_**
- Fl_Boxtype **menu_box_**
- Fl_Color **textcolor_**
- Fl_Font **textfont_**
- Fl_Fontsize **textsize_**

13.23.1 Detailed Description

A button that is used to pop up a menu.

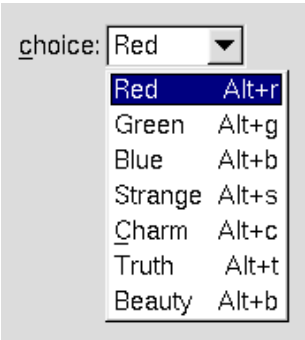


Figure 13.8 Fl_Choice

This is a button that, when pushed, pops up a menu (or hierarchy of menus) defined by an array of Fl_Menu_Item objects. Motif calls this an OptionButton.

The only difference between this and a Fl_Menu_Button is that the name of the most recent chosen menu item is displayed inside the box, while the label is displayed outside the box. However, since the use of this is most often to control a single variable rather than do individual callbacks, some of the Fl_Menu_Button methods are redescribed here in those terms.
When the user clicks a menu item, value() is set to that item and then:

```
- The item's callback is done if one has been set; the
  Fl_Choice is passed as the Fl_Widget* argument,
  along with any userdata configured for the callback.

- If the item does not have a callback, the Fl_Choice widget's
  callback is done instead, along with any userdata configured
  for it.  The callback can determine which item was picked using
  value(), mvalue(), item_pathname(), etc.
```

All three mouse buttons pop up the menu. The Forms behavior of the first two buttons to increment/decrement the choice is not implemented. This could be added with a subclass, however.
The menu will also pop up in response to shortcuts indicated by putting a '&' character in the label(). See Fl_Button::shortcut(int s) for a description of this.
Typing the shortcut() of any of the items will do exactly the same as when you pick the item with the mouse. The '&' character in item names are only looked at when the menu is popped up, however.
The inherited Fl_Widget::changed() and related methods can be used as follows:

- `int Fl_Widget::changed() const` This value is true when the user picks a different value. *It is turned off by value() and just before doing a callback (the callback can turn it back on if desired).*
- `void Fl_Widget::set_changed()` This method sets the changed() flag.
- `void Fl_Widget::clear_changed()` This method clears the changed() flag.

The inherited Fl_Menu_::down_box() methods can be used as follows:

- `Fl_Boxtype Fl_Menu_::down_box() const` Gets the current down box, which is used when the menu is popped up. The default down box type is `FL_DOWN_BOX`.
- `void Fl_Menu_::down_box(Fl_Boxtype b)` Sets the current down box type to `b`.

Simple example:

```
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Choice.H>
int main() {
  Fl_Window *win = new Fl_Window(300,200);
  Fl_Choice *choice = new Fl_Choice(100,10,100,25,"Choice:");
  choice->add("Zero");
  choice->add("One");
  choice->add("Two");
  choice->add("Three");
  choice->value(2);    // make "Two" selected by default (zero based!)
  win->end();
  win->show();
  return Fl::run();
}
```

FLTK triggers an `FL_BEFORE_MENU` event for this widget right before displaying the menu. This event provides an opportunity to update menu item states and activation. See `test/menubar.cxx`, class `Dynamic_Choice` "Flip" and "Flop" for a usage example.

13.23.2 Constructor & Destructor Documentation

13.23.2.1 Fl_Choice()

```
Fl_Choice::Fl_Choice (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

Create a new Fl_Choice widget using the given position, size and label string.
The default boxtype is `FL_UP_BOX`.
The constructor sets menu() to NULL. See Fl_Menu_ for the methods to set or change the menu.

Parameters

in	*X,Y,W,H*	position and size of the widget
in	*L*	widget label, default is no label

13.23.3 Member Function Documentation

13.23.3.1 draw()

```
void Fl_Choice::draw ( )  [override], [protected], [virtual]
```

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();                 // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

13.23.3.2 handle()

```
int Fl_Choice::handle (
            int event )  [override], [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to *override* the behavior of the base class. This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to *filter* event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you *consumed* the event by returning 1 even if you didn't *do* anything with it.

Parameters

in	*event*	the kind of event received

Return values

0	if the event was not used or understood
1	if the event was used and can be deleted

See also

Fl_Event

Reimplemented from Fl_Widget.
Reimplemented in Fl_Scheme_Choice.

13.23.3.3 value() [1/3]

```
int Fl_Choice::value ( ) const  [inline]
```

Gets the index of the last item chosen by the user.
The index is -1 initially.

13.23.3.4 value() [2/3]

```
int Fl_Choice::value (
            const Fl_Menu_Item * v )
```

Sets the currently selected value using a pointer to menu item.
Changing the selected value causes a redraw().

Parameters

in	*v*	pointer to menu item in the menu item array.

Returns

non-zero if the new value is different to the old one.

13.23.3.5 value() [3/3]

```
int Fl_Choice::value (
            int v )
```

Sets the currently selected value using the index into the menu item array.
Changing the selected value causes a redraw().

Parameters

in	*v*	index of value in the menu item array.

Returns

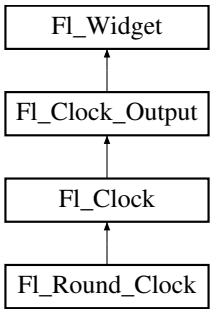
non-zero if the new value is different to the old one.

The documentation for this class was generated from the following files:

- Fl_Choice.H
- Fl_Choice.cxx

13.24 Fl_Clock Class Reference

This widget provides a round analog clock display.
`#include <Fl_Clock.H>`
Inheritance diagram for Fl_Clock:



Public Member Functions

- Fl_Clock (int X, int Y, int W, int H, const char ∗L=0)

  Create an Fl_Clock widget using the given position, size, and label string.
- Fl_Clock (uchar t, int X, int Y, int W, int H, const char ∗L)

  Create an Fl_Clock widget using the given clock type `t`, position, size, and label string.
- int handle (int) override

  Handles the specified event.
- ∼**Fl_Clock** ()

  The destructor removes the clock.

Public Member Functions inherited from Fl_Clock_Output

- Fl_Clock_Output (int X, int Y, int W, int H, const char ∗L=0)

  Create a new Fl_Clock_Output widget with the given position, size and label.
- int hour () const

  Returns the displayed hour (0 to 23).
- int minute () const

  Returns the displayed minute (0 to 59).
- int second () const

  Returns the displayed second (0 to 60, 60=leap second).
- int shadow () const

  Returns the shadow drawing mode of the hands.
- void shadow (int mode)

  Sets the shadow drawing mode of the hands.
- ulong value () const

  Returns the displayed time.
- void value (int H, int m, int s)

  Set the displayed time.
- void value (ulong v)

  Set the displayed time.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

- virtual class Fl_Gl_Window const ∗ **as_gl_window** () const
- virtual Fl_Group ∗ as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const ∗ **as_group** () const
- virtual Fl_Window ∗ as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ **as_window** () const
- void bind_deimage (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗cb, void ∗p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.

- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

Returns whether the image is managed by the widget.

- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.

- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

Get the spacing between the label and the vertical edge of the widget.

- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Clock_Output

- void draw () override

  Draw clock with current position and size.
- void draw (int X, int Y, int W, int H)

  Draw clock with the given position and size.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

13.24.1 Detailed Description

This widget provides a round analog clock display.
Fl_Clock is provided for Forms compatibility. It installs a 1-second timeout callback using Fl::add_timeout(). You can choose the rounded or square type of the clock with type(). Please see Fl_Clock_Output widget for applicable values.

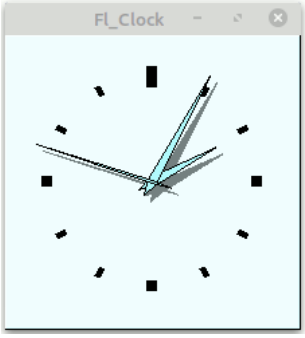


Figure 13.9 FL_SQUARE_CLOCK type

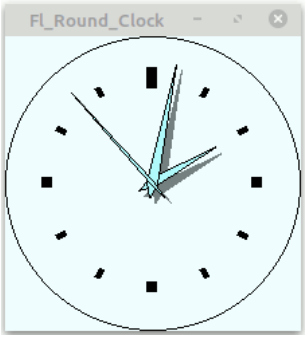


Figure 13.10 FL_ROUND_CLOCK type

See also

class Fl_Clock_Output

13.24.2 Constructor & Destructor Documentation

13.24.2.1 Fl_Clock() [1/2]

```
Fl_Clock::Fl_Clock (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

Create an Fl_Clock widget using the given position, size, and label string.
The default clock type is FL_SQUARE_CLOCK and the default boxtype is `FL_UP_BOX`.

Parameters

in	*X,Y,W,H*	position and size of the widget
in	*L*	widget label, default is no label

13.24.2.2 Fl_Clock() [2/2]

```
Fl_Clock::Fl_Clock (
            uchar t,
            int X,
            int Y,
            int W,
            int H,
```

```
const char * L )
```

Create an Fl_Clock widget using the given clock type `t`, position, size, and label string.
The default clock type `t` is `FL_SQUARE_CLOCK`. You can set the clock type to FL_ROUND_CLOCK or any other valid clock type. See Fl_Clock_Output widget for applicable values.
The default boxtype is `FL_UP_BOX` for `FL_SQUARE_CLOCK` and `FL_NO_BOX` for `FL_ROUND_CLOCK`, if set by the constructor. If you change the clock type with type() later you should also set the boxtype with box().

Parameters

in	*t*	type of clock: FL_ROUND_CLOCK or FL_SQUARE_CLOCK (0)
in	*X,Y,W,H*	position and size of the widget
in	*L*	widget label, default is no label

See also

class Fl_Clock_Output

13.24.3 Member Function Documentation

13.24.3.1 handle()

```
int Fl_Clock::handle (
            int event )  [override], [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to *override* the behavior of the base class. This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to *filter* event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you *consumed* the event by returning 1 even if you didn't *do* anything with it.

Parameters

in	*event*	the kind of event received

Return values

0	if the event was not used or understood
1	if the event was used and can be deleted

See also

Fl_Event

Reimplemented from Fl_Widget.
The documentation for this class was generated from the following files:

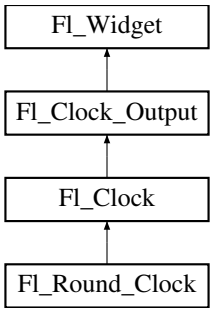
- Fl_Clock.H
- Fl_Clock.cxx

13.25 Fl_Clock_Output Class Reference

This widget can be used to display a program-supplied time.

```
#include <Fl_Clock.H>
```

Inheritance diagram for Fl_Clock_Output:



Public Member Functions

- Fl_Clock_Output (int X, int Y, int W, int H, const char ∗L=0)

  Create a new Fl_Clock_Output widget with the given position, size and label.
- int hour () const

  Returns the displayed hour (0 to 23).
- int minute () const

  Returns the displayed minute (0 to 59).
- int second () const

  Returns the displayed second (0 to 60, 60=leap second).
- int shadow () const

  Returns the shadow drawing mode of the hands.
- void shadow (int mode)

  Sets the shadow drawing mode of the hands.
- ulong value () const

  Returns the displayed time.
- void value (int H, int m, int s)

  Set the displayed time.
- void value (ulong v)

  Set the displayed time.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.

- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.

- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual int handle (int event)

  Handles the specified event.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

Sets the image to use as part of the widget label when in the active state.

- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.

- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

Sets the user data for this widget.

- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void draw () override

  Draw clock with current position and size.
- void draw (int X, int Y, int W, int H)

  Draw clock with the given position and size.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

Draws a focus rectangle around the widget.

- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.25.1 Detailed Description

This widget can be used to display a program-supplied time.
The time shown on the clock is not updated. To display the current time, use Fl_Clock instead.

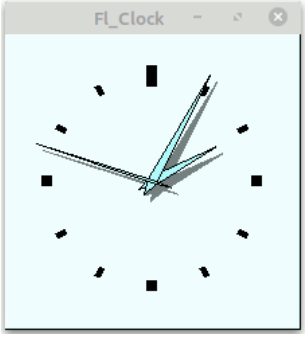


Figure 13.11 FL_SQUARE_CLOCK type

Figure 13.12 FL_ROUND_CLOCK type

Values for clock type() (#include <FL/Clock.H>):

```
#define FL_SQUARE_CLOCK              0       // Square Clock variant
#define FL_ROUND_CLOCK               1       // Round Clock variant
#define FL_ANALOG_CLOCK FL_SQUARE_CLOCK      // An analog clock is square
#define FL_DIGITAL_CLOCK FL_SQUARE_CLOCK     // Not yet implemented
```

13.25.2 Constructor & Destructor Documentation

13.25.2.1 Fl_Clock_Output()

```
Fl_Clock_Output::Fl_Clock_Output (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

Create a new Fl_Clock_Output widget with the given position, size and label.
The default clock type is `FL_SQUARE_CLOCK` and the default boxtype is `FL_UP_BOX`.

Parameters

in	*X,Y,W,H*	position and size of the widget
in	*L*	widget label, default is no label

13.25.3 Member Function Documentation

13.25.3.1 draw() [1/2]

```
void Fl_Clock_Output::draw (
            void ) [override], [protected], [virtual]
```

Draw clock with current position and size.
Implements Fl_Widget.

13.25.3.2 draw() [2/2]

```
void Fl_Clock_Output::draw (
            int X,
            int Y,
            int W,
            int H ) [protected]
```

Draw clock with the given position and size.

Parameters

in	*X,Y,W,H*	position and size

13.25.3.3 hour()

```
int Fl_Clock_Output::hour ( ) const [inline]
```

Returns the displayed hour (0 to 23).

See also

value(), minute(), second()

13.25.3.4 minute()

```
int Fl_Clock_Output::minute ( ) const [inline]
```

Returns the displayed minute (0 to 59).

See also

value(), hour(), second()

13.25.3.5 second()

```
int Fl_Clock_Output::second ( ) const [inline]
```

Returns the displayed second (0 to 60, 60=leap second).

See also

value(), hour(), minute()

13.25.3.6 shadow() [1/2]

```
int Fl_Clock_Output::shadow ( ) const [inline]
```

Returns the shadow drawing mode of the hands.

Returns

shadow drawing mode of the hands

Return values

0	no shadows
1	draw shadows of hands (default)

13.25.3.7 shadow() [2/2]

```
void Fl_Clock_Output::shadow (
            int mode )  [inline]
```

Sets the shadow drawing mode of the hands.
Enables (1) or disables (0) drawing the hands with shadows.
Values except 0 and 1 are reserved for future extensions and yield undefined behavior.
The default is to draw the shadows (1).

Parameters

in	*mode*	1 = shadows (default), 0 = no shadows

13.25.3.8 value() [1/3]

```
ulong Fl_Clock_Output::value ( ) const  [inline]
```

Returns the displayed time.
Returns the time in seconds since the UNIX epoch (January 1, 1970).

See also

value(ulong)

13.25.3.9 value() [2/3]

```
void Fl_Clock_Output::value (
            int H,
            int m,
            int s )
```

Set the displayed time.
Set the time in hours, minutes, and seconds.

Parameters

in	*H,m,s*	displayed time

See also

hour(), minute(), second()

13.25.3.10 value() [3/3]

```
void Fl_Clock_Output::value (
            ulong v )
```

Set the displayed time.
Set the time in seconds since the UNIX epoch (January 1, 1970).

Parameters

in	*v*	seconds since epoch

See also

value()

The documentation for this class was generated from the following files:

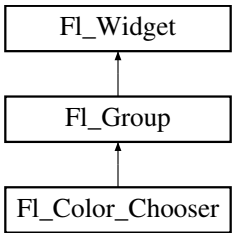
- Fl_Clock.H
- Fl_Clock.cxx

13.26 Fl_Color_Chooser Class Reference

The Fl_Color_Chooser widget provides a standard RGB color chooser.

`#include <Fl_Color_Chooser.H>`

Inheritance diagram for Fl_Color_Chooser:



Public Member Functions

- double b () const

  Returns the current blue value.
- Fl_Color_Chooser (int X, int Y, int W, int H, const char *L=0)

  Creates a new Fl_Color_Chooser widget using the given position, size, and label string.
- double g () const

  Returns the current green value.
- int handle (int e) override

  Handles all events received by this widget.
- int hsv (double H, double S, double V)

  Set the hsv values.
- double hue () const

  Returns the current hue.
- int mode () const

  Returns which Fl_Color_Chooser variant is currently active.
- void mode (int newMode)

  Set which Fl_Color_Chooser variant is currently active.
- double r () const

  Returns the current red value.
- int rgb (double R, double G, double B)

  Sets the current rgb color values.
- double saturation () const

  Returns the saturation.
- double value () const

  Returns the value/brightness.

Public Member Functions inherited from Fl_Group

- Fl_Widget *& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget *o)

  See void Fl_Group::add(Fl_Widget &w)
- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget *const * array () const

  Returns a pointer to the internal array of children.
- Fl_Group const * as_group () const override
- Fl_Group * as_group () override

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const

  Returns the n'th child.
- int **children** () const

  Returns how many child widgets the group has.
- void clear ()

  Deletes all child widgets from memory recursively.
- unsigned int clip_children () const

  Returns the current clipping mode.
- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)

  Removes the widget at `index` from the group and deletes it.
- void end ()

  Exactly the same as current(this->parent()).
- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*
- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.
- void **forms_end** ()

  This is for forms compatibility only.
- int handle (int) override

  Handles the specified event.
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

  This does insert(w, find(before)).
- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.

- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.
- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- void resize (int, int, int, int) override

  Resizes the Fl_Group widget and all of its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.

- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

Sets the image to use as part of the widget label when in the inactive state.

- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.

- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

Changes the size of the widget.

- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Static Public Member Functions

- static void hsv2rgb (double H, double S, double V, double &R, double &G, double &B)

  This static method converts HSV colors to RGB colorspace.
- static void rgb2hsv (double R, double G, double B, double &H, double &S, double &V)

  This static method converts RGB colors to HSV colorspace.

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()

  Returns the currently active group.
- static void current (Fl_Group *g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Related Symbols

(Note that these are not member symbols.)

- int fl_color_chooser (const char *name, double &r, double &g, double &b, int cmode)

  Pops up a window to let the user pick an arbitrary RGB color.
- int fl_color_chooser (const char *name, uchar &r, uchar &g, uchar &b, int cmode)

  Pops up a window to let the user pick an arbitrary RGB color.

Additional Inherited Members

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Group

- Fl_Rect ∗ bounds ()

  Returns the internal array of widget sizes and positions.
- void draw () override

  Draws the widget.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget ∗, int)

  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)

  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)

  Allow derived groups to act when a child widget is removed from the group.
- int ∗ sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

Sets a flag in the flags mask.

- void w (int v)

  Internal use only.

- void x (int v)

  Internal use only.

- void y (int v)

  Internal use only.

13.26.1 Detailed Description

The Fl_Color_Chooser widget provides a standard RGB color chooser.

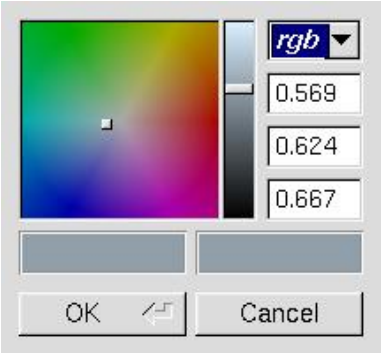


Figure 13.13 fl_color_chooser()

You can place any number of the widgets into a panel of your own design. The diagram shows the widget as part of a color chooser dialog created by the fl_color_chooser() function. The Fl_Color_Chooser widget contains the hue box, value slider, and rgb input fields from the above diagram (it does not have the color chips or the Cancel or OK buttons). The callback is done every time the user changes the rgb value. It is not done if they move the hue control in a way that produces the *same* rgb value, such as when saturation or value is zero.
The fl_color_chooser() function pops up a window to let the user pick an arbitrary RGB color. They can pick the hue and saturation in the "hue box" on the left (hold down CTRL to just change the saturation), and the brightness using the vertical slider. Or they can type the 8-bit numbers into the RGB Fl_Value_Input fields, or drag the mouse across them to adjust them. The pull-down menu lets the user set the input fields to show RGB, HSV, or 8-bit RGB (0 to 255).
The user can press CTRL-C to copy the currently selected color value as text in RGB hex format with leading zeroes to the clipboard, for instance `FL_GREEN` would be '00FF00' (since FLTK 1.4.0).
fl_color_chooser() returns non-zero if the user picks ok, and updates the RGB values. If the user picks cancel or closes the window this returns zero and leaves RGB unchanged.
If you use the color chooser on an 8-bit screen, it will allocate all the available colors, leaving you no space to exactly represent the color the user picks! You can however use fl_rectf() to fill a region with a simulated color using dithering.
Callback reasons can be `FL_REASON_DRAGGED`, `FL_REASON_CHANGED`, or `FL_REASON_RESELECTED`.

13.26.2 Constructor & Destructor Documentation

13.26.2.1 Fl_Color_Chooser()

```
Fl_Color_Chooser::Fl_Color_Chooser (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

Creates a new Fl_Color_Chooser widget using the given position, size, and label string.
The recommended dimensions are 200x95. The color is initialized to black.

Parameters

in	*X,Y,W,H*	position and size of the widget
in	*L*	widget label, default is no label

13.26.3 Member Function Documentation

13.26.3.1 b()

```
double Fl_Color_Chooser::b ( ) const  [inline]
```

Returns the current blue value.
0 <= b <= 1.

13.26.3.2 g()

```
double Fl_Color_Chooser::g ( ) const  [inline]
```

Returns the current green value.
0 <= g <= 1.

13.26.3.3 handle()

```
int Fl_Color_Chooser::handle (
            int e )  [override], [virtual]
```

Handles all events received by this widget.
This specific handle() method processes the standard 'copy' function as seen in other input widgets. It copies the current color value to the clipboard as a string in RGB format ('RRGGBB').
This format is independent of the Fl_Color_Chooser display format setting. No other formats are supplied.
The keyboard events handled are:

- ctrl-c
- ctrl-x
- ctrl-Insert

All other events are processed by the parent class `Fl_Group`.
This enables the **user** to choose a color value, press `ctrl-c` to copy the value to the clipboard and paste it into a color selection widget in another application window or any other text input (e.g. a preferences dialog or an editor).

Note

Keyboard event handling by the current focus widget has priority, hence moving the focus to one of the buttons or selecting text in one of the input widgets effectively disables this special method.

Parameters

in	*e*	current event

Returns

1 if event has been handled, 0 otherwise

See also

Fl_Group::handle(int)

Reimplemented from Fl_Widget.

13.26.3.4 hsv()

```
int Fl_Color_Chooser::hsv (
            double H,
            double S,
            double V )
```

Set the hsv values.
The passed values are clamped (or for hue, modulus 6 is used) to get legal values. Does not do the callback.

Parameters

in	*H,S,V*	color components.

Returns

1 if a new hsv value was set, 0 if the hsv value was the previous one.

13.26.3.5 hsv2rgb()

```
void Fl_Color_Chooser::hsv2rgb (
            double H,
            double S,
            double V,
            double & R,
            double & G,
            double & B )  [static]
```

This *static* method converts HSV colors to RGB colorspace.

Parameters

in	*H,S,V*	color components
out	*R,G,B*	color components

13.26.3.6 hue()

```
double Fl_Color_Chooser::hue ( ) const  [inline]
```

Returns the current hue.
0 <= hue < 6. Zero is red, one is yellow, two is green, etc. *This value is convenient for the internal calculations - some other systems consider hue to run from zero to one, or from 0 to 360.*

13.26.3.7 mode() [1/2]

```
int Fl_Color_Chooser::mode ( ) const  [inline]
```

Returns which Fl_Color_Chooser variant is currently active.

Returns

color modes are rgb(0), byte(1), hex(2), or hsv(3)

13.26.3.8 mode() [2/2]

```
void Fl_Color_Chooser::mode (
            int newMode )
```

Set which Fl_Color_Chooser variant is currently active.

Parameters

in	*newMode*	color modes are rgb(0), byte(1), hex(2), or hsv(3)

13.26.3.9 r()

```
double Fl_Color_Chooser::r ( ) const  [inline]
```

Returns the current red value.
0 <= r <= 1.

13.26.3.10 rgb()

```
int Fl_Color_Chooser::rgb (
            double R,
            double G,
            double B )
```

Sets the current rgb color values.
Does not do the callback. Does not clamp (but out of range values will produce psychedelic effects in the hue selector).

Parameters

in	*R,G,B*	color components.

Returns

1 if a new rgb value was set, 0 if the rgb value was the previous one.

13.26.3.11 rgb2hsv()

```
void Fl_Color_Chooser::rgb2hsv (
            double R,
            double G,
            double B,
            double & H,
            double & S,
            double & V )  [static]
```

This *static* method converts RGB colors to HSV colorspace.

Parameters

in	*R,G,B*	color components
out	*H,S,V*	color components

13.26.3.12 saturation()

```
double Fl_Color_Chooser::saturation ( ) const  [inline]
```

Returns the saturation.
0 <= saturation <= 1.

13.26.3.13 value()

```
double Fl_Color_Chooser::value ( ) const  [inline]
```

Returns the value/brightness.
0 <= value <= 1.

The documentation for this class was generated from the following files:

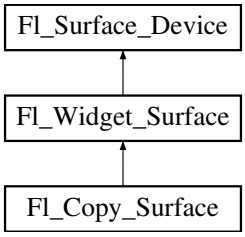
- Fl_Color_Chooser.H
- Fl_Color_Chooser.cxx

13.27 Fl_Copy_Surface Class Reference

Supports copying of graphical data to the clipboard.

`#include <Fl_Copy_Surface.H>`

Inheritance diagram for Fl_Copy_Surface:



Public Member Functions

- Fl_Copy_Surface (int w, int h)

  the constructor
- int **h** ()

  Returns the pixel height of the copy surface.
- bool is_current () override

  Is this surface the current drawing surface?
- void origin (int *x, int *y) override

  Computes the coordinates of the current origin of graphics functions.
- void origin (int x, int y) override

  Sets the position of the origin of graphics in the drawable part of the drawing surface.
- int printable_rect (int *w, int *h) override

  Computes the width and height of the drawable area of the drawing surface.
- void set_current () override

  Make this surface the current drawing surface.
- int **w** ()

  Returns the pixel width of the copy surface.
- ~**Fl_Copy_Surface** ()

  the destructor

Public Member Functions inherited from Fl_Widget_Surface

- void draw (Fl_Widget *widget, int delta_x=0, int delta_y=0)

  Draws the widget on the drawing surface.
- void draw_decorated_window (Fl_Window *win, int x_offset=0, int y_offset=0)

  Draws a window with its title bar and frame if any.
- void print_window_part (Fl_Window *win, int x, int y, int w, int h, int delta_x=0, int delta_y=0)

  Draws a rectangular part of an on-screen window.

Public Member Functions inherited from Fl_Surface_Device

- Fl_Graphics_Driver * **driver** ()

  Returns the graphics driver of this drawing surface.
- virtual ~**Fl_Surface_Device** ()

  The destructor.

Protected Member Functions

- void translate (int x, int y) override

  Translates the current graphics origin accounting for the current rotation.
- void untranslate () override

  Undoes the effect of a previous translate() call.

Protected Member Functions inherited from Fl_Widget_Surface

- Fl_Widget_Surface (Fl_Graphics_Driver *d)

  The constructor.

Protected Member Functions inherited from Fl_Surface_Device

- void **driver** (Fl_Graphics_Driver *graphics_driver)

  Sets the graphics driver of this drawing surface.
- virtual void end_current ()

  FLTK calls this each time a surface ceases to be the current drawing surface.
- **Fl_Surface_Device** (Fl_Graphics_Driver *graphics_driver)

  Constructor that sets the graphics driver to use for the created surface.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Surface_Device

- static Fl_Surface_Device * pop_current ()

  Removes the top element from the current drawing surface stack, and makes the new top element current.
- static void push_current (Fl_Surface_Device *new_current)

  Pushes `new_current` on top of the stack of current drawing surfaces, and makes it current.
- static Fl_Surface_Device * surface ()

  The current drawing surface.

Protected Attributes inherited from Fl_Widget_Surface

- int **x_offset**

  horizontal offset to the origin of graphics coordinates
- int **y_offset**

  vertical offset to the origin of graphics coordinates

13.27.1 Detailed Description

Supports copying of graphical data to the clipboard.

After creation of an Fl_Copy_Surface object, make it the current drawing surface calling Fl_Surface_Device::push_current(), and all subsequent graphics requests will be recorded in the clipboard. It's possible to draw widgets (using Fl_Copy_Surface::draw()) or to use any of the Drawing functions or the Color & Font functions. Finally, delete the Fl_Copy_Surface object to load the clipboard with the graphical data.
Fl_Gl_Window 's can be copied to the clipboard as well.
Usage example:

```
Fl_Widget *g = ...; // a widget you want to copy to the clipboard
Fl_Copy_Surface *copy_surf = new Fl_Copy_Surface(g->w(), g->h()); // create an Fl_Copy_Surface object
Fl_Surface_Device::push_current(copy_surf); // direct graphics requests to the clipboard
fl_color(FL_WHITE); fl_rectf(0, 0, g->w(), g->h()); // draw a white background
copy_surf->draw(g); // draw the g widget in the clipboard
Fl_Surface_Device::pop_current();  // direct graphics requests back to their previous destination
delete copy_surf; // after this, the clipboard is loaded
```

Platform details:

- Windows: Transparent RGB images copy without transparency. The graphical data are copied to the clipboard in two formats: 1) as an 'enhanced metafile'; 2) as a color bitmap. Applications to which the clipboard content is pasted can use the format that suits them best.
- Mac OS: The graphical data are copied to the clipboard (a.k.a. pasteboard) in two 'flavors': 1) in vectorial form as PDF data; 2) in bitmap form as a TIFF image. Applications to which the clipboard content is pasted can use the flavor that suits them best.
- X11 and Wayland: the graphical data are copied to the clipboard as an image in BMP format.

13.27.2 Constructor & Destructor Documentation

13.27.2.1 Fl_Copy_Surface()

```
Fl_Copy_Surface::Fl_Copy_Surface (
            int w,
            int h )
```

the constructor

Parameters

w,h	Width and height of the drawing surface in FLTK units

13.27.3 Member Function Documentation

13.27.3.1 is_current()

```
bool Fl_Copy_Surface::is_current ( )  [override], [virtual]
```

Is this surface the current drawing surface?
Reimplemented from Fl_Surface_Device.

13.27.3.2 origin() [1/2]

```
void Fl_Copy_Surface::origin (
            int * x,
            int * y )  [override], [virtual]
```

Computes the coordinates of the current origin of graphics functions.

Parameters

out	*x,y*	If non-null, ∗x and ∗y are set to the horizontal and vertical coordinates of the graphics origin.

Reimplemented from Fl_Widget_Surface.

13.27.3.3 origin() [2/2]

```
void Fl_Copy_Surface::origin (
            int x,
            int y )  [override], [virtual]
```

Sets the position of the origin of graphics in the drawable part of the drawing surface.
Arguments should be expressed relatively to the result of a previous printable_rect() call. That is, `printable_rect(&w, &h); origin(w/2, 0);` sets the graphics origin at the top center of the drawable area. Successive origin() calls don't combine their effects. Origin() calls are not affected by rotate() calls (for classes derived from Fl_Paged_Device).

Parameters

in	*x,y*	Horizontal and vertical positions in the drawing surface of the desired origin of graphics.

Reimplemented from Fl_Widget_Surface.

13.27.3.4 printable_rect()

```
int Fl_Copy_Surface::printable_rect (
            int * w,
            int * h )  [override], [virtual]
```

Computes the width and height of the drawable area of the drawing surface.
Values are in the same unit as that used by FLTK drawing functions and are unchanged by calls to origin(). If the object is derived from class Fl_Paged_Device, values account for the user-selected paper type and print orientation and are changed by scale() calls.

Returns

0 if OK, non-zero if any error

Reimplemented from Fl_Widget_Surface.

13.27.3.5 set_current()

```
void Fl_Copy_Surface::set_current (
            void  )  [override], [virtual]
```

Make this surface the current drawing surface.
This surface will receive all future graphics requests.
Since FLTK 1.4.0 the preferred API to change the current drawing surface is Fl_Surface_Device::push_current( ) / Fl_Surface_Device::pop_current().

Note

It is recommended to use this function only as follows :

- The current drawing surface is the display;
- make current another surface, e.g., an Fl_Printer or an Fl_Image_Surface object, calling set_current() on this object;
- draw to that surface;
- make the display current again with Fl_Display_Device::display_device()->set_current();
  don't do any other call to set_current() before this one.

Other scenarios of drawing surface changes should be performed via Fl_Surface_Device::push_current() and Fl_Surface_Device::pop_current().
Reimplemented from Fl_Surface_Device.

13.27.3.6 translate()

```
void Fl_Copy_Surface::translate (
            int x,
            int y )  [override], [protected], [virtual]
```

Translates the current graphics origin accounting for the current rotation.
Each translate() call must be matched by an untranslate() call. Successive translate() calls add up their effects.
Reimplemented from Fl_Widget_Surface.

13.27.3.7 untranslate()

```
void Fl_Copy_Surface::untranslate (
            void  )  [override], [protected], [virtual]
```

Undoes the effect of a previous translate() call.
Reimplemented from Fl_Widget_Surface.
The documentation for this class was generated from the following files:

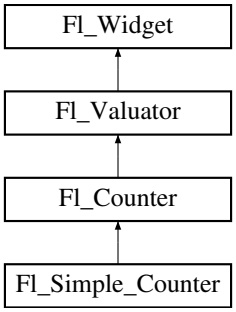
- Fl_Copy_Surface.H
- Fl_Copy_Surface.cxx

13.28 Fl_Counter Class Reference

Controls a single floating point value with button (or keyboard) arrows.

`#include <Fl_Counter.H>`

Inheritance diagram for Fl_Counter:



Public Member Functions

- Fl_Counter (int X, int Y, int W, int H, const char ∗L=0)

  Creates a new Fl_Counter widget using the given position, size, and label string.
- int handle (int) override

  Handles the specified event.
- void lstep (double a)

  Sets the increment for the large step buttons.
- double **step** () const

  Returns the increment for normal step buttons.
- void step (double a)

  Sets the increment for the normal step buttons.
- void step (double a, double b)

  Sets the increments for the normal and large step buttons.
- Fl_Color **textcolor** () const

  Gets the font color.
- void **textcolor** (Fl_Color s)

  Sets the font color to `s`.
- Fl_Font **textfont** () const

  Gets the text font.
- void **textfont** (Fl_Font s)

  Sets the text font to `s`.
- Fl_Fontsize **textsize** () const

  Gets the font size.
- void **textsize** (Fl_Fontsize s)

  Sets the font size to `s`.
- ∼**Fl_Counter** ()

  Destroys the valuator.

Public Member Functions inherited from Fl_Valuator

- void **bounds** (double a, double b)

  Sets the minimum (a) and maximum (b) values for the valuator widget.
- double **clamp** (double)

  Clamps the passed value to the valuator range.
- virtual int format (char ∗)

Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.

- virtual std::string format_str ()

  C++11 API for Fl_Valuator::format(char buffer).*
- double increment (double, int)

  Adds n times the step value to the passed value.
- double maximum () const

  Gets the maximum value for the valuator.
- void maximum (double a)

  Sets the maximum value for the valuator.
- double minimum () const

  Gets the minimum value for the valuator.
- void minimum (double a)

  Sets the minimum value for the valuator.
- void precision (int digits)

  Sets the step value to 1.0 / 10^{digits}.
- void range (double a, double b)

  Sets the minimum and maximum values for the valuator.
- double round (double)

  Round the passed value to the nearest step increment.
- double step () const

  Gets or sets the step value.
- void **step** (double a, int b)

  See double Fl_Valuator::step() const
- void **step** (double s)

  See double Fl_Valuator::step() const.
- void **step** (int a)

  See double Fl_Valuator::step() const
- double value () const

  Gets the floating point(double) value.
- int value (double)

  Sets the current value.
- ~**Fl_Valuator** () override

  Destructor is accessible despite protected constructor.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.

- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.

- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

Sets the image to use as part of the widget label when in the active state.

- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.

- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

Sets the user data for this widget.

- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void arrow_widths (int &w1, int &w2)

  Compute sizes (widths) of arrow boxes.
- void draw () override

  Draws the widget.

Protected Member Functions inherited from Fl_Valuator

- Fl_Valuator (int X, int Y, int W, int H, const char ∗L)

  Creates a new Fl_Valuator widget using the given position, size, and label string.
- void **handle_drag** (double newvalue)

  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.
- void **handle_push** ()

  Stores the current value in the previous value.
- void **handle_release** ()

  Called after an FL_WHEN_RELEASE event is received and before the callback.
- int **horizontal** () const

  Tells if the valuator is an FL_HORIZONTAL one.
- double **previous_value** () const

  Gets the previous floating point value before an event changed it.
- void **set_value** (double v)

Sets the current floating point value.

- double **softclamp** (double)

  Clamps the value, but accepts v if the previous value is not already out of range.
- virtual void value_damage ()

  Asks for partial redraw.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.28.1 Detailed Description

Controls a single floating point value with button (or keyboard) arrows.
Double arrows buttons achieve larger steps than simple arrows.

See also

Fl_Spinner for value input with vertical step arrows.

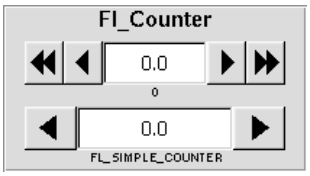


Figure 13.14 Fl_Counter

The type of an Fl_Counter object can be set using Fl_Widget::type(uchar) to:

- `FL_NORMAL_COUNTER`: Displays a counter with 4 arrow buttons.
- `FL_SIMPLE_COUNTER`: Displays a counter with only 2 arrow buttons.

13.28.2 Constructor & Destructor Documentation

13.28.2.1 Fl_Counter()

```
Fl_Counter::Fl_Counter (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

Creates a new Fl_Counter widget using the given position, size, and label string.
The default type is FL_NORMAL_COUNTER.

Parameters

in	*X,Y,W,H*	position and size of the widget
in	*L*	widget label, default is no label

13.28.3 Member Function Documentation

13.28.3.1 arrow_widths()

```
void Fl_Counter::arrow_widths (
            int & w1,
            int & w2 )  [protected]
```

Compute sizes (widths) of arrow boxes.

This method computes the two sizes of the arrow boxes of Fl_Counter. You can override it in a subclass if you want to draw fancy arrows or change the layout. However, the basic layout is fixed and can't be changed w/o overriding the draw() and handle() methods.

Basic layout:

```
+------+-----+-------------+-----+------+
|  «   |  <  |    value    |  >  |  »   |
+------+-----+-------------+-----+------+
```

The returned value `w2` should be zero if the counter type() is FL_SIMPLE_COUNTER.

Parameters

out	*w1*	width of single arrow box
out	*w2*	width of double arrow box

13.28.3.2 draw()

```
void Fl_Counter::draw ( )  [override], [protected], [virtual]
```

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();                 // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

13.28.3.3 handle()

```
int Fl_Counter::handle (
            int event )  [override], [virtual]
```

Handles the specified event.

You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.

When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.

Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

One exception to the rule in the previous paragraph is if you really want to *override* the behavior of the base class. This requires knowledge of the details of the inherited class.

In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to *filter* event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you *consumed* the event by returning 1 even if you didn't *do* anything with it.

Parameters

in	*event*	the kind of event received

Return values

0	if the event was not used or understood

Return values

1	if the event was used and can be deleted

See also

Fl_Event

Reimplemented from Fl_Widget.

13.28.3.4 lstep()

```
void Fl_Counter::lstep (
            double a )  [inline]
```

Sets the increment for the large step buttons.
The default value is 1.0.

Parameters

in	*a*	large step increment.

13.28.3.5 step() [1/2]

```
void Fl_Counter::step (
            double a )  [inline]
```

Sets the increment for the normal step buttons.

Parameters

in	*a*	normal step increment.

13.28.3.6 step() [2/2]

```
void Fl_Counter::step (
            double a,
            double b )  [inline]
```

Sets the increments for the normal and large step buttons.

Parameters

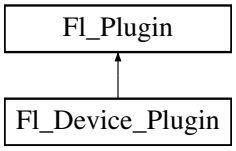
in	*a,b*	normal and large step increments.

The documentation for this class was generated from the following files:

- Fl_Counter.H
- Fl_Counter.cxx

13.29 Fl_Device_Plugin Class Reference

This plugin socket allows the integration of new device drivers for special window or screen types.
`#include <Fl_Device.H>`
Inheritance diagram for Fl_Device_Plugin:



Public Member Functions

- virtual void **delete_image_texture** (Fl_Image ∗)=0
- **Fl_Device_Plugin** (const char ∗pluginName)

  The constructor.
- virtual const char ∗ **klass** ()

  Returns the class name.
- virtual const char ∗ **name** ()=0

  Returns the plugin name.
- virtual int **print** (Fl_Widget ∗w)=0

  Prints a widget.
- virtual Fl_RGB_Image ∗ rectangle_capture (Fl_Widget ∗widget, int x, int y, int w, int h)=0

  Captures a rectangle of a widget as an image.

Public Member Functions inherited from Fl_Plugin

- Fl_Plugin (const char ∗klass, const char ∗name)

  Create a plugin.
- virtual ∼**Fl_Plugin** ()

  Clear the plugin and remove it from the database.

Static Public Member Functions

- static Fl_Device_Plugin ∗ **opengl_plugin** ()

  Returns the OpenGL plugin.

13.29.1 Detailed Description

This plugin socket allows the integration of new device drivers for special window or screen types.
This class is not intended for use outside the FLTK library. It is currently used to provide an automated printing service and screen capture for OpenGL windows, if linked with fltk_gl.

13.29.2 Member Function Documentation

13.29.2.1 rectangle_capture()

```
virtual Fl_RGB_Image * Fl_Device_Plugin::rectangle_capture (
            Fl_Widget * widget,
            int x,
            int y,
            int w,
            int h )  [pure virtual]
```

Captures a rectangle of a widget as an image.

Returns

The captured pixels as an RGB image

The documentation for this class was generated from the following files:

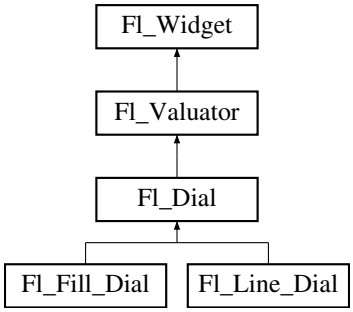
- Fl_Device.H
- Fl_Device.cxx

13.30 Fl_Dial Class Reference

The Fl_Dial widget provides a circular dial to control a single floating point value.

`#include <Fl_Dial.H>`

Inheritance diagram for Fl_Dial:



Public Member Functions

- short angle1 () const

  Sets Or gets the angles used for the minimum and maximum values.
- void **angle1** (short a)

  See short angle1() const.
- short **angle2** () const

  See short angle1() const.
- void **angle2** (short a)

  See short angle1() const.
- void **angles** (short a, short b)

  See short angle1() const.
- Fl_Dial (int x, int y, int w, int h, const char *l=0)

  Creates a new Fl_Dial widget using the given position, size, and label string.
- int handle (int) override

  Allow subclasses to handle event based on current position and size.

Public Member Functions inherited from Fl_Valuator

- void **bounds** (double a, double b)

  Sets the minimum (a) and maximum (b) values for the valuator widget.
- double **clamp** (double)

  Clamps the passed value to the valuator range.
- virtual int format (char *)

  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.
- virtual std::string format_str ()

  C++11 API for Fl_Valuator::format(char buffer).*
- double increment (double, int)

  Adds n times the step value to the passed value.
- double maximum () const

  Gets the maximum value for the valuator.
- void maximum (double a)

  Sets the maximum value for the valuator.
- double minimum () const

  Gets the minimum value for the valuator.
- void minimum (double a)

Sets the minimum value for the valuator.

- void precision (int digits)

  Sets the step value to 1.0 / 10^{digits}.
- void range (double a, double b)

  Sets the minimum and maximum values for the valuator.
- double round (double)

  Round the passed value to the nearest step increment.
- double step () const

  Gets or sets the step value.
- void **step** (double a, int b)

  See double Fl_Valuator::step() const
- void **step** (double s)

  See double Fl_Valuator::step() const.
- void **step** (int a)

  See double Fl_Valuator::step() const
- double value () const

  Gets the floating point(double) value.
- int value (double)

  Sets the current value.
- ~**Fl_Valuator** () override

  Destructor is accessible despite protected constructor.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

Sets the image to use as part of the widget label when in the inactive state.

- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.

- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

Sets the current label pointer.

- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.

- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

Modifies keyboard focus navigation.

- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void draw () override

  Draws dial at current position and size.
- void draw (int X, int Y, int W, int H)

  Draws dial at given position and size.
- int handle (int event, int X, int Y, int W, int H)

  Allows subclasses to handle event based on given position and size.

Protected Member Functions inherited from Fl_Valuator

- Fl_Valuator (int X, int Y, int W, int H, const char ∗L)

  Creates a new Fl_Valuator widget using the given position, size, and label string.
- void **handle_drag** (double newvalue)

  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.
- void **handle_push** ()

  Stores the current value in the previous value.
- void **handle_release** ()

  Called after an FL_WHEN_RELEASE event is received and before the callback.
- int **horizontal** () const

  Tells if the valuator is an FL_HORIZONTAL one.
- double **previous_value** () const

  Gets the previous floating point value before an event changed it.
- void **set_value** (double v)

  Sets the current floating point value.
- double **softclamp** (double)

  Clamps the value, but accepts v if the previous value is not already out of range.
- virtual void value_damage ()

  Asks for partial redraw.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 , AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.30.1 Detailed Description

The Fl_Dial widget provides a circular dial to control a single floating point value.

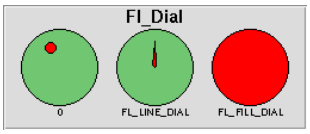


Figure 13.15 Fl_Dial

Use type() to set the type of the dial to:

- FL_NORMAL_DIAL - Draws a normal dial with a knob.
- FL_LINE_DIAL - Draws a dial with a line.
- FL_FILL_DIAL - Draws a dial with a filled arc.

13.30.2 Constructor & Destructor Documentation

13.30.2.1 Fl_Dial()

```
Fl_Dial::Fl_Dial (
            int X,
            int Y,
            int W,
            int H,
            const char * l = 0 )
```

Creates a new Fl_Dial widget using the given position, size, and label string.
The default type is FL_NORMAL_DIAL.

13.30.3 Member Function Documentation

13.30.3.1 angle1()

```
short Fl_Dial::angle1 ( ) const  [inline]
```

Sets Or gets the angles used for the minimum and maximum values.
The default values are 45 and 315 (0 degrees is straight down and the angles progress clockwise). Normally angle1 is less than angle2, but if you reverse them the dial moves counter-clockwise.

13.30.3.2 draw() [1/2]

```
void Fl_Dial::draw (
            void ) [override], [protected], [virtual]
```

Draws dial at current position and size.
Implements Fl_Widget.

13.30.3.3 draw() [2/2]

```
void Fl_Dial::draw (
            int X,
            int Y,
            int W,
            int H ) [protected]
```

Draws dial at given position and size.

Parameters

in	*X,Y,W,H*	position and size

13.30.3.4 handle() [1/2]

```
int Fl_Dial::handle (
            int event,
            int X,
            int Y,
            int W,
            int H ) [protected]
```

Allows subclasses to handle event based on given position and size.

Parameters

in	*event,X,Y,W,H*	event to handle, related position and size.

13.30.3.5 handle() [2/2]

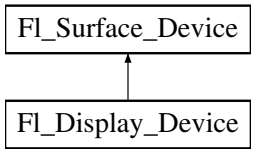
```
int Fl_Dial::handle (
            int e ) [override], [virtual]
```

Allow subclasses to handle event based on current position and size.
Reimplemented from Fl_Widget.
The documentation for this class was generated from the following files:

- Fl_Dial.H
- Fl_Dial.cxx

13.31 Fl_Display_Device Class Reference

The computer's display.
`#include <Fl_Device.H>`
Inheritance diagram for Fl_Display_Device:



Static Public Member Functions

- static Fl_Display_Device ∗ **display_device** ()

  Returns a pointer to the unique display device.

Static Public Member Functions inherited from Fl_Surface_Device

- static Fl_Surface_Device ∗ pop_current ()

  Removes the top element from the current drawing surface stack, and makes the new top element current.
- static void push_current (Fl_Surface_Device ∗new_current)

  Pushes `new_current` on top of the stack of current drawing surfaces, and makes it current.
- static Fl_Surface_Device ∗ surface ()

  The current drawing surface.

Additional Inherited Members

Public Member Functions inherited from Fl_Surface_Device

- Fl_Graphics_Driver ∗ **driver** ()

  Returns the graphics driver of this drawing surface.
- virtual bool is_current ()

  Is this surface the current drawing surface?
- virtual void set_current (void)

  Make this surface the current drawing surface.
- virtual ∼**Fl_Surface_Device** ()

  The destructor.

Protected Member Functions inherited from Fl_Surface_Device

- void **driver** (Fl_Graphics_Driver ∗graphics_driver)

  Sets the graphics driver of this drawing surface.
- virtual void end_current ()

  FLTK calls this each time a surface ceases to be the current drawing surface.
- **Fl_Surface_Device** (Fl_Graphics_Driver ∗graphics_driver)

  Constructor that sets the graphics driver to use for the created surface.

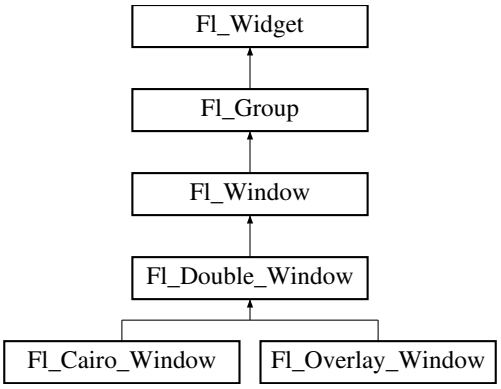
13.31.1 Detailed Description

The computer's display.
When FLTK begins to access the computer's display, it creates an object of class Fl_Display_Device and makes it the current drawing surface.
The documentation for this class was generated from the following files:

- Fl_Device.H
- Fl_Device.cxx

13.32 Fl_Double_Window Class Reference

The Fl_Double_Window provides a double-buffered window.
`#include <Fl_Double_Window.H>`
Inheritance diagram for Fl_Double_Window:



Public Member Functions

- Fl_Double_Window ∗ as_double_window () override

  Return non-null if this is an Fl_Double_Window object.
- **Fl_Double_Window** (int W, int H, const char ∗l=0)

  Creates a new Fl_Double_Window widget using the given position, size, and label (title) string.
- **Fl_Double_Window** (int X, int Y, int W, int H, const char ∗l=0)

  See Fl_Double_Window::Fl_Double_Window(int w, int h, const char ∗label = 0)
- void flush () override

  Forces the window to be drawn, this window is also made current and calls draw().
- void hide () override

  Makes a widget invisible.
- void resize (int, int, int, int) override

  Changes the size or position of the widget.
- void show () override

  Makes a widget visible.
- void **show** (int a, char ∗∗b)

  Same as Fl_Window::show(int a, char ∗∗b)
- ∼Fl_Double_Window ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Window

- void allow_expand_outside_parent ()

  Allow this subwindow to expand outside the area of its parent window.
- virtual class Fl_Overlay_Window ∗ as_overlay_window ()

  Return non-null if this is an Fl_Overlay_Window object.
- Fl_Window const ∗ as_window () const override
- Fl_Window ∗ as_window () override

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- unsigned int **border** () const

  Returns whether the window possesses a border.
- void border (int b)

  Sets whether or not the window manager border is around the window.
- void clear_border ()

  Fast inline function to turn the window manager border off.
- void clear_modal_states ()

  Clears the "modal" flags and converts a "modal" or "non-modal" window back into a "normal" window.
- void **copy_label** (const char ∗a)

Sets the window titlebar label to a copy of a character string.

- void cursor (const Fl_RGB_Image *, int, int)

  Changes the cursor for this window using the provided image as cursor's shape.
- void cursor (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)

  For back compatibility only.
- void cursor (Fl_Cursor)

  Changes the cursor for this window.
- int decorated_h () const

  Returns the window height including any window title bar and any frame added by the window manager.
- int decorated_w () const

  Returns the window width including any frame added by the window manager.
- void default_cursor (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)

  For back compatibility only.
- void default_cursor (Fl_Cursor)

  Sets the default window cursor.
- void **draw_backdrop** ()

  Draw the background image if one is set and is aligned inside.
- Fl_Window (int w, int h, const char *title=0)

  Creates a window from the given width `w`, height `h`, and `title`.
- Fl_Window (int x, int y, int w, int h, const char *title=0)

  Creates a window from the given position (x, y), size (w, h) and title.
- void free_position ()

  Undoes the effect of a previous resize() or show() so that the next time show() is called the window manager is free to position the window.
- void fullscreen ()

  Makes the window completely fill one or more screens, without any window manager border visible.
- unsigned int **fullscreen_active** () const

  Returns non zero if FULLSCREEN flag is set, 0 otherwise.
- void **fullscreen_off** ()

  Turns off any side effects of fullscreen()
- void **fullscreen_off** (int X, int Y, int W, int H)

  Turns off any side effects of fullscreen() and does resize(X,Y,W,H).
- void fullscreen_screens (int top, int bottom, int left, int right)

  Sets which screens should be used when this window is in fullscreen mode.
- uchar get_size_range (int *minw, int *minh, int *maxw=NULL, int *maxh=NULL, int *dw=NULL, int *dh=NULL, int *aspect=NULL) const

  Gets the allowable range to which the user can resize this window.
- int handle (int) override

  Handles the specified event.
- void hide () override

  Removes the window from the screen.
- void **hotspot** (const Fl_Widget &p, int offscreen=0)

  See void Fl_Window::hotspot(int x, int y, int offscreen = 0)
- void **hotspot** (const Fl_Widget *, int offscreen=0)

  See void Fl_Window::hotspot(int x, int y, int offscreen = 0)
- void hotspot (int x, int y, int offscreen=0)

  Positions the window so that the mouse is pointing at the given position, or at the center of the given widget, which may be the window itself.
- const void * icon () const

  Gets the current icon window target dependent data.
- void icon (const Fl_RGB_Image *)

Sets or resets a single window icon.

- void icon (const void *ic)

  Platform-specific method to set the window icon usable on Windows and X11 only.
- void iconize ()

  Iconifies the window.
- const char * **iconlabel** () const

  See void Fl_Window::iconlabel(const char)*
- void **iconlabel** (const char *)

  Sets the icon label.
- void icons (const Fl_RGB_Image *[], int)

  Sets the window icons.
- void icons (HICON big_icon, HICON small_icon)

  Sets the window icons using HICON handles (Windows platform only).
- const char * **label** () const

  See void Fl_Window::label(const char)*
- void **label** (const char *)

  Sets the window title bar label.
- void **label** (const char *label, const char *iconlabel)

  Sets the icon label.
- void make_current ()

  Sets things up so that the drawing functions in <FL/fl_draw.H> will go into this window.
- void maximize ()

  Maximizes a top-level window to its current screen.
- unsigned int **maximize_active** () const

  Returns whether the window is currently maximized.
- unsigned int **menu_window** () const

  Returns true if this window is a menu window.
- unsigned int modal () const

  Returns true if this window is modal.
- unsigned int **non_modal** () const

  Returns true if this window is modal or non-modal.
- fl_uintptr_t os_id () const

  Returns a platform-specific identification of a shown window, or 0 if not shown.
- unsigned int **override** () const

  Returns non zero if OVERRIDE flag is set, 0 otherwise.
- void resize (int X, int Y, int W, int H) override

  Changes the size and position of the window.
- int screen_num () const

  The number of the screen containing the mapped window.
- void screen_num (int screen_num)

  Set the number of the screen where to map the window.
- void set_menu_window ()

  Marks the window as a menu window.
- void set_modal ()

  A "modal" window, when shown(), will prevent any events from being delivered to other windows in the same program, and will also remain on top of the other windows (if the X window manager supports the "transient for" property).
- void set_non_modal ()

  A "non-modal" window (terminology borrowed from Microsoft Windows) acts like a modal() one in that it remains on top, but it has no effect on event delivery.

- void **set_override** ()

  Activates the flags NOBORDER|OVERRIDE.
- void set_tooltip_window ()

  Marks the window as a tooltip window.
- const Fl_Image ∗ shape () const

  Returns the image controlling the window shape or NULL.
- void shape (const Fl_Image &b)

  Set the window's shape with an Fl_Image.
- void shape (const Fl_Image ∗img)

  Assigns a non-rectangular shape to the window.
- void show () override

  Puts the window on the screen.
- void show (int argc, char ∗∗argv)

  Puts the window on the screen with show() and parses command-line arguments.
- int shown () const

  Returns non-zero if show() has been called (but not hide()).
- void size_range (int minw, int minh, int maxw=0, int maxh=0, int dw=0, int dh=0, int aspect=0)

  Sets the allowable range to which the user can resize this window.
- unsigned int **tooltip_window** () const

  Returns true if this window is a tooltip window.
- void un_maximize ()

  Returns a previously maximized top-level window to its previous size.
- void wait_for_expose ()

  Waits for the window to be displayed after calling show().
- int **x_root** () const

  Gets the x position of the window on the screen.
- const char ∗ xclass () const

  Returns the xclass for this window, or a default.
- void xclass (const char ∗c)

  Sets the xclass for this window.
- int **y_root** () const

  Gets the y position of the window on the screen.
- virtual ∼Fl_Window ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Group

- Fl_Widget ∗& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget ∗o)

  See void Fl_Group::add(Fl_Widget &w)
- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget ∗const ∗ array () const

  Returns a pointer to the internal array of children.
- Fl_Group const ∗ as_group () const override
- Fl_Group ∗ as_group () override

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()

Sets the current group so you can build the widget tree by just constructing the widgets.

- Fl_Widget * child (int n) const

  Returns the n'th child.

- int **children** () const

  Returns how many child widgets the group has.

- void clear ()

  Deletes all child widgets from memory recursively.

- unsigned int clip_children () const

  Returns the current clipping mode.

- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.

- virtual int delete_child (int n)

  Removes the widget at `index` from the group and deletes it.

- void end ()

  Exactly the same as current(this->parent()).

- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*

- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.

- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.

- void **forms_end** ()

  This is for forms compatibility only.

- void init_sizes ()

  Resets the internal array of widget sizes and positions.

- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.

- void insert (Fl_Widget &o, Fl_Widget *before)

  This does insert(w, find(before)).

- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.

- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.

- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.

- Fl_Widget * resizable () const

  Returns the group's resizable widget.

- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.

- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.

- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.

- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- int horizontal_label_margin ()

Get the spacing between the label and the horizontal edge of the widget.

- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.

- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Public Types inherited from Fl_Window

- typedef struct HICON__ ∗ **HICON**

Static Public Member Functions inherited from Fl_Window

- static Fl_Window ∗ current ()

  Returns the last window that was made current.
- static void **default_callback** (Fl_Window ∗, void ∗v)

  Back compatibility: Sets the default callback v for win to call on close event.
- static void default_icon (const Fl_RGB_Image ∗)

  Sets a single default window icon.
- static void default_icons (const Fl_RGB_Image ∗[ ], int)

  Sets the default window icons.
- static void default_icons (HICON big_icon, HICON small_icon)

  Sets the default window icons (Windows platform only).

- static const char * default_xclass ()

  Returns the default xclass.
- static void default_xclass (const char *)

  Sets the default window xclass.
- static bool is_a_rescale ()

  Returns true when a window is being rescaled.
- static char show_next_window_iconic ()

  Returns the static flag whether the next window should be opened iconified.
- static void show_next_window_iconic (char stat)

  Sets a static flag whether the next window should be opened iconified.

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()

  Returns the currently active group.
- static void current (Fl_Group *g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Window

- void default_size_range ()

  Protected method to calculate the default size range of a window.
- void draw () override

  Draws the widget.
- int force_position () const

  Returns the internal state of the window's `FORCE_POSITION` flag.
- void force_position (int force)

Sets an internal flag that tells FLTK and the window manager to honor position requests.

- void free_icons ()

  Deletes all icons previously attached to the window.
- int is_resizable ()

  Protected method to determine whether a window is resizable.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()

  Returns the internal array of widget sizes and positions.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget *, int)

  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)

  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)

  Allow derived groups to act when a child widget is removed from the group.
- int * sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

Gets the widget flags mask.

- void h (int v)

  Internal use only.

- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.

- void w (int v)

  Internal use only.

- void x (int v)

  Internal use only.

- void y (int v)

  Internal use only.

Static Protected Attributes inherited from Fl_Window

- static Fl_Window * current_

  Stores the last window that was made current.

13.32.1 Detailed Description

The Fl_Double_Window provides a double-buffered window.
It will draw the window data into an off-screen pixmap, and then copy it to the on-screen window.

13.32.2 Constructor & Destructor Documentation

13.32.2.1 ~Fl_Double_Window()

`Fl_Double_Window::~Fl_Double_Window ( )`
The destructor *also deletes all the children*.
This allows a whole tree to be deleted at once, without having to keep a pointer to all the children in the user code.

13.32.3 Member Function Documentation

13.32.3.1 as_double_window()

`Fl_Double_Window * Fl_Double_Window::as_double_window ( ) [inline], [override], [virtual]`
Return non-null if this is an Fl_Double_Window object.
Reimplemented from Fl_Window.

13.32.3.2 flush()

`void Fl_Double_Window::flush ( ) [override], [virtual]`
Forces the window to be drawn, this window is also made current and calls draw().
Reimplemented from Fl_Window.
Reimplemented in Fl_Overlay_Window.

13.32.3.3 hide()

`void Fl_Double_Window::hide ( ) [override], [virtual]`
Makes a widget invisible.

See also

show(), visible(), visible_r()

Reimplemented from Fl_Widget.
Reimplemented in Fl_Overlay_Window.

13.32.3.4 resize()

```
void Fl_Double_Window::resize (
            int x,
            int y,
            int w,
            int h )  [override], [virtual]
```

Changes the size or position of the widget.
This is a virtual function so that the widget may implement its own handling of resizing. The default version does *not* call the redraw() method, but instead relies on the parent widget to do so because the parent may know a faster way to update the display, such as scrolling from the old position.
Some window managers under X11 call resize() a lot more often than needed. Please verify that the position or size of a widget did actually change before doing any extensive calculations.
position(X, Y) is a shortcut for resize(X, Y, w(), h()), and size(W, H) is a shortcut for resize(x(), y(), W, H).

Parameters

in	*x,y*	new position relative to the parent window
in	*w,h*	new size

See also

position(int,int), size(int,int)

Reimplemented from Fl_Widget.
Reimplemented in Fl_Overlay_Window.

13.32.3.5 show()

```
void Fl_Double_Window::show ( )  [override], [virtual]
```

Makes a widget visible.
An invisible widget never gets redrawn and does not get keyboard or mouse events, but can receive a few other events like FL_SHOW.
The visible() method returns true if the widget is set to be visible. The visible_r() method returns true if the widget and all of its parents are visible. A widget is only visible if visible() is true on it *and all of its parents*.
Changing it will send FL_SHOW or FL_HIDE events to the widget. *Do not change it if the parent is not visible, as this will send false FL_SHOW or FL_HIDE events to the widget.* redraw() is called if necessary on this or the parent.

See also

hide(), visible(), visible_r()

Reimplemented from Fl_Widget.
Reimplemented in Fl_Overlay_Window.
The documentation for this class was generated from the following files:

- Fl_Double_Window.H
- Fl_Double_Window.cxx

13.33 Fl_End Class Reference

This is a dummy class that allows you to end a Fl_Group in a constructor list of a class:
`#include <Fl_Group.H>`

Public Member Functions

- **Fl_End** ()

  All it does is calling Fl_Group::current()->end()

13.33.1 Detailed Description

This is a dummy class that allows you to end a Fl_Group in a constructor list of a class:

```
class MyClass {
  Fl_Group group;
  Fl_Button button_in_group;
  Fl_End end;
  Fl_Button button_outside_group;
  MyClass();
};
MyClass::MyClass() :
  group(10,10,100,100),
  button_in_group(20,20,60,30),
  end(),
  button_outside_group(10,120,60,30) {
  [..ctor code..]
}
```

The documentation for this class was generated from the following file:

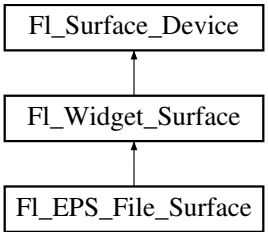
- Fl_Group.H

13.34 Fl_EPS_File_Surface Class Reference

Encapsulated PostScript drawing surface.

`#include <Fl_PostScript.H>`

Inheritance diagram for Fl_EPS_File_Surface:



Public Member Functions

- int close ()

  Completes all EPS output.
- FILE ∗ **file** ()

  Returns the underlying FILE pointer.
- Fl_EPS_File_Surface (int width, int height, FILE ∗eps_output, Fl_Color background=FL_WHITE, Fl_PostScript_Close_Command closef=NULL)

  Constructor.
- void origin (int ∗px, int ∗py) override

  Computes the coordinates of the current origin of graphics functions.
- void origin (int x, int y) override

  Sets the position of the origin of graphics in the drawable part of the drawing surface.
- int printable_rect (int ∗w, int ∗h) override

  Computes the width and height of the drawable area of the drawing surface.
- void translate (int x, int y) override

  Translates the current graphics origin accounting for the current rotation.
- void untranslate () override

  Undoes the effect of a previous translate() call.
- ∼Fl_EPS_File_Surface ()

  Destructor.

Public Member Functions inherited from Fl_Widget_Surface

- void draw (Fl_Widget *widget, int delta_x=0, int delta_y=0)

  Draws the widget on the drawing surface.
- void draw_decorated_window (Fl_Window *win, int x_offset=0, int y_offset=0)

  Draws a window with its title bar and frame if any.
- void print_window_part (Fl_Window *win, int x, int y, int w, int h, int delta_x=0, int delta_y=0)

  Draws a rectangular part of an on-screen window.

Public Member Functions inherited from Fl_Surface_Device

- Fl_Graphics_Driver * **driver** ()

  Returns the graphics driver of this drawing surface.
- virtual bool is_current ()

  Is this surface the current drawing surface?
- virtual void set_current (void)

  Make this surface the current drawing surface.
- virtual ~**Fl_Surface_Device** ()

  The destructor.

Protected Member Functions

- Fl_PostScript_Graphics_Driver * **driver** ()

  Returns the PostScript driver of this drawing surface.

Protected Member Functions inherited from Fl_Widget_Surface

- Fl_Widget_Surface (Fl_Graphics_Driver *d)

  The constructor.

Protected Member Functions inherited from Fl_Surface_Device

- void **driver** (Fl_Graphics_Driver *graphics_driver)

  Sets the graphics driver of this drawing surface.
- virtual void end_current ()

  FLTK calls this each time a surface ceases to be the current drawing surface.
- **Fl_Surface_Device** (Fl_Graphics_Driver *graphics_driver)

  Constructor that sets the graphics driver to use for the created surface.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Surface_Device

- static Fl_Surface_Device * pop_current ()

  Removes the top element from the current drawing surface stack, and makes the new top element current.
- static void push_current (Fl_Surface_Device *new_current)

  Pushes `new_current` on top of the stack of current drawing surfaces, and makes it current.
- static Fl_Surface_Device * surface ()

  The current drawing surface.

Protected Attributes inherited from Fl_Widget_Surface

- int **x_offset**

  horizontal offset to the origin of graphics coordinates
- int **y_offset**

  vertical offset to the origin of graphics coordinates

13.34.1 Detailed Description

Encapsulated PostScript drawing surface.
This drawing surface allows to store any FLTK graphics in vectorial form in an "Encapsulated PostScript" file.
Usage example:

```
Fl_Window *win = ...// Window to draw to an .eps file
int ww = win->decorated_w();
int wh = win->decorated_h();
FILE *eps = fl_fopen("/path/to/mywindow.eps", "w");
if (eps) {
  Fl_EPS_File_Surface *surface = new Fl_EPS_File_Surface(ww, wh, eps, win->color());
  Fl_Surface_Device::push_current(surface);
  surface->draw_decorated_window(win);
  Fl_Surface_Device::pop_current();
  delete surface; // the .eps file is not complete until the destructor was run
}
```

13.34.2 Constructor & Destructor Documentation

13.34.2.1 Fl_EPS_File_Surface()

```
Fl_EPS_File_Surface::Fl_EPS_File_Surface (
            int width,
            int height,
            FILE * eps_output,
            Fl_Color background = FL_WHITE,
            Fl_PostScript_Close_Command closef = NULL )
```

Constructor.

Parameters

width,height	Width and height of the EPS drawing area
eps_output	A writable FILE pointer where the Encapsulated PostScript data will be sent
background	Color expected to cover the background of the EPS drawing area. This parameter affects only the drawing of transparent Fl_RGB_Image objects: transparent areas of RGB images are blended with the `background` color. Under the X11 + pango platform, transparent RGB images are correctly blended to their background, thus this parameter has no effect.
closef	If not NULL, the destructor or close() will call `closef(eps_output)` after all EPS data has been sent. If NULL, `fclose(eps_output)` is called instead. This allows to close the FILE pointer by, e.g., `pclose`, or, using a function such as "`int keep_open(FILE*){return 0;}`", to keep it open after completion of all output to `eps_output`. Function `closef` should return non zero to indicate an error.

13.34.2.2 ~Fl_EPS_File_Surface()

```
Fl_EPS_File_Surface::~Fl_EPS_File_Surface ( )
```

Destructor.
By default, the destructor closes with function `fclose()` the underlying FILE. See the constructor for how to close it differently or to keep it open. Use close() before object destruction to receive the status code of output operations. If close() is not used and if EPS output results in error, the destructor displays an alert message with fl_alert().

13.34.3 Member Function Documentation

13.34.3.1 close()

```
int Fl_EPS_File_Surface::close ( )
```

Completes all EPS output.
The only operation possible with the Fl_EPS_File_Surface object after calling close() is its destruction.

Returns

The status code of output operations to the FILE object. 0 indicates success.

13.34.3.2 origin() [1/2]

```
void Fl_EPS_File_Surface::origin (
            int * x,
            int * y )  [override], [virtual]
```

Computes the coordinates of the current origin of graphics functions.

Parameters

out	*x,y*	If non-null, ∗x and ∗y are set to the horizontal and vertical coordinates of the graphics origin.

Reimplemented from Fl_Widget_Surface.

13.34.3.3 origin() [2/2]

```
void Fl_EPS_File_Surface::origin (
            int x,
            int y )  [override], [virtual]
```

Sets the position of the origin of graphics in the drawable part of the drawing surface.
Arguments should be expressed relatively to the result of a previous printable_rect() call. That is, `printable_rect(&w, &h); origin(w/2, 0);` sets the graphics origin at the top center of the drawable area. Successive origin() calls don't combine their effects. Origin() calls are not affected by rotate() calls (for classes derived from Fl_Paged_Device).

Parameters

in	*x,y*	Horizontal and vertical positions in the drawing surface of the desired origin of graphics.

Reimplemented from Fl_Widget_Surface.

13.34.3.4 printable_rect()

```
int Fl_EPS_File_Surface::printable_rect (
            int * w,
            int * h )  [override], [virtual]
```

Computes the width and height of the drawable area of the drawing surface.
Values are in the same unit as that used by FLTK drawing functions and are unchanged by calls to origin(). If the object is derived from class Fl_Paged_Device, values account for the user-selected paper type and print orientation and are changed by scale() calls.

Returns

0 if OK, non-zero if any error

Reimplemented from Fl_Widget_Surface.

13.34.3.5 translate()

```
void Fl_EPS_File_Surface::translate (
            int x,
            int y )  [override], [virtual]
```

Translates the current graphics origin accounting for the current rotation.
Each translate() call must be matched by an untranslate() call. Successive translate() calls add up their effects.
Reimplemented from Fl_Widget_Surface.

13.34.3.6 untranslate()

```
void Fl_EPS_File_Surface::untranslate (
            void  )  [override], [virtual]
```

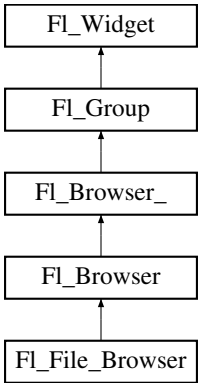
Undoes the effect of a previous translate() call.

Reimplemented from Fl_Widget_Surface.
The documentation for this class was generated from the following file:

- Fl_PostScript.H

13.35 Fl_File_Browser Class Reference

The Fl_File_Browser widget displays a list of filenames, optionally with file-specific icons.
`#include <Fl_File_Browser.H>`
Inheritance diagram for Fl_File_Browser:



Public Types

- enum { **FILES** , **DIRECTORIES** }

Public Types inherited from Fl_Browser

- enum Fl_Line_Position { **TOP** , **BOTTOM** , **MIDDLE** }

  For internal use only?

Public Types inherited from Fl_Browser_

- enum {
  HORIZONTAL = 1 , VERTICAL = 2 , BOTH = 3 , ALWAYS_ON = 4 ,
  HORIZONTAL_ALWAYS = 5 , VERTICAL_ALWAYS = 6 , BOTH_ALWAYS = 7 }

  Values for has_scrollbar().

Public Member Functions

- const char ∗ errmsg () const

  Returns OS error messages, or NULL if none.
- void errmsg (const char ∗emsg)

  Sets OS error message to a string, which can be NULL.
- int filetype () const

  Sets or gets the file browser type, FILES or DIRECTORIES.
- void filetype (int t)

  Sets or gets the file browser type, FILES or DIRECTORIES.
- const char ∗ filter () const

  Sets or gets the filename filter.
- void filter (const char ∗pattern)

  Sets or gets the filename filter.
- Fl_File_Browser (int, int, int, int, const char ∗=0)

The constructor creates the Fl_File_Browser widget at the specified position and size.

- uchar iconsize () const

  Sets or gets the size of the icons.
- void iconsize (uchar s)

  Sets or gets the size of the icons.
- int load (const char *directory, Fl_File_Sort_F *sort=fl_numericsort)

  Loads the specified directory into the browser.
- Fl_Fontsize **textsize** () const
- void **textsize** (Fl_Fontsize s)

Public Member Functions inherited from Fl_Browser

- void add (const char *newtext, void *d=0)

  Adds a new line to the end of the browser.
- void bottomline (int line)

  Scrolls the browser so the bottom item in the browser is showing the specified `line`.
- void clear ()

  Removes all the lines in the browser.
- char column_char () const

  Gets the current column separator character.
- void column_char (char c)

  Sets the column separator to c.
- const int * column_widths () const

  Gets the current column width array.
- void column_widths (const int *arr)

  Sets the current array to `arr`.
- void * data (int line) const

  Returns the user data() for specified `line`.
- void data (int line, void *d)

  Sets the user data for specified `line` to `d`.
- void display (int line, int val=1)

  For back compatibility.
- int displayed (int line) const

  Returns non-zero if `line` has been scrolled to a position where it is being displayed.
- Fl_Browser (int X, int Y, int W, int H, const char *L=0)

  The constructor makes an empty browser.
- char format_char () const

  Gets the current format code prefix character, which by default is '@'.
- void format_char (char c)

  Sets the current format code prefix character to `c`.
- void hide () override

  Hides the entire Fl_Browser widget – opposite of show().
- void hide (int line)

  Makes `line` invisible, preventing selection by the user.
- Fl_Image * icon (int line) const

  Returns the icon currently defined for `line`.
- void icon (int line, Fl_Image *icon)

  Set the image icon for `line` to the value `icon`.
- void insert (int line, const char *newtext, void *d=0)

  Insert a new entry whose label is `newtext` above given `line`, optional data `d`.
- void lineposition (int line, Fl_Line_Position pos)

Updates the browser so that `line` is shown at position `pos`.

- int load (const char *filename)

  Clears the browser and reads the file, adding each line from the file to the browser.
- void make_visible (int line)

  Make the item at the specified `line` visible().
- void middleline (int line)

  Scrolls the browser so the middle item in the browser is showing the specified `line`.
- void move (int to, int from)

  Line `from` is removed and reinserted at `to`.
- void remove (int line)

  Remove entry for given `line` number, making the browser one line shorter.
- void remove_icon (int line)

  Removes the icon for `line`.
- void **replace** (int a, const char *b)

  For back compatibility only.
- int select (int line, int val=1)

  Sets the selection state of the item at `line` to the value `val`.
- int selected (int line) const

  Returns 1 if specified `line` is selected, 0 if not.
- void show () override

  Shows the entire Fl_Browser widget – opposite of hide().
- void show (int line)

  Makes `line` visible, and available for selection by user.
- int size () const

  Returns how many lines are in the browser.
- void **size** (int W, int H)
- void swap (int a, int b)

  Swaps two browser lines `a` and `b`.
- const char * text (int line) const

  Returns the label text for the specified `line`.
- void text (int line, const char *newtext)

  Sets the text for the specified `line` to `newtext`.
- Fl_Fontsize **textsize** () const

  Gets the default text size (in pixels) for the lines in the browser.
- void textsize (Fl_Fontsize newSize)

  Sets the default text size (in pixels) for the lines in the browser to `newSize`.
- int topline () const

  Returns the line that is currently visible at the top of the browser.
- void topline (int line)

  Scrolls the browser so the top item in the browser is showing the specified `line`.
- int value () const

  Returns the line number of the currently selected line, or 0 if none selected.
- void value (int line)

  Sets the browser's value(), which selects the specified `line`.
- int visible (int line) const

  Returns non-zero if the specified `line` is visible, 0 if hidden.
- ~**Fl_Browser** ()

  The destructor deletes all list items and destroys the browser.

Public Member Functions inherited from Fl_Browser_

- int deselect (int docallbacks=0)

  Deselects all items in the list and returns 1 if the state changed or 0 if it did not.
- void display (void *item)

  Displays the `item`, scrolling the list as necessary.
- int handle (int event) override

  Handles the `event` within the normal widget bounding box.
- uchar **has_scrollbar** () const

  Returns the current scrollbar mode, see Fl_Browser_::has_scrollbar(uchar)
- void has_scrollbar (uchar mode)

  Sets whether the widget should have scrollbars or not (default Fl_Browser_::BOTH).
- int hposition () const

  Gets the horizontal scroll position of the list as a pixel position `pos`.
- void hposition (int)

  Sets the horizontal scroll position of the list to pixel position `pos`.
- int linespacing () const

  Return the height of additional spacing between browser lines.
- void linespacing (int pixels)

  Add some space between browser lines.
- int position () const
- void position (int pos)
- void **position** (int x, int y)
- void resize (int X, int Y, int W, int H) override

  Repositions and/or resizes the browser.
- void scrollbar_left ()

  Moves the vertical scrollbar to the lefthand side of the list.
- void scrollbar_right ()

  Moves the vertical scrollbar to the righthand side of the list.
- int scrollbar_size () const

  Gets the current size of the scrollbars' troughs, in pixels.
- void scrollbar_size (int newSize)

  Sets the pixel size of the scrollbars' troughs to `newSize`, in pixels.
- int select (void *item, int val=1, int docallbacks=0)

  Sets the selection state of `item` to `val`, and returns 1 if the state changed or 0 if it did not.
- int select_only (void *item, int docallbacks=0)

  Selects `item` and returns 1 if the state changed or 0 if it did not.
- void sort (int flags=0)

  Sort the items in the browser based on `flags`.
- Fl_Color **textcolor** () const

  Gets the default text color for the lines in the browser.
- void **textcolor** (Fl_Color col)

  Sets the default text color for the lines in the browser to color `col`.
- Fl_Font textfont () const

  Gets the default text font for the lines in the browser.
- void **textfont** (Fl_Font font)

  Sets the default text font for the lines in the browser to `font`.
- Fl_Fontsize **textsize** () const

  Gets the default text size (in pixels) for the lines in the browser.
- void **textsize** (Fl_Fontsize newSize)

  Sets the default text size (in pixels) for the lines in the browser to `size`.

- int vposition () const

  Gets the vertical scroll position of the list as a pixel position `pos`.
- void vposition (int pos)

  Sets the vertical scroll position of the list to pixel position `pos`.

Public Member Functions inherited from Fl_Group

- Fl_Widget *& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget *o)

  See void Fl_Group::add(Fl_Widget &w)
- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget *const * array () const

  Returns a pointer to the internal array of children.
- Fl_Group const * as_group () const override
- Fl_Group * as_group () override

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const

  Returns the n'th child.
- int **children** () const

  Returns how many child widgets the group has.
- void clear ()

  Deletes all child widgets from memory recursively.
- unsigned int clip_children () const

  Returns the current clipping mode.
- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)

  Removes the widget at `index` *from the group and deletes it.*
- void end ()

  Exactly the same as current(this->parent()).
- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*
- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.
- void **forms_end** ()

  This is for forms compatibility only.
- int handle (int) override

  Handles the specified event.
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

This does insert(w, find(before)).

- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.
- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.
- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- void resize (int, int, int, int) override

  Resizes the Fl_Group widget and all of its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

Gets the box type of the widget.

- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.

- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

Sets the label color.

- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.

- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()

  Returns the currently active group.
- static void current (Fl_Group *g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Public Attributes inherited from Fl_Browser_

- Fl_Scrollbar hscrollbar

  Horizontal scrollbar.
- Fl_Scrollbar scrollbar

  Vertical scrollbar.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Browser

- FL_BLINE * _remove (int line)

  Removes the item at the specified `line`.
- const void * **bline_data** (const FL_BLINE *b) const

  Get FL_BLINE data.
- void *& **bline_data** (FL_BLINE *b) const

  Get writable reference to FL_BLINE data.
- char **bline_flags** (const FL_BLINE *b) const

  Get FL_BLINE flags.
- char & **bline_flags** (FL_BLINE *b) const

  Get writable reference to FL_BLINE flags.

- const Fl_Image * **bline_icon** (const FL_BLINE *b) const

  Get FL_BLINE icon.
- Fl_Image *& **bline_icon** (FL_BLINE *b) const

  Get writable reference to FL_BLINE icon.
- short bline_length (const FL_BLINE *b) const

  Get FL_BLINE allocated text buffer size (excl.
- const char * **bline_txt** (const FL_BLINE *b) const

  Get FL_BLINE text.
- char * **bline_txt** (FL_BLINE *b) const

  Get writable reference to FL_BLINE text.
- FL_BLINE * find_line (int line) const

  Returns the item for specified `line`.
- void insert (int line, FL_BLINE *item)

  Insert specified `item` above `line`.
- void * item_at (int line) const override

  Return the item at specified `line`.
- void * item_first () const override

  Returns the very first item in the list.
- void * item_last () const override

  Returns the very last item in the list.
- void * item_next (void *item) const override

  Returns the next item after `item`.
- void * item_prev (void *item) const override

  Returns the previous item before `item`.
- void item_select (void *item, int val) override

  Change the selection state of `item` to the value `val`.
- int item_selected (void *item) const override

  See if `item` is selected.
- void item_swap (void *a, void *b) override

  Swap the items `a` and `b`.
- const char * item_text (void *item) const override

  Returns the label text for `item`.
- int lineno (void *item) const

  Returns line number corresponding to `item`, or zero if not found.
- void swap (FL_BLINE *a, FL_BLINE *b)

  Swap the two items `a` and `b`.

Protected Member Functions inherited from Fl_Browser_

- void bbox (int &X, int &Y, int &W, int &H) const

  Returns the bounding box for the interior of the list's display window, inside the scrollbars.
- void deleting (void *item)

  This method should be used when `item` is being deleted from the list.
- int displayed (void *item) const

  Returns non-zero if `item` has been scrolled to a position where it is being displayed.
- void draw () override

  Draws the list within the normal widget bounding box.
- void * find_item (int ypos)

  This method returns the item under mouse y position `ypos`.
- Fl_Browser_ (int X, int Y, int W, int H, const char *L=0)

  The constructor makes an empty browser.

- virtual int full_width () const

  This method may be provided by the subclass to indicate the full width of the item list, in pixels.
- void inserting (void *a, void *b)

  This method should be used when an item is in the process of being inserted into the list.
- virtual int item_quick_height (void *item) const

  This method may be provided by the subclass to return the height of the `item`, in pixels.
- int leftedge () const

  This method returns the X position of the left edge of the list area after adjusting for the scrollbar and border, if any.
- void new_list ()

  This method should be called when the list data is completely replaced or cleared.
- void redraw_line (void *item)

  This method should be called when the contents of `item` has changed, but not its height.
- void redraw_lines ()

  This method will cause the entire list to be redrawn.
- void replacing (void *a, void *b)

  This method should be used when item `a` is being replaced by item `b`.
- void * selection () const

  Returns the item currently selected, or NULL if there is no selection.
- void swapping (void *a, void *b)

  This method should be used when two items `a` and `b` are being swapped.
- void * **top** () const

  Returns the item that appears at the top of the list.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()

  Returns the internal array of widget sizes and positions.
- void draw () override

  Draws the widget.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget *, int)

  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)

  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)

  Allow derived groups to act when a child widget is removed from the group.
- int * sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Static Protected Attributes inherited from Fl_Browser

- static constexpr char **BLINE_NOTDISPLAYED** = 2
- static constexpr char **BLINE_SELECTED** = 1

13.35.1 Detailed Description

The Fl_File_Browser widget displays a list of filenames, optionally with file-specific icons.

13.35.2 Constructor & Destructor Documentation

13.35.2.1 Fl_File_Browser()

```
Fl_File_Browser::Fl_File_Browser (
            int X,
            int Y,
            int W,
            int H,
            const char * l = 0 )
```

The constructor creates the Fl_File_Browser widget at the specified position and size.
The destructor destroys the widget and frees all memory that has been allocated.

13.35.3 Member Function Documentation

13.35.3.1 errmsg() [1/2]

```
const char * Fl_File_Browser::errmsg ( ) const  [inline]
```

Returns OS error messages, or NULL if none.
Use when advised.

13.35.3.2 errmsg() [2/2]

```
void Fl_File_Browser::errmsg (
            const char * emsg )
```

Sets OS error message to a string, which can be NULL.
Frees previous if any. void errmsg(const char *emsg);

13.35.3.3 filetype() [1/2]

```
int Fl_File_Browser::filetype ( ) const  [inline]
```

Sets or gets the file browser type, FILES or DIRECTORIES.
When set to FILES, both files and directories are shown. Otherwise only directories are shown.

13.35.3.4 filetype() [2/2]

```
void Fl_File_Browser::filetype (
            int t )  [inline]
```

Sets or gets the file browser type, FILES or DIRECTORIES.
When set to FILES, both files and directories are shown. Otherwise only directories are shown.

13.35.3.5 filter() [1/2]

```
const char * Fl_File_Browser::filter ( ) const  [inline]
```

Sets or gets the filename filter.
The pattern matching uses the fl_filename_match() function in FLTK.

13.35.3.6 filter() [2/2]

```
void Fl_File_Browser::filter (
            const char * pattern )
```

Sets or gets the filename filter.
The pattern matching uses the fl_filename_match() function in FLTK.

13.35.3.7 iconsize() [1/2]

```
uchar Fl_File_Browser::iconsize ( ) const  [inline]
```

Sets or gets the size of the icons.
The default size is 20 pixels.

13.35.3.8 iconsize() [2/2]

```
void Fl_File_Browser::iconsize (
            uchar s )  [inline]
```

Sets or gets the size of the icons.
The default size is 20 pixels.

13.35.3.9 load()

```
int Fl_File_Browser::load (
            const char * directory,
            Fl_File_Sort_F * sort = fl_numericsort )
```

Loads the specified directory into the browser.
If icons have been loaded then the correct icon is associated with each file in the list.
If directory is "", all mount points (unix) or drive letters (Windows) are listed.
The sort argument specifies a sort function to be used with fl_filename_list().
Return value is the number of filename entries, or 0 if none. On error, 0 is returned, and errmsg() has OS error string if non-NULL.
The documentation for this class was generated from the following files:

- Fl_File_Browser.H
- Fl_File_Browser.cxx

13.36 Fl_File_Chooser Class Reference

The Fl_File_Chooser widget displays a standard file selection dialog that supports various selection modes.

Public Types

- enum Type { SINGLE = 0 , MULTI = 1 , CREATE = 2 , DIRECTORY = 4 }

  Determines the type of file chooser presented to the user.

Public Member Functions

- Fl_Widget * add_extra (Fl_Widget *gr)

  Adds an extra widget at the bottom of the Fl_File_Chooser window.
- void **callback** (void(*cb)(Fl_File_Chooser *, void *), void *d=0)

  Sets the file chooser callback cb and associated data d.
- Fl_Color **color** ()

  Gets the background color of the Fl_File_Browser list.
- void **color** (Fl_Color c)

  Sets the background color of the Fl_File_Browser list.
- int **count** ()

  Returns the number of selected files.
- char * **directory** ()

  Gets the current directory.
- void **directory** (const char *d)

  Sets the current directory.
- const char * **filter** ()

  Gets the current filename filter patterns.
- void filter (const char *p)

  Sets the current filename filter patterns.
- int **filter_value** ()

  Gets the current filename filter selection.

- void **filter_value** (int f)

  Sets the current filename filter selection.
- Fl_File_Chooser (const char *pathname, const char *pattern, int type_val, const char *title)

  The constructor creates the Fl_File_Chooser dialog shown.
- int **h** () const
- void **hide** ()

  Hides the Fl_File_Chooser window.
- uchar iconsize ()

  Gets the size of the icons in the Fl_File_Browser.
- void iconsize (uchar s)

  Sets the size of the icons in the Fl_File_Browser.
- const char * **label** ()

  Gets the title bar text for the Fl_File_Chooser.
- void **label** (const char *l)

  Sets the title bar text for the Fl_File_Chooser.
- const char * **ok_label** ()

  Gets the label for the "ok" button in the Fl_File_Chooser.
- void **ok_label** (const char *l)

  Sets the label for the "ok" button in the Fl_File_Chooser.
- void **position** (int x, int y)
- int **preview** () const

  Returns the current state of the preview box.
- void preview (int e)

  Enable or disable the preview tile.
- void **rescan** ()

  Reloads the current directory in the Fl_File_Browser.
- void **rescan_keep_filename** ()

  Rescan the current directory without clearing the filename, then select the file if it is in the list.
- void **resize** (int x, int y, int w, int h)
- void **show** ()

  Shows the Fl_File_Chooser window.
- int shown ()

  Returns non-zero if the file chooser main window show() has been called, but not hide().
- void **size** (int w, int h)
- Fl_Color **textcolor** ()

  Gets the current Fl_File_Browser text color.
- void **textcolor** (Fl_Color c)

  Sets the current Fl_File_Browser text color.
- Fl_Font **textfont** ()

  Gets the current Fl_File_Browser text font.
- void **textfont** (Fl_Font f)

  Sets the current Fl_File_Browser text font.
- Fl_Fontsize **textsize** ()

  Gets the current Fl_File_Browser text size.
- void **textsize** (Fl_Fontsize s)

  Sets the current Fl_File_Browser text size.
- int **type** ()

  Gets the current type of Fl_File_Chooser.
- void **type** (int t)

  Sets the current type of Fl_File_Chooser.
- void * **user_data** () const

Gets the file chooser user data.

- void **user_data** (void *d)

  Sets the file chooser user data d.
- void value (const char *filename)

  Sets the current value of the selected file.
- const char * value (int f=1)

  Gets the current value of the selected file(s).
- int **visible** ()

  Returns 1 if the Fl_File_Chooser window is visible.
- int **w** () const
- int **x** () const
- int **y** () const
- ~**Fl_File_Chooser** ()

  Destroys the widget and frees all memory used by it.

Public Attributes

- Fl_Button * **newButton**

  The "new directory" button is exported so that application developers can control the appearance and use.
- Fl_Check_Button * **previewButton**

  The "preview" button is exported so that application developers can control the appearance and use.
- Fl_Check_Button * showHiddenButton

  When checked, hidden files (i.e., filename begins with dot) are displayed.

Static Public Attributes

- static const char * **add_favorites_label** = "Add to Favorites"

  [standard text may be customized at run-time]
- static const char * **all_files_label** = "All Files (*)"

  [standard text may be customized at run-time]
- static const char * **custom_filter_label** = "Custom Filter"

  [standard text may be customized at run-time]
- static const char * **existing_file_label** = "Please choose an existing file!"

  [standard text may be customized at run-time]
- static const char * **favorites_label** = "Favorites"

  [standard text may be customized at run-time]
- static const char * **filename_label** = "Filename:"

  [standard text may be customized at run-time]
- static const char * **filesystems_label** = Fl::system_driver()->filesystems_label()

  [standard text may be customized at run-time]
- static const char * **hidden_label** = "Show hidden files"

  [standard text may be customized at run-time]
- static const char * **manage_favorites_label** = "Manage Favorites"

  [standard text may be customized at run-time]
- static const char * **new_directory_label** = "New Directory?"

  [standard text may be customized at run-time]
- static const char * **new_directory_tooltip** = "Create a new directory."

  [standard text may be customized at run-time]
- static const char * **preview_label** = "Preview"

  [standard text may be customized at run-time]
- static const char * **save_label** = "Save"

  [standard text may be customized at run-time]

- static const char ∗ **show_label** = "Show:"

  [standard text may be customized at run-time]
- static Fl_File_Sort_F ∗ **sort** = fl_numericsort

  the sort function that is used when loading the contents of a directory.

Protected Member Functions

- void **show_error_box** (int val)

  Show error box if val=1, hide if val=0.

Related Symbols

(Note that these are not member symbols.)

- char ∗ fl_dir_chooser (const char ∗message, const char ∗fname, int relative)

  Shows a file chooser dialog and gets a directory.
- char ∗ fl_file_chooser (const char ∗message, const char ∗pat, const char ∗fname, int relative)

  Shows a file chooser dialog and gets a filename.
- void fl_file_chooser_callback (void(∗cb)(const char ∗))

  Set the file chooser callback.
- void fl_file_chooser_ok_label (const char ∗l)

  Set the "OK" button label.

13.36.1 Detailed Description

The Fl_File_Chooser widget displays a standard file selection dialog that supports various selection modes.

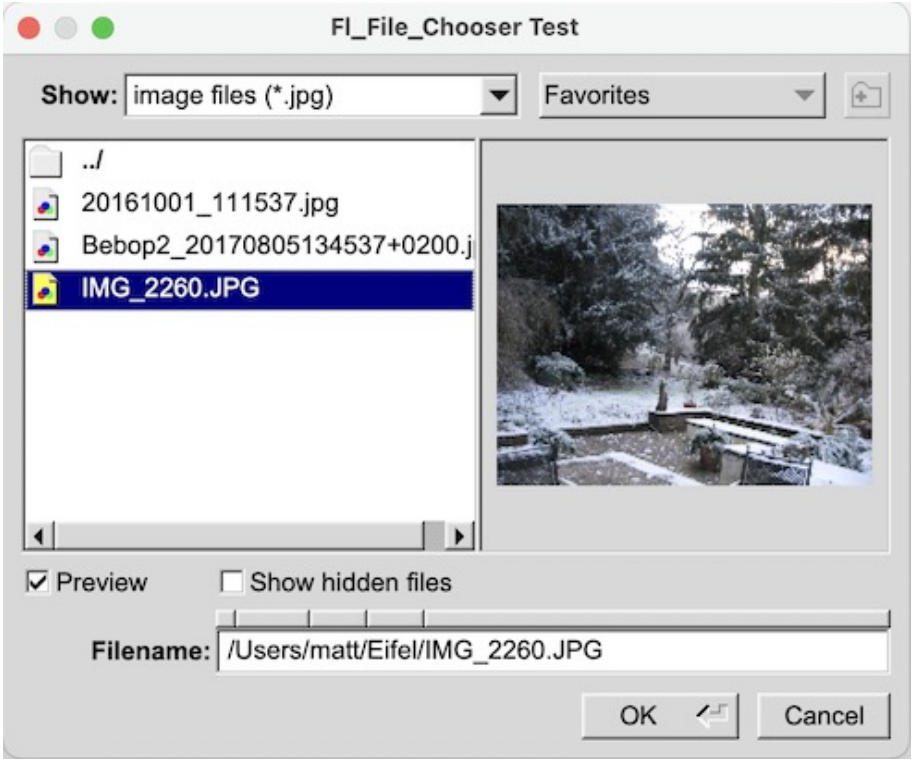


Figure 13.16 Fl_File_Chooser

Features include:

- Multiple filter patterns can be specified, with parenthesis around filters, and tabs to separate each pattern, e.g.:
```
char pattern[] = "Image Files (*.{bmp,gif,jpg,png,xbm,xpm})\t"
                 "Web Files (*.{htm,html,php})\t"
                 "All Files (*)";
```
- If no "*" pattern is provided, then an entry for "All Files (*)" is automatically added.
- An optional file preview box is provided which can be toggled by programmer or user showing images, or the first 2048 bytes of printable text.
- Preview image loading functions can be registered to provide custom file previews.
- The favorites button shows up to 100 user-saved favorite directories, the user's home directory, and a filesystems item.
- A simple dialog is provided for managing saved directories.
- Shortcut keys are provided:

Shortcut	Description
Alt+a	Adds a directory to the favorites list
Alt+m	Manages the favorites list
Alt+f	Shows the filesystem list
Alt+h	Go to the home directory
Alt+0..9	going to any of the first 10 favorites

The Fl_File_Chooser widget transmits UTF-8 encoded filenames to its user. It is recommended to open files that may have non-ASCII names with the fl_fopen() or fl_open() utility functions that handle these names in a cross-platform way (whereas the standard fopen()/open() functions fail on the Windows platform to open files with a non-ASCII name).
The Fl_File_Chooser class also exports several static values that may be used to localize or customize the appearance of all file chooser dialogs:

Member	Default value
add_favorites_label	"Add to Favorites"
all_files_label	"All Files (*)"
custom_filter_label	"Custom Filter"
existing_file_label	"Please choose an existing file!"
favorites_label	"Favorites"
filename_label	"Filename:"
filesystems_label	"My Computer" (Windows) "File Systems" (all others)
hidden_label	"Show hidden files:"
manage_favorites_label	"Manage Favorites"
new_directory_label	"New Directory?"
new_directory_tooltip	"Create a new directory."
preview_label	"Preview"
save_label	"Save"
show_label	"Show:"
sort	fl_numericsort

The Fl_File_Chooser::sort member specifies the sort function that is used when loading the contents of a directory and can be customized at run-time.
The Fl_File_Chooser class also exports the Fl_File_Chooser::newButton and Fl_File_Chooser::previewButton widgets so that application developers can control their appearance and use.

13.36.2 Member Enumeration Documentation

13.36.2.1 Type

```
enum Fl_File_Chooser::Type
```

Determines the type of file chooser presented to the user.

Enumerator

SINGLE	Select a single, existing file.
MULTI	Select one or more existing files.
CREATE	When used alone, select a single, existing file or specify a new filename. Can be combined with DIRECTORY (e.g. CREATE|DIRECTORY) to have the same effect for directories.
DIRECTORY	Select a single, existing directory. Can be combined with CREATE.

13.36.3 Constructor & Destructor Documentation

13.36.3.1 Fl_File_Chooser()

```
Fl_File_Chooser::Fl_File_Chooser (
            const char * pathname,
            const char * pattern,
            int type_val,
            const char * title )
```

The constructor creates the Fl_File_Chooser dialog shown.

- The `pathname` argument can be a directory name or a complete file name (in which case the corresponding file is highlighted in the list and in the filename input field.)
- The `pattern` argument can be a NULL string or "*" to list all files, or it can be a series of descriptions and filter strings separated by tab characters (\t). The format of filters is either "Description text (patterns)" or just "patterns". A file chooser that provides filters for HTML and image files might look like:
  `"HTML Files (*.html)\tImage Files (*.{bmp,gif,jpg,png})"`

  The file chooser will automatically add the "All Files (*)" pattern to the end of the string you pass if you do not provide one. The first filter in the string is the default filter. `See` the FLTK documentation on fl_filename_match() for the kinds of pattern strings that are supported.
- The `type_val` argument can be one of the Fl_File_Chooser::Type values.
- The `title` argument is used to set the title bar text for the Fl_File_Chooser window.

13.36.4 Member Function Documentation

13.36.4.1 add_extra()

```
Fl_Widget * Fl_File_Chooser::add_extra (
            Fl_Widget * extra )
```

Adds an extra widget at the bottom of the Fl_File_Chooser window.
You can use any Fl_Widget or Fl_Group. If you use an Fl_Group, set its (x, y) coordinates to (0, 0) and position its children relative to (0, 0) inside the Fl_Group container widget. Make sure that all child widgets of the Fl_Group are entirely included inside the bounding box of their parents, i.e. the Fl_Group widget, and the Fl_File_Chooser window, respectively.

Note

The width of the Fl_File_Chooser window is an undocumented implementation detail and may change in the future.

If `extra` is NULL any previous extra widget is removed.

Parameters

in	*extra*	Custom widget or group to be added to the Fl_File_Chooser window.

Returns

Pointer to previous extra widget or NULL if not set previously.

Note

Fl_File_Chooser does **not** delete the extra widget in its destructor! The extra widget is removed from the Fl_File_Chooser window before the Fl_File_Chooser widget gets destroyed. To prevent memory leakage, don't forget to delete unused extra widgets.

13.36.4.2 filter()

```
void Fl_File_Chooser::filter (
            const char * p )
```

Sets the current filename filter patterns.
The filter patterns use fl_filename_match(). Multiple patterns can be used by separating them with tabs, like `"*.jpg\t*.png\t*.gif\t*"`. In addition, you can provide human-readable labels with the patterns inside parenthesis, like `"JPEG Files (*.jpg)\tPNG Files (*.png)\tGIF Files (*.gif)\tAll Files (*)"` .
Use filter(NULL) to show all files.

13.36.4.3 iconsize() [1/2]

```
uchar Fl_File_Chooser::iconsize ( )
```

Gets the size of the icons in the Fl_File_Browser.
By default the icon size is set to 1.5 times the textsize().

13.36.4.4 iconsize() [2/2]

```
void Fl_File_Chooser::iconsize (
            uchar s )
```

Sets the size of the icons in the Fl_File_Browser.
By default the icon size is set to 1.5 times the textsize().

13.36.4.5 preview()

```
void Fl_File_Chooser::preview (
            int e )
```

Enable or disable the preview tile.
1 = enable preview, 0 = disable preview.

13.36.4.6 shown()

```
int Fl_File_Chooser::shown ( )
```

Returns non-zero if the file chooser main window show() has been called, but not hide().

See also

Fl_Window::shown()

13.36.4.7 value() [1/2]

```
void Fl_File_Chooser::value (
            const char * filename )
```

Sets the current value of the selected file.
If a relative path is provided in `filename` it is converted to an absolute path.
If `NULL` or an empty string is provided, the working directory is changed to the user's current directory and the filename is set to "".
After assigning the filename the entire string (if any) is selected, i.e.

- insert_position() is 0 (zero)
- mark() is strlen(<expanded filename>).

Note

The selection of the entire string may not always be desired but it is kept for backwards compatibility.

Parameters

in	*filename*	relative or absolute filename, may be NULL or ""

13.36.4.8 value() [2/2]

```
const char * Fl_File_Chooser::value (
            int f = 1 )
```

Gets the current value of the selected file(s).
`f` is a `1-based` index into a list of file names. The number of selected files is returned by Fl_File_Chooser::count().
This sample code loops through all selected files:

```
// Get list of filenames user selected from a MULTI chooser
for ( int t=1; t<=chooser->count(); t++ ) {
    const char *filename = chooser->value(t);
    ...
}
```

13.36.5 Member Data Documentation

13.36.5.1 showHiddenButton

```
Fl_File_Chooser::showHiddenButton
```

When checked, hidden files (i.e., filename begins with dot) are displayed.
The "showHiddenButton" button is exported so that application developers can control its appearance.
The documentation for this class was generated from the following files:

- Fl_File_Chooser.H
- Fl_File_Chooser.cxx
- Fl_File_Chooser2.cxx
- fl_file_dir.cxx

13.37 Fl_File_Icon Class Reference

The Fl_File_Icon class manages icon images that can be used as labels in other widgets and as icons in the FileBrowser widget.
`#include <Fl_File_Icon.H>`

Public Types

- enum {
  ANY , **PLAIN** , **FIFO** , **DEVICE** ,
  LINK , **DIRECTORY** }

- enum {
  END , **COLOR** , **LINE** , **CLOSEDLINE** ,
  POLYGON , **OUTLINEPOLYGON** , **VERTEX** }

Public Member Functions

- short ∗ add (short d)

  Adds a keyword value to the icon array, returning a pointer to it.
- short ∗ add_color (Fl_Color c)

  Adds a color value to the icon array, returning a pointer to it.
- short ∗ add_vertex (float x, float y)

  Adds a vertex value to the icon array, returning a pointer to it.
- short ∗ add_vertex (int x, int y)

  Adds a vertex value to the icon array, returning a pointer to it.
- void **clear** ()

  Clears all icon data from the icon.
- void draw (int x, int y, int w, int h, Fl_Color ic, int active=1)

  Draws an icon in the indicated area.
- Fl_File_Icon (const char ∗p, int t, int nd=0, short ∗d=0)

  Creates a new Fl_File_Icon with the specified information.
- void label (Fl_Widget ∗w)

  Applies the icon to the widget, registering the Fl_File_Icon label type as needed.
- void load (const char ∗f)

  Loads the specified icon image.
- int load_fti (const char ∗fti)

  Loads an SGI icon file.
- int load_image (const char ∗i)

  Load an image icon file from an image filename.
- Fl_File_Icon ∗ next () const

  Returns next file icon object.
- const char ∗ **pattern** () const

  Returns the filename matching pattern for the icon.
- int **size** () const

  Returns the number of words of data used by the icon.
- int type () const

  Returns the filetype associated with the icon, which can be one of the following:
- short ∗ **value** () const

  Returns the data array for the icon.
- ∼**Fl_File_Icon** ()

  The destructor destroys the icon and frees all memory that has been allocated for it.

Static Public Member Functions

- static Fl_File_Icon ∗ find (const char ∗filename, int filetype=ANY)

  Finds an icon that matches the given filename and file type.
- static Fl_File_Icon ∗ **first** ()

  Returns a pointer to the first icon in the list.
- static void labeltype (const Fl_Label ∗o, int x, int y, int w, int h, Fl_Align a)

  Draw the icon label.
- static void load_system_icons (void)

  Loads all system-defined icons.

13.37.1 Detailed Description

The Fl_File_Icon class manages icon images that can be used as labels in other widgets and as icons in the FileBrowser widget.

13.37.2 Constructor & Destructor Documentation

13.37.2.1 Fl_File_Icon()

```
Fl_File_Icon::Fl_File_Icon (
            const char * p,
            int t,
            int nd = 0,
            short * d = 0 )
```

Creates a new Fl_File_Icon with the specified information.

Parameters

in	*p*	filename pattern
in	*t*	file type
in	*nd*	number of data values
in	*d*	data values

13.37.3 Member Function Documentation

13.37.3.1 add()

```
short * Fl_File_Icon::add (
            short d )
```

Adds a keyword value to the icon array, returning a pointer to it.

Parameters

in	*d*	data value

13.37.3.2 add_color()

```
short * Fl_File_Icon::add_color (
            Fl_Color c )  [inline]
```

Adds a color value to the icon array, returning a pointer to it.

Parameters

in	*c*	color value

13.37.3.3 add_vertex() [1/2]

```
short * Fl_File_Icon::add_vertex (
            float x,
            float y )  [inline]
```

Adds a vertex value to the icon array, returning a pointer to it.
The floating point version goes from 0.0 to 1.0. The origin (0.0) is in the lower-lefthand corner of the icon.

Parameters

in	*x,y*	vertex coordinates

13.37.3.4 add_vertex() [2/2]

```
short * Fl_File_Icon::add_vertex (
            int x,
            int y )  [inline]
```

Adds a vertex value to the icon array, returning a pointer to it.
The integer version accepts coordinates from 0 to 10000. The origin (0.0) is in the lower-lefthand corner of the icon.

Parameters

in	*x,y*	vertex coordinates

13.37.3.5 draw()

```
void Fl_File_Icon::draw (
            int x,
            int y,
            int w,
            int h,
            Fl_Color ic,
            int active = 1 )
```

Draws an icon in the indicated area.

Parameters

in	*x,y,w,h*	position and size
in	*ic*	icon color
in	*active*	status, default is active [non-zero]

13.37.3.6 find()

```
Fl_File_Icon * Fl_File_Icon::find (
            const char * filename,
            int filetype = ANY )  [static]
```

Finds an icon that matches the given filename and file type.

Parameters

in	*filename*	name of file
in	*filetype*	enumerated file type

Returns

matching file icon or NULL

13.37.3.7 label()

```
void Fl_File_Icon::label (
            Fl_Widget * w )
```

Applies the icon to the widget, registering the Fl_File_Icon label type as needed.

Parameters

in	*w*	widget for which this icon will become the label

13.37.3.8 labeltype()

```
void Fl_File_Icon::labeltype (
            const Fl_Label * o,
            int x,
            int y,
            int w,
            int h,
            Fl_Align a )  [static]
```

Draw the icon label.

Parameters

in	*o*	label data
in	*x,y,w,h*	position and size of label
in	*a*	label alignment [not used]

13.37.3.9 load()

```
void Fl_File_Icon::load (
            const char * f )
```

Loads the specified icon image.
The format is deduced from the filename.

Parameters

in	*f*	filename

13.37.3.10 load_fti()

```
int Fl_File_Icon::load_fti (
            const char * fti )
```

Loads an SGI icon file.

Parameters

in	*fti*	icon filename

Returns

0 on success, non-zero on error

13.37.3.11 load_image()

```
int Fl_File_Icon::load_image (
            const char * ifile )
```

Load an image icon file from an image filename.

Parameters

in	*ifile*	image filename

Returns

0 on success, non-zero on error

13.37.3.12 load_system_icons()

```
void Fl_File_Icon::load_system_icons (
            void ) [static]
```

Loads all system-defined icons.
This call is useful when using the FileChooser widget and should be used when the application starts:
`Fl_File_Icon::load_system_icons();`

13.37.3.13 next()

```
Fl_File_Icon * Fl_File_Icon::next ( ) const [inline]
```

Returns next file icon object.
See Fl_File_Icon::first()

13.37.3.14 type()

```
int Fl_File_Icon::type ( ) const [inline]
```

Returns the filetype associated with the icon, which can be one of the following:

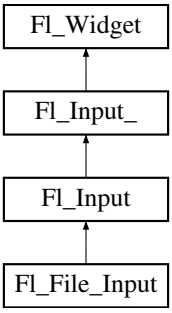
- Fl_File_Icon::ANY, any kind of file.
- Fl_File_Icon::PLAIN, plain files.
- Fl_File_Icon::FIFO, named pipes.
- Fl_File_Icon::DEVICE, character and block devices.
- Fl_File_Icon::LINK, symbolic links.
- Fl_File_Icon::DIRECTORY, directories.

The documentation for this class was generated from the following files:

- Fl_File_Icon.H
- Fl_File_Icon.cxx
- Fl_File_Icon2.cxx

13.38 Fl_File_Input Class Reference

This widget displays a pathname in a text input field.
`#include <Fl_File_Input.H>`
Inheritance diagram for Fl_File_Input:



Public Member Functions

- Fl_Boxtype **down_box** () const

  Gets the box type used for the navigation bar.
- void down_box (Fl_Boxtype b)

  Sets the box type to use for the navigation bar.
- Fl_File_Input (int X, int Y, int W, int H, const char *L=0)

  Creates a new Fl_File_Input widget using the given position, size, and label string.
- int handle (int event) override

  Handle events in the widget.
- const char * **value** () const

  Returns the current value, which is a pointer to an internal buffer and is valid only until the next event is handled.
- int value (const char *str)

  Sets the value of the widget given a new string value.
- int value (const char *str, int len)

  Sets the value of the widget given a new string value and its length.

Public Member Functions inherited from Fl_Input

- Fl_Input (int, int, int, int, const char *=0)

  Creates a new Fl_Input widget using the given position, size, and label string.
- int handle (int) override

  Handles the specified event.

Public Member Functions inherited from Fl_Input_

- int append (const char *t, int l=0, char keep_selection=0)

  Append text at the end.
- bool can_redo () const

  Check if there is a redo action available.
- bool can_undo () const

  Check if the last operation can be undone.
- int copy (int clipboard)

  Put the current selection into the clipboard.
- int copy_cuts ()

  Copies the yank buffer to the clipboard.
- Fl_Color cursor_color () const

  Gets the color of the cursor.
- void cursor_color (Fl_Color n)

  Sets the color of the cursor.
- int cut ()

  Deletes the current selection.
- int cut (int a, int b)

  Deletes all characters between index `a` and `b`.
- int cut (int n)

  Deletes the next `n` bytes rounded to characters before or after the cursor.
- double dvalue () const

  Returns the widget text interpreted as a floating point number.
- Fl_Input_ (int, int, int, int, const char *=0)

  Creates a new Fl_Input_ widget.
- unsigned int index (int i) const

  Returns the character at index `i`.

- int input_type () const

  Gets the input field type.
- void input_type (int t)

  Sets the input field type.
- int insert (const char *t, int l=0)

  Inserts text at the cursor position.
- int insert_position () const

  Gets the position of the text cursor.
- int insert_position (int p)

  Sets the cursor position and mark.
- int insert_position (int p, int m)

  Sets the index for the cursor and mark.
- int ivalue () const

  Returns the widget text interpreted as a signed integer.
- int mark () const

  Gets the current selection mark.
- int mark (int m)

  Sets the current selection mark.
- int maximum_size () const

  Gets the maximum length of the input field in characters.
- void maximum_size (int m)

  Sets the maximum length of the input field in characters.
- const char * placeholder () const

  Get the placeholder text.
- void placeholder (const char *text)

  Set the placeholder text.
- int position () const
- int position (int p)
- int position (int p, int m)
- int readonly () const

  Gets the read-only state of the input field.
- void readonly (int b)

  Sets the read-only state of the input field.
- int redo ()

  Redo previous undo operation.
- int replace (int b, int e, const char *text, int ilen=0)

  Deletes text from `b` to `e` and inserts the new string `text`.
- void resize (int, int, int, int) override

  Changes the size of the widget.
- int shortcut () const

  Return the shortcut key associated with this widget.
- void shortcut (int s)

  Sets the shortcut key associated with this widget.
- int size () const

  Returns the number of bytes in value().
- void size (int W, int H)

  Sets the width and height of this widget.
- int static_value (const char *)

  Changes the widget text.
- int static_value (const char *, int)

  Changes the widget text.

- int tab_nav () const

  Gets whether the Tab key causes focus navigation in multiline input fields or not.
- void tab_nav (int val)

  Sets whether the Tab key does focus navigation, or inserts tab characters into Fl_Multiline_Input.
- Fl_Color textcolor () const

  Gets the color of the text in the input field.
- void textcolor (Fl_Color n)

  Sets the color of the text in the input field.
- Fl_Font textfont () const

  Gets the font of the text in the input field.
- void textfont (Fl_Font s)

  Sets the font of the text in the input field.
- Fl_Fontsize textsize () const

  Gets the size of the text in the input field.
- void textsize (Fl_Fontsize s)

  Sets the size of the text in the input field.
- int undo ()

  Undoes previous changes to the text buffer.
- const char ∗ value () const

  Returns the text displayed in the widget.
- int value (const char ∗)

  Changes the widget text.
- int value (const char ∗, int)

  Changes the widget text.
- int value (double value)

  Changes the widget text to a floating point number ("%g").
- int value (int value)

  Changes the widget text to a signed integer number.
- int wrap () const

  Gets the word wrapping state of the input field.
- void wrap (int b)

  Sets the word wrapping state of the input field.
- ∼Fl_Input_ ()

  Destroys the widget.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.

- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.

- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

Sets the image to use as part of the widget label when in the active state.

- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.

- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

Sets the user data for this widget.

- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void draw () override

  Draws the file input widget.

Protected Member Functions inherited from Fl_Input

- void draw () override

  Draws the widget.
- int handle_key ()

  Handles a keystroke.
- int handle_rmb ()

  Handle right mouse button down events.

Protected Member Functions inherited from Fl_Input_

- int apply_undo ()

  Apply the current undo/redo operation.
- void drawtext (int, int, int, int)

  Draws the text in the passed bounding box.
- void drawtext (int, int, int, int, bool draw_active)

  Draws the text in the passed bounding box.
- void handle_mouse (int, int, int, int, int keepmark=0)

  Handles mouse clicks and mouse moves.

- int handletext (int e, int, int, int, int)

  Handles all kinds of text field related events.
- int line_end (int i) const

  Finds the end of a line.
- int line_start (int i) const

  Finds the start of a line.
- int **linesPerPage** ()
- void **maybe_do_callback** (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
- int up_down_position (int, int keepmark=0)

  Moves the cursor to the column given by `up_down_pos`.
- int word_end (int i) const

  Finds the end of a word.
- int word_start (int i) const

  Finds the start of a word.
- int **xscroll** () const
- int **yscroll** () const
- void **yscroll** (int yOffset)

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Static Public Attributes inherited from Fl_Input

- static const char * **copy_menu_text** = "Copy"

  [this text may be customized at run-time]
- static const char * **cut_menu_text** = "Cut"

  [this text may be customized at run-time]
- static const char * **paste_menu_text** = "Paste"

  [this text may be customized at run-time]

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.38.1 Detailed Description

This widget displays a pathname in a text input field.
A navigation bar located above the input field allows the user to navigate upward in the directory tree. You may want to handle FL_WHEN_CHANGED events for tracking text changes and also FL_WHEN_RELEASE for button release when changing to parent dir. FL_WHEN_RELEASE callback won't be called if the directory clicked is the same as the current one.

Figure 13.17 Fl_File_Input

Note

As all Fl_Input derived objects, Fl_File_Input may call its callback when losing focus (see FL_UNFOCUS) to update its state like its cursor shape. One resulting side effect is that you should call clear_changed() early in your callback to avoid reentrant calls if you plan to show another window or dialog box in the callback.

13.38.2 Constructor & Destructor Documentation

13.38.2.1 Fl_File_Input()

```
Fl_File_Input::Fl_File_Input (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

Creates a new Fl_File_Input widget using the given position, size, and label string.
The default boxtype is FL_DOWN_BOX.

Parameters

in	*X,Y,W,H*	position and size of the widget
in	*L*	widget label, default is no label

13.38.3 Member Function Documentation

13.38.3.1 down_box()

```
void Fl_File_Input::down_box (
            Fl_Boxtype b )  [inline]
```

Sets the box type to use for the navigation bar.

13.38.3.2 draw()

```
void Fl_File_Input::draw (
            void  )  [override], [protected], [virtual]
```

Draws the file input widget.
Implements Fl_Widget.

13.38.3.3 handle()

```
int Fl_File_Input::handle (
            int event )  [override], [virtual]
```

Handle events in the widget.
Return non zero if event is handled.

Parameters

in	*event*	

Reimplemented from Fl_Widget.

13.38.3.4 value() [1/2]

```
int Fl_File_Input::value (
            const char * str )
```

Sets the value of the widget given a new string value.
Returns non 0 on success.

Parameters

in	*str*	new string value

13.38.3.5 value() [2/2]

```
int Fl_File_Input::value (
            const char * str,
            int len )
```

Sets the value of the widget given a new string value and its length.
Returns non 0 on success.

Parameters

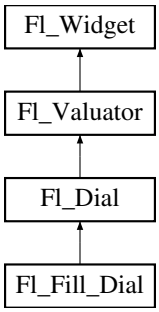
in	*str*	new string value
in	*len*	lengh of value

The documentation for this class was generated from the following files:

- Fl_File_Input.H
- Fl_File_Input.cxx

13.39 Fl_Fill_Dial Class Reference

Draws a dial with a filled arc.
`#include <Fl_Fill_Dial.H>`
Inheritance diagram for Fl_Fill_Dial:



Public Member Functions

- **Fl_Fill_Dial** (int X, int Y, int W, int H, const char ∗L=NULL)

  Creates a filled dial, also setting its type to FL_FILL_DIAL.

Public Member Functions inherited from Fl_Dial

- short angle1 () const

  Sets Or gets the angles used for the minimum and maximum values.
- void **angle1** (short a)

  See short angle1() const.
- short **angle2** () const

  See short angle1() const.
- void **angle2** (short a)

  See short angle1() const.
- void **angles** (short a, short b)

  See short angle1() const.
- Fl_Dial (int x, int y, int w, int h, const char ∗l=0)

  Creates a new Fl_Dial widget using the given position, size, and label string.
- int handle (int) override

  Allow subclasses to handle event based on current position and size.

Public Member Functions inherited from Fl_Valuator

- void **bounds** (double a, double b)

  Sets the minimum (a) and maximum (b) values for the valuator widget.
- double **clamp** (double)

  Clamps the passed value to the valuator range.
- virtual int format (char ∗)

  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.
- virtual std::string format_str ()

  C++11 API for Fl_Valuator::format(char∗ buffer).
- double increment (double, int)

  Adds n times the step value to the passed value.
- double maximum () const

  Gets the maximum value for the valuator.
- void maximum (double a)

  Sets the maximum value for the valuator.
- double minimum () const

  Gets the minimum value for the valuator.
- void minimum (double a)

  Sets the minimum value for the valuator.
- void precision (int digits)

  Sets the step value to 1.0 / 10^{digits}.
- void range (double a, double b)

  Sets the minimum and maximum values for the valuator.
- double round (double)

  Round the passed value to the nearest step increment.
- double step () const

  Gets or sets the step value.
- void **step** (double a, int b)

  See double Fl_Valuator::step() const
- void **step** (double s)

  See double Fl_Valuator::step() const.
- void **step** (int a)

  See double Fl_Valuator::step() const
- double value () const

  Gets the floating point(double) value.
- int value (double)

  Sets the current value.
- ∼**Fl_Valuator** () override

  Destructor is accessible despite protected constructor.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

Returns whether the widget and all of its parents are active.

- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

Sets a widget to accept input.

- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.

- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

Returns a pointer to the parent widget.

- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.

- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Dial

- void draw () override

  Draws dial at current position and size.
- void draw (int X, int Y, int W, int H)

  Draws dial at given position and size.
- int handle (int event, int X, int Y, int W, int H)

  Allows subclasses to handle event based on given position and size.

Protected Member Functions inherited from Fl_Valuator

- Fl_Valuator (int X, int Y, int W, int H, const char *L)

  Creates a new Fl_Valuator widget using the given position, size, and label string.
- void **handle_drag** (double newvalue)

  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.
- void **handle_push** ()

  Stores the current value in the previous value.
- void **handle_release** ()

  Called after an FL_WHEN_RELEASE event is received and before the callback.
- int **horizontal** () const

  Tells if the valuator is an FL_HORIZONTAL one.
- double **previous_value** () const

  Gets the previous floating point value before an event changed it.
- void **set_value** (double v)

  Sets the current floating point value.
- double **softclamp** (double)

  Clamps the value, but accepts v if the previous value is not already out of range.
- virtual void value_damage ()

  Asks for partial redraw.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

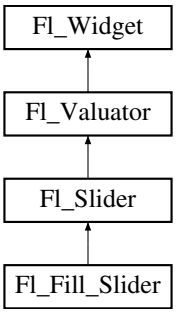
13.39.1 Detailed Description

Draws a dial with a filled arc.
The documentation for this class was generated from the following files:

- Fl_Fill_Dial.H
- Fl_Dial.cxx

13.40 Fl_Fill_Slider Class Reference

Widget that draws a filled horizontal slider, useful as a progress or value meter.
`#include <Fl_Fill_Slider.H>`
Inheritance diagram for Fl_Fill_Slider:



Public Member Functions

- **Fl_Fill_Slider** (int X, int Y, int W, int H, const char ∗L=0)

  Creates the slider from its position,size and optional title.

Public Member Functions inherited from Fl_Slider

- void bounds (double a, double b)

  Sets the minimum (a) and maximum (b) values for the valuator widget.
- Fl_Slider (int X, int Y, int W, int H, const char ∗L=0)

  Creates a new Fl_Slider widget using the given position, size, and label string.
- **Fl_Slider** (uchar t, int X, int Y, int W, int H, const char ∗L)

  Creates a new Fl_Slider widget using the given type, position, size, and label string.
- int handle (int) override

  Handles the specified event.
- uchar num_ticks () const

  Gets the number of tick marks.
- Scale_Type scale () const

  Gets the scale type.
- void scale (Scale_Type s)

  Sets the scale type.
- int scrollvalue (int pos, int size, int first, int total)

  Sets the size and position of the sliding knob in the box.
- Fl_Boxtype **slider** () const

  Gets the slider box type.
- void **slider** (Fl_Boxtype c)

  Sets the slider box type.
- float **slider_size** () const

  Get the dimensions of the moving piece of slider.
- void slider_size (double v)

  Set the dimensions of the moving piece of slider.
- uchar ticks () const

  Gets the tick mark position.
- void ticks (uchar t, uchar num_ticks=11)

  Sets the tick mark position for Fl_Nice_Slider.

Public Member Functions inherited from Fl_Valuator

- void **bounds** (double a, double b)

  Sets the minimum (a) and maximum (b) values for the valuator widget.
- double **clamp** (double)

  Clamps the passed value to the valuator range.
- virtual int format (char ∗)

  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.
- virtual std::string format_str ()

  C++11 API for Fl_Valuator::format(char∗ buffer).
- double increment (double, int)

  Adds n times the step value to the passed value.
- double maximum () const

  Gets the maximum value for the valuator.
- void maximum (double a)

  Sets the maximum value for the valuator.

- double minimum () const

  Gets the minimum value for the valuator.
- void minimum (double a)

  Sets the minimum value for the valuator.
- void precision (int digits)

  Sets the step value to 1.0 / 10^{digits}.
- void range (double a, double b)

  Sets the minimum and maximum values for the valuator.
- double round (double)

  Round the passed value to the nearest step increment.
- double step () const

  Gets or sets the step value.
- void **step** (double a, int b)

  See double Fl_Valuator::step() const
- void **step** (double s)

  See double Fl_Valuator::step() const.
- void **step** (int a)

  See double Fl_Valuator::step() const
- double value () const

  Gets the floating point(double) value.
- int value (double)

  Sets the current value.
- ~**Fl_Valuator** () override

  Destructor is accessible despite protected constructor.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

Returns an Fl_Window pointer if this widget is an Fl_Window.

- virtual Fl_Window const ∗ **as_window** () const
- void bind_deimage (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗cb, void ∗p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget ∗w) const

  Checks if w is a child of this widget.
- void copy_label (const char ∗new_label)

  Sets the current label.
- void copy_tooltip (const char ∗text)

Sets the current tooltip text.

- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().

- const char ∗ label () const

  Gets the current label text.
- void label (const char ∗text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char ∗b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group ∗ parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group ∗p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

Sends the specified event to the widget.

- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.

- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Public Types inherited from Fl_Slider

- enum Scale_Type : uchar { LINEAR_SCALE = 0 , LOG_SCALE }

  Bitset for tick mark positions for Fl_Nice_Slider.
- enum Tick_Position : uchar {
  TICKS_NONE = 0x00 , TICKS_BELOW = 0x01 , TICKS_ABOVE = 0x02 , TICKS_LEFT = 0x01 ,
  TICKS_RIGHT = 0x02 }

  Bitset for tick mark positions for Fl_Nice_Slider.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget ∗widget, void ∗data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char ∗t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char ∗, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,

AUTO_DELETE_USER_DATA = 1<<23 ,
MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
USERFLAG1 = 1<<31 }

flags possible values enumeration.

Protected Member Functions inherited from Fl_Slider

- void draw () override

  Draws the widget.
- void **draw** (int, int, int, int)
- void draw_ticks (const Fl_Rect &r, int S)

  Draw tick marks.
- int **handle** (int, int, int, int, int)
- double increment_lin_log (double v, int n, int range)

  Adds a sensible step value to the passed value.
- double position_to_value (double pos) const

  Converts a normalized position (0.0 to 1.0) to a value based on the current scale type.
- double value_to_position (double val) const

  Converts a value to a normalized position (0.0 to 1.0) based on the current scale type.

Protected Member Functions inherited from Fl_Valuator

- Fl_Valuator (int X, int Y, int W, int H, const char *L)

  Creates a new Fl_Valuator widget using the given position, size, and label string.
- void **handle_drag** (double newvalue)

  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.
- void **handle_push** ()

  Stores the current value in the previous value.
- void **handle_release** ()

  Called after an FL_WHEN_RELEASE event is received and before the callback.
- int **horizontal** () const

  Tells if the valuator is an FL_HORIZONTAL one.
- double **previous_value** () const

  Gets the previous floating point value before an event changed it.
- void **set_value** (double v)

  Sets the current floating point value.
- double **softclamp** (double)

  Clamps the value, but accepts v if the previous value is not already out of range.
- virtual void value_damage ()

  Asks for partial redraw.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

Draws a box of type t, of color c at the position X,Y and size W,H.

- void draw_focus () const

  Draws a focus rectangle around the widget.

- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.

- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.

- void draw_label () const

  Draws the widget's label at the defined label position.

- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.

- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.

- unsigned int **flags** () const

  Gets the widget flags mask.

- void h (int v)

  Internal use only.

- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.

- void w (int v)

  Internal use only.

- void x (int v)

  Internal use only.

- void y (int v)

  Internal use only.

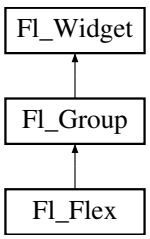
13.40.1 Detailed Description

Widget that draws a filled horizontal slider, useful as a progress or value meter.
The documentation for this class was generated from the following files:

- Fl_Fill_Slider.H
- Fl_Slider.cxx

13.41 Fl_Flex Class Reference

Fl_Flex is a container (layout) widget for one row or one column of widgets.
`#include <Fl_Flex.H>`
Inheritance diagram for Fl_Flex:



Public Types

- enum { VERTICAL = 0 , HORIZONTAL = 1 , COLUMN = 0 , ROW = 1 }

Public Member Functions

- virtual void end ()

  Ends automatic child addition and resizes all children.
- void fixed (Fl_Widget &w, int size)

  Set the horizontal or vertical size of a child widget.
- int fixed (Fl_Widget *w) const

  Return whether the given widget has a fixed size or resizes dynamically.
- void fixed (Fl_Widget *w, int size)

  Set the horizontal or vertical size of a child widget.
- Fl_Flex (int direction)

  Construct a new Fl_Flex widget specifying its layout.
- Fl_Flex (int w, int h, int direction)

  Construct a new Fl_Flex widget specifying its layout and size.
- Fl_Flex (int X, int Y, int W, int H, const char *L=0)

  Construct a new Fl_Flex widget with the given position, size, and label.
- Fl_Flex (int x, int y, int w, int h, int direction)

  Construct a new Fl_Flex widget specifying its layout, position, and size.
- int gap () const

  Return the gap size of the widget.
- void gap (int g)

  Set the gap size of the widget.
- int horizontal () const

  Returns non-zero (true) if Fl_Flex alignment is horizontal (row mode).
- void layout ()

  Calculates the layout of the widget and redraws it.
- int margin () const

  Returns the left margin size of the widget.
- int margin (int *left, int *top, int *right, int *bottom) const

  Returns all (four) margin sizes of the widget.
- void margin (int left, int top, int right, int bottom)

  Set the margin sizes at all four edges of the Fl_Flex widget.
- void margin (int m, int g=-1)

  Set the margin and optionally the gap size of the widget.
- bool need_layout () const

  Returns whether layout calculation is required.
- void need_layout (int set)

  Set or reset the request to calculate the layout of children.
- void resize (int x, int y, int w, int h) override

  Resize the container and calculate all child positions and sizes.
- int spacing () const

  Gets the number of extra pixels of blank space that are added between the children.
- void spacing (int i)

  Sets the number of extra pixels of blank space that are added between the children.

Public Member Functions inherited from Fl_Group

- Fl_Widget *& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget *o)

  See void Fl_Group::add(Fl_Widget &w)
- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget *const * array () const

  Returns a pointer to the internal array of children.
- Fl_Group const * as_group () const override
- Fl_Group * as_group () override

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const

  Returns the n'th child.
- int **children** () const

  Returns how many child widgets the group has.
- void clear ()

  Deletes all child widgets from memory recursively.
- unsigned int clip_children () const

  Returns the current clipping mode.
- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)

  Removes the widget at `index` from the group and deletes it.
- void end ()

  Exactly the same as current(this->parent()).
- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*
- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.
- void **forms_end** ()

  This is for forms compatibility only.
- int handle (int) override

  Handles the specified event.
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

  This does insert(w, find(before)).
- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.

- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.
- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- void resize (int, int, int, int) override

  Resizes the Fl_Group widget and all of its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.

- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

Sets the image to use as part of the widget label when in the inactive state.

- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.

- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

Changes the size of the widget.

- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- virtual int alloc_size (int size) const

  Return new size to be allocated for array of fixed size widgets.
- void draw () override

  Draw the widget.
- void **init** (int t=VERTICAL)
- void on_remove (int) override

  Allow derived groups to act when a child widget is removed from the group.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()

  Returns the internal array of widget sizes and positions.
- void draw () override

  Draws the widget.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget *, int)

  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)

  Allow derived groups to act when a widget is moved within the group.
- int * sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

Draws the label in an arbitrary bounding box.

- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()

  Returns the currently active group.
- static void current (Fl_Group *g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.41.1 Detailed Description

Fl_Flex is a container (layout) widget for one row or one column of widgets.
It provides flexible positioning of its children either in one row or in one column.
Fl_Flex is designed to be as simple as possible. You can set individual widget sizes or let Fl_Flex position and size the widgets to fit in the container. All "flexible" (i.e. non-fixed size) widgets are assigned the same width or height, respectively. For details see below.
You can set the margins **around** all children at the inner side of the box frame (if any). Fl_Flex supports setting different margin sizes on top, bottom, left, and right sides. The default margin size is 0 on all edges of the container.
You can set the gap size **between** all children. The gap size is always the same between all of its children. This is similar to the 'spacing' of Fl_Pack. The default gap size is 0.
Fl_Flex can either consist of a single row, i.e. `type(Fl_Flex::HORIZONTAL)` or a single column, i.e. `type(Fl_Flex::VERTICAL)`. The default value is Fl_Flex::VERTICAL for consistency with Fl_Pack but you can use `type()` to assign a row (Fl_Flex::HORIZONTAL) layout.
If type() == Fl_Flex::HORIZONTAL widgets are resized horizontally to fit in the container and their height is the full Fl_Flex height minus border size and margins. You can set a fixed widget width by using fixed().
If type() == Fl_Flex::VERTICAL widgets are resized vertically to fit in the container and their width is the full Fl_Flex width minus border size and margins. You can set a fixed widget height by using fixed().
To create arbitrary spacing you can use invisible boxes of flexible or fixed sizes (see example below).
Alternate constructors let you specify the layout as Fl_Flex::HORIZONTAL or Fl_Flex::VERTICAL directly. Fl_Flex::ROW is an alias of Fl_Flex::HORIZONTAL and Fl_Flex::COLUMN is an alias of Fl_Flex::VERTICAL.
The default box type is FL_NO_BOX as inherited from Fl_Group. You **may** need to set a box type with a solid background depending on your layout.
Important: You should always make sure that the Fl_Flex container cannot be resized smaller than its designed minimal size. This can usually be done by setting a size_range() on the window as shown in the example below. Fl_Flex does not take care of sensible sizes. If it is resized too small the behavior is undefined, i.e. widgets may overlap and/or shrink to zero size.
Hint: In many cases Fl_Flex can be used as a drop-in replacement for Fl_Pack. This is the recommended single row/column container since FLTK 1.4.0. Its resizing behavior is much more predictable than that of Fl_Pack which *"resizes itself to shrink-wrap itself around all of the children"*.
Fl_Flex containers can be nested so you can create flexible layouts with multiple columns and rows. However, if your UI design is more complex you may want to use Fl_Grid instead.
Example:

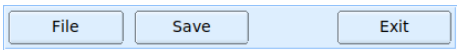


Figure 13.18 Fl_Flex

Example code:

```
#include <FL/Fl.H>
#include <FL/Fl_Double_Window.H>
#include <FL/Fl_Flex.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Button.H>

int main(int argc, char **argv) {
  Fl_Double_Window window(410, 40, "Simple Fl_Flex Demo");
  Fl_Flex flex(5, 5, 400, 30, Fl_Flex::HORIZONTAL);
  Fl_Button b1(0, 0, 0, 0, "File");
  Fl_Button b2(0, 0, 0, 0, "Save");
  Fl_Box    bx(0, 0, 0, 0);
  Fl_Button b3(0, 0, 0, 0, "Exit");
  flex.fixed(bx, 60); // set fix width of invisible box
  flex.gap(10);
  flex.end();
  window.resizable(flex);
  window.end();
  window.size_range(300, 30);
  window.show(argc, argv);
  return Fl::run();
}
```

Since

1.4.0

13.41.2 Member Enumeration Documentation

13.41.2.1 anonymous enum

```
anonymous enum
```

Enumerator

VERTICAL	vertical layout (one column)
HORIZONTAL	horizontal layout (one row)
COLUMN	alias for VERTICAL
ROW	alias for HORIZONTAL

13.41.3 Constructor & Destructor Documentation

13.41.3.1 Fl_Flex() [1/4]

```
Fl_Flex::Fl_Flex (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

Construct a new Fl_Flex widget with the given position, size, and label.
You can set `type(Fl_Flex::HORIZONTAL)` or `type(Fl_Flex::VERTICAL)`. The default is `type(Fl_Flex::VERTICAL)`.
Alternate constructors let you specify the layout as Fl_Flex::HORIZONTAL or Fl_Flex::VERTICAL directly. Fl_Flex::ROW is an alias of Fl_Flex::HORIZONTAL and Fl_Flex::COLUMN is an alias of Fl_Flex::VERTICAL.

Parameters

in	*X,Y*	position
in	*W,H*	size (width and height)
in	*L*	label (optional)

See also

Fl_Flex::Fl_Flex(int direction)

Fl_Flex::Fl_Flex(int w, int h, int direction)

Fl_Flex::Fl_Flex(int x, int y, int w, int h, int direction)

Fl_Flex::Fl_Flex(int x, int y, int w, int h, const char *L)

13.41.3.2 Fl_Flex() [2/4]

```
Fl_Flex::Fl_Flex (
            int direction )
```

Construct a new Fl_Flex widget specifying its layout.
Use Fl_Flex::HORIZONTAL (aka Fl_Flex::ROW) or Fl_Flex::VERTICAL (aka Fl_Flex::COLUMN) as the `direction` argument.
This constructor sets the position and size to (0, 0, 0, 0) which is suitable for nested Fl_Flex widgets. Use one of the other constructors to set the desired position and size as well.

Parameters

in	*direction*	horizontal (row) or vertical (column) layout

See also

Fl_Flex::Fl_Flex(int w, int h, int direction)

Fl_Flex::Fl_Flex(int x, int y, int w, int h, int direction)

Fl_Flex::Fl_Flex(int x, int y, int w, int h, const char *L)

13.41.3.3 Fl_Flex() [3/4]

```
Fl_Flex::Fl_Flex (
            int w,
            int h,
            int direction )
```

Construct a new Fl_Flex widget specifying its layout and size.
Use Fl_Flex::HORIZONTAL (aka Fl_Flex::ROW) or Fl_Flex::VERTICAL (aka Fl_Flex::COLUMN) as the `direction` argument.
This constructor sets the position to (x = 0, y = 0) which is suitable for nested Fl_Flex widgets. Use one of the other constructors to set the desired position as well.

Parameters

in	*w,h*	widget size
in	*direction*	horizontal (row) or vertical (column) layout

See also

Fl_Flex::Fl_Flex(int direction)

Fl_Flex::Fl_Flex(int x, int y, int w, int h, int direction)

Fl_Flex::Fl_Flex(int x, int y, int w, int h, const char *L)

13.41.3.4 Fl_Flex() [4/4]

```
Fl_Flex::Fl_Flex (
            int x,
            int y,
            int w,
            int h,
            int direction )
```

Construct a new Fl_Flex widget specifying its layout, position, and size.
Use Fl_Flex::HORIZONTAL (aka Fl_Flex::ROW) or Fl_Flex::VERTICAL (aka Fl_Flex::COLUMN) as the `direction` argument.
This constructor sets the position and size of the widget which is suitable for top level Fl_Flex widgets but does not set a widget label. Use Fl_Widget::label() to set one if desired.

Parameters

in	*x,y*	widget position
in	*w,h*	widget size
in	*direction*	horizontal (row) or vertical (column) layout

See also

Fl_Flex::Fl_Flex(int direction)

Fl_Flex::Fl_Flex(int w, int h, int direction)

Fl_Flex::Fl_Flex(int x, int y, int w, int h, const char *L)

13.41.4 Member Function Documentation

13.41.4.1 alloc_size()

```
int Fl_Flex::alloc_size (
            int size ) const  [protected], [virtual]
```

Return new size to be allocated for array of fixed size widgets.
This method is called when the array of fixed size widgets needs to be expanded. The current `size` is provided (size can be 0). The default method adds 8 to the current size.
This can be used in derived classes to change the allocation strategy. Note that this method only `queries` the new size which shall be allocated but does not allocate the memory.

Parameters

in	*size*	current size

Returns

int new size (to be allocated)

13.41.4.2 draw()

```
void Fl_Flex::draw (
            void )  [override], [protected], [virtual]
```

Draw the widget.
This will finally calculate the layout of the widget and of all its children if necessary and draw the widget.
Some changes of included children may require a new layout to be calculated. If this is the case the user may need to call layout() to make sure everything is calculated properly.

See also

layout()

Implements Fl_Widget.

13.41.4.3 end()

```
void Fl_Flex::end ( )  [virtual]
```

Ends automatic child addition and resizes all children.
This marks the Fl_Flex widget as changed (need_layout(1)) which forces the widget to calculate its layout depending on all children and whether they have been assigned fix sizes or not right before it is drawn.

See also

need_layout(int)

draw()

13.41.4.4 fixed() [1/3]

```
void Fl_Flex::fixed (
            Fl_Widget & w,
            int size )  [inline]
```

Set the horizontal or vertical size of a child widget.

Parameters

in	*w*	widget to be affected
in	*size*	width (Fl_Flex::HORIZONTAL) or height (Fl_Flex::VERTICAL)

See also

fixed(Fl_Widget *w, int size)

13.41.4.5 fixed() [2/3]

```
int Fl_Flex::fixed (
            Fl_Widget * w ) const
```

Return whether the given widget has a fixed size or resizes dynamically.

Parameters

in	*w*	widget

Returns

whether the widget has a fixed size

Return values

1	the widget has a fixed size
0	the widget resizes dynamically

13.41.4.6 fixed() [3/3]

```
void Fl_Flex::fixed (
            Fl_Widget * child,
            int size )
```

Set the horizontal or vertical size of a child widget.

This sets either the width or height of a child widget, depending on the type() of the Fl_Flex container (Fl_Flex::HORIZONTAL or Fl_Flex::VERTICAL). The other dimension is set to the full width or height of the Fl_Flex widget minus border and margin sizes.

This can be used to set a fixed widget width or height of children of Fl_Flex so they are not resized dynamically.

If `size` is 0 (zero) or negative the widget size is reset to flexible size.

Parameters

in	*child*	widget to be affected
in	*size*	width (Fl_Flex::HORIZONTAL) or height (Fl_Flex::VERTICAL)

13.41.4.7 gap() [1/2]

```
int Fl_Flex::gap ( ) const  [inline]
```

Return the gap size of the widget.

Returns

gap size between all child widgets.

13.41.4.8 gap() [2/2]

```
void Fl_Flex::gap (
            int g ) [inline]
```

Set the gap size of the widget.
The gap size is some free space **between** child widgets. The size must be >= 0. Negative values are clamped to 0.

Parameters

in	*g*	gap size

13.41.4.9 horizontal()

```
int Fl_Flex::horizontal ( ) const [inline]
```

Returns non-zero (true) if Fl_Flex alignment is horizontal (row mode).

Returns

non-zero if Fl_Flex alignment is horizontal

Return values

1	if type() == Fl_Flex::HORIZONTAL
0	if type() == Fl_Flex::VERTICAL

See class Fl_Flex documentation for details.

13.41.4.10 layout()

```
void Fl_Flex::layout ( )
```

Calculates the layout of the widget and redraws it.
If you change widgets in the Fl_Flex container you should call this method to force recalculation of child widget sizes and positions. This can be useful (necessary) if you hide(), show(), add() or remove() children.
Call this method if you need to recalculate widget positions for usage in an algorithm that places widgets at certain positions or when you need to display (show) or hide one or more children depending on the current layout (for instance a side bar).
This method also calls redraw() on the Fl_Flex widget.

13.41.4.11 margin() [1/4]

```
int Fl_Flex::margin ( ) const [inline]
```

Returns the left margin size of the widget.
This returns the **left** margin of the widget which is not necessarily the same as all other margins.

Note

This method is useful if you never set different margin sizes.

See also

int margin(int *left, int *top, int *right, int *bottom) to get all four margin values.

Returns

size of left margin.

13.41.4.12 margin() [2/4]

```
int Fl_Flex::margin (
            int * left,
            int * top,
            int * right,
            int * bottom ) const  [inline]
```

Returns all (four) margin sizes of the widget.
All margin sizes are returned in the given arguments. If any argument is NULL the respective value is not returned.

Parameters

in	*left*	returns left margin if not NULL
in	*top*	returns top margin if not NULL
in	*right*	returns right margin if not NULL
in	*bottom*	returns bottom margin if not NULL

Returns

whether all margins are equal

Return values

1	all margins have the same size
0	at least one margin has a different size

13.41.4.13 margin() [3/4]

```
void Fl_Flex::margin (
            int left,
            int top,
            int right,
            int bottom )  [inline]
```

Set the margin sizes at all four edges of the Fl_Flex widget.
The margin is the free space inside the widget border **around** all child widgets.
You must use all four parameters of this method to set the four margins in the order left, top, right, bottom.
Negative values are set to 0 (zero).
To set all margins to equal sizes, use margin(int m) which sets all four margins to the same size.

Parameters

in	*left,top,right,bottom*	margin sizes, must be >= 0

See also

margin(int, int)

13.41.4.14 margin() [4/4]

```
void Fl_Flex::margin (
            int m,
            int g = -1 )  [inline]
```

Set the margin and optionally the gap size of the widget.
This method can be used to set both the margin and the gap size.
If you don't use the second parameter g or supply a negative value the gap size is not changed.

The margin is the free space inside the widget border **around** all child widgets.
This method sets the margin to the same size at all four edges of the Fl_Flex widget.
The gap size `g` is the free space **between** child widgets. Negative values do not change the gap value. This is the default if this argument is omitted.

Parameters

in	*m*	margin size, must be >= 0
in	*g*	gap size (ignored, if negative)

See also

gap(int)

13.41.4.15 need_layout() [1/2]

```
bool Fl_Flex::need_layout ( ) const  [inline]
```

Returns whether layout calculation is required.
This should rarely be needed by user code. Used internally in draw().

13.41.4.16 need_layout() [2/2]

```
void Fl_Flex::need_layout (
            int set )  [inline]
```

Set or reset the request to calculate the layout of children.
This is intended for internal use but can also be used by user code to request layout calculation before the widget is drawn.
Call this if you changed attributes or sizes of children to ensure that the layout is calculated properly. Changing other Fl_Flex attributes or resizing the widget does this automatically.

Note

Never call this with '`set` == 0' because this would defeat its purpose to recalculate the layout before the widget is drawn.

13.41.4.17 on_remove()

```
void Fl_Flex::on_remove (
            int index )  [override], [protected], [virtual]
```

Allow derived groups to act when a child widget is removed from the group.
Widgets derived from Fl_Group may store additional data for their children. Overriding this method will allow derived classes to remove these data structures just before the child is removed.

Parameters

index	remove the child at this position

Reimplemented from Fl_Group.

13.41.4.18 resize()

```
void Fl_Flex::resize (
            int x,
            int y,
            int w,
            int h )  [override], [virtual]
```

Resize the container and calculate all child positions and sizes.

Parameters

in	*x,y*	position
in	*w,h*	width and height

Reimplemented from Fl_Widget.

13.41.4.19 spacing() [1/2]

```
int Fl_Flex::spacing ( ) const  [inline]
```

Gets the number of extra pixels of blank space that are added between the children.
This method is the same as 'int gap()' and is defined to enable using Fl_Flex as a drop-in replacement of Fl_Pack.

See also

int gap()

13.41.4.20 spacing() [2/2]

```
void Fl_Flex::spacing (
            int i )  [inline]
```

Sets the number of extra pixels of blank space that are added between the children.
This method is the same as 'gap(int)' and is defined to enable using Fl_Flex as a drop-in replacement of Fl_Pack.

See also

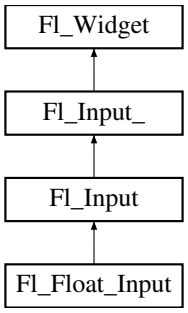
void gap(int)

The documentation for this class was generated from the following files:

- Fl_Flex.H
- Fl_Flex.cxx

13.42 Fl_Float_Input Class Reference

The Fl_Float_Input class is a subclass of Fl_Input that only allows the user to type floating point numbers (sign, digits, decimal point, more digits, 'E' or 'e', sign, digits).
`#include <Fl_Float_Input.H>`
Inheritance diagram for Fl_Float_Input:



Public Member Functions

- Fl_Float_Input (int X, int Y, int W, int H, const char *l=0)

  Creates a new Fl_Float_Input widget using the given position, size, and label string.

Public Member Functions inherited from Fl_Input

- Fl_Input (int, int, int, int, const char *=0)

  Creates a new Fl_Input widget using the given position, size, and label string.
- int handle (int) override

  Handles the specified event.

Public Member Functions inherited from Fl_Input_

- int append (const char *t, int l=0, char keep_selection=0)

  Append text at the end.
- bool can_redo () const

  Check if there is a redo action available.
- bool can_undo () const

  Check if the last operation can be undone.
- int copy (int clipboard)

  Put the current selection into the clipboard.
- int copy_cuts ()

  Copies the yank buffer to the clipboard.
- Fl_Color cursor_color () const

  Gets the color of the cursor.
- void cursor_color (Fl_Color n)

  Sets the color of the cursor.
- int cut ()

  Deletes the current selection.
- int cut (int a, int b)

  Deletes all characters between index `a` and `b`.
- int cut (int n)

  Deletes the next `n` bytes rounded to characters before or after the cursor.
- double dvalue () const

  Returns the widget text interpreted as a floating point number.
- Fl_Input_ (int, int, int, int, const char *=0)

  Creates a new Fl_Input_ widget.
- unsigned int index (int i) const

  Returns the character at index `i`.
- int input_type () const

  Gets the input field type.
- void input_type (int t)

  Sets the input field type.
- int insert (const char *t, int l=0)

  Inserts text at the cursor position.
- int insert_position () const

  Gets the position of the text cursor.
- int insert_position (int p)

  Sets the cursor position and mark.
- int insert_position (int p, int m)

  Sets the index for the cursor and mark.
- int ivalue () const

  Returns the widget text interpreted as a signed integer.
- int mark () const

  Gets the current selection mark.
- int mark (int m)

Sets the current selection mark.

- int maximum_size () const

  Gets the maximum length of the input field in characters.
- void maximum_size (int m)

  Sets the maximum length of the input field in characters.
- const char * placeholder () const

  Get the placeholder text.
- void placeholder (const char *text)

  Set the placeholder text.
- int position () const
- int position (int p)
- int position (int p, int m)
- int readonly () const

  Gets the read-only state of the input field.
- void readonly (int b)

  Sets the read-only state of the input field.
- int redo ()

  Redo previous undo operation.
- int replace (int b, int e, const char *text, int ilen=0)

  Deletes text from `b` to `e` and inserts the new string `text`.
- void resize (int, int, int, int) override

  Changes the size of the widget.
- int shortcut () const

  Return the shortcut key associated with this widget.
- void shortcut (int s)

  Sets the shortcut key associated with this widget.
- int size () const

  Returns the number of bytes in value().
- void size (int W, int H)

  Sets the width and height of this widget.
- int static_value (const char *)

  Changes the widget text.
- int static_value (const char *, int)

  Changes the widget text.
- int tab_nav () const

  Gets whether the Tab key causes focus navigation in multiline input fields or not.
- void tab_nav (int val)

  Sets whether the Tab key does focus navigation, or inserts tab characters into Fl_Multiline_Input.
- Fl_Color textcolor () const

  Gets the color of the text in the input field.
- void textcolor (Fl_Color n)

  Sets the color of the text in the input field.
- Fl_Font textfont () const

  Gets the font of the text in the input field.
- void textfont (Fl_Font s)

  Sets the font of the text in the input field.
- Fl_Fontsize textsize () const

  Gets the size of the text in the input field.
- void textsize (Fl_Fontsize s)

  Sets the size of the text in the input field.
- int undo ()

Undoes previous changes to the text buffer.

- const char * value () const

  Returns the text displayed in the widget.
- int value (const char *)

  Changes the widget text.
- int value (const char *, int)

  Changes the widget text.
- int value (double value)

  Changes the widget text to a floating point number ("%g").
- int value (int value)

  Changes the widget text to a signed integer number.
- int wrap () const

  Gets the word wrapping state of the input field.
- void wrap (int b)

  Sets the word wrapping state of the input field.
- ~Fl_Input_ ()

  Destroys the widget.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.

- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

Internal use only.

- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.

- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

Enables keyboard focus navigation with this widget.

- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.

- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Static Public Attributes inherited from Fl_Input

- static const char * **copy_menu_text** = "Copy"

  [this text may be customized at run-time]
- static const char * **cut_menu_text** = "Cut"

  [this text may be customized at run-time]
- static const char * **paste_menu_text** = "Paste"

  [this text may be customized at run-time]

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Input

- void draw () override

  Draws the widget.
- int handle_key ()

  Handles a keystroke.

- int handle_rmb ()

  Handle right mouse button down events.

Protected Member Functions inherited from Fl_Input_

- int apply_undo ()

  Apply the current undo/redo operation.
- void drawtext (int, int, int, int)

  Draws the text in the passed bounding box.
- void drawtext (int, int, int, int, bool draw_active)

  Draws the text in the passed bounding box.
- void handle_mouse (int, int, int, int, int keepmark=0)

  Handles mouse clicks and mouse moves.
- int handletext (int e, int, int, int, int)

  Handles all kinds of text field related events.
- int line_end (int i) const

  Finds the end of a line.
- int line_start (int i) const

  Finds the start of a line.
- int **linesPerPage** ()
- void **maybe_do_callback** (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
- int up_down_position (int, int keepmark=0)

  Moves the cursor to the column given by `up_down_pos`.
- int word_end (int i) const

  Finds the end of a word.
- int word_start (int i) const

  Finds the start of a word.
- int **xscroll** () const
- int **yscroll** () const
- void **yscroll** (int yOffset)

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

Draws the label in an arbitrary bounding box.

- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

13.42.1 Detailed Description

The Fl_Float_Input class is a subclass of Fl_Input that only allows the user to type floating point numbers (sign, digits, decimal point, more digits, 'E' or 'e', sign, digits).

13.42.2 Constructor & Destructor Documentation

13.42.2.1 Fl_Float_Input()

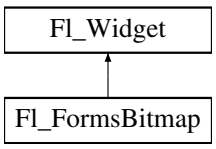
```
Fl_Float_Input::Fl_Float_Input (
            int X,
            int Y,
            int W,
            int H,
            const char * l = 0 )
```

Creates a new Fl_Float_Input widget using the given position, size, and label string.
The default boxtype is FL_DOWN_BOX.
Inherited destructor destroys the widget and any value associated with it.
The documentation for this class was generated from the following files:

- Fl_Float_Input.H
- Fl_Input.cxx

13.43 Fl_FormsBitmap Class Reference

Forms compatibility Bitmap Image Widget.
`#include <Fl_FormsBitmap.H>`
Inheritance diagram for Fl_FormsBitmap:



Public Member Functions

- Fl_Bitmap * **bitmap** () const

  Gets a the current associated Fl_Bitmap objects.
- void **bitmap** (Fl_Bitmap *B)

  Sets a new bitmap.
- **Fl_FormsBitmap** (Fl_Boxtype, int, int, int, int, const char *=0)

  Creates a bitmap widget from a box type, position, size and optional label specification.
- void set (int W, int H, const uchar *bits)

  Sets a new bitmap bits with size W,H.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.

- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

Sets the image to use as part of the widget label when in the inactive state.

- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual int handle (int event)

  Handles the specified event.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.

- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

Sets whether the widget's label uses '&' to indicate shortcuts.

- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.

- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void draw () override

  Draws the bitmap and its associated box.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.43.1 Detailed Description

Forms compatibility Bitmap Image Widget.

13.43.2 Member Function Documentation

13.43.2.1 draw()

```
void Fl_FormsBitmap::draw (
            void ) [override], [protected], [virtual]
```

Draws the bitmap and its associated box.
Implements Fl_Widget.

13.43.2.2 set()

```
void Fl_FormsBitmap::set (
            int W,
            int H,
            const uchar * bits )
```

Sets a new bitmap bits with size W,H.
Deletes the previous one.
The documentation for this class was generated from the following files:

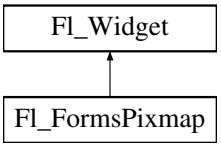
- Fl_FormsBitmap.H
- forms_bitmap.cxx

13.44 Fl_FormsPixmap Class Reference

Forms pixmap drawing routines.

`#include <Fl_FormsPixmap.H>`

Inheritance diagram for Fl_FormsPixmap:



Public Member Functions

- Fl_FormsPixmap (Fl_Boxtype t, int X, int Y, int W, int H, const char *L=0)

  Creates a new Fl_FormsPixmap widget using the given box type, position, size and label string.
- Fl_Pixmap * **Pixmap** () const

  Get the internal pixmap pointer.
- void Pixmap (Fl_Pixmap *B)

  Set the internal pixmap pointer to an existing pixmap.
- void set (char *const *bits)

  Set/create the internal pixmap using raw data.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.

- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual int handle (int event)

  Handles the specified event.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

Sets the current label pointer.

- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.

- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

Modifies keyboard focus navigation.

- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void draw () override

  Draws the widget.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

Sets a flag in the flags mask.

- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.44.1 Detailed Description

Forms pixmap drawing routines.

13.44.2 Constructor & Destructor Documentation

13.44.2.1 Fl_FormsPixmap()

```
Fl_FormsPixmap::Fl_FormsPixmap (
            Fl_Boxtype t,
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

Creates a new Fl_FormsPixmap widget using the given box type, position, size and label string.

Parameters

in	*t*	box type

Parameters

in	*X,Y,W,H*	position and size
in	*L*	widget label, default is no label

13.44.3 Member Function Documentation

13.44.3.1 draw()

```
void Fl_FormsPixmap::draw ( ) [override], [protected], [virtual]
```

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();                 // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

13.44.3.2 Pixmap()

```
void Fl_FormsPixmap::Pixmap (
            Fl_Pixmap * B ) [inline]
```

Set the internal pixmap pointer to an existing pixmap.

Parameters

in	*B*	existing pixmap

13.44.3.3 set()

```
void Fl_FormsPixmap::set (
            char *const * bits )
```

Set/create the internal pixmap using raw data.

Parameters

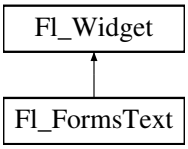
in	*bits*	raw data

The documentation for this class was generated from the following files:

- Fl_FormsPixmap.H
- forms_pixmap.cxx

13.45 Fl_FormsText Class Reference

Inheritance diagram for Fl_FormsText:



Public Member Functions

- **Fl_FormsText** (Fl_Boxtype b, int X, int Y, int W, int H, const char *l=0)

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

Sets the current callback function for the widget.

- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.

- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual int handle (int event)

  Handles the specified event.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

Sets the font size in pixels.

- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.

- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void draw () override

  Draws the widget.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 , AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.45.1 Member Function Documentation

13.45.1.1 draw()

`void Fl_FormsText::draw ( ) [override], [protected], [virtual]`

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();                 // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

The documentation for this class was generated from the following file:

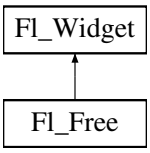
- forms.H

13.46 Fl_Free Class Reference

Emulation of the Forms "free" widget.

`#include <Fl_Free.H>`

Inheritance diagram for Fl_Free:



Public Member Functions

- Fl_Free (uchar t, int X, int Y, int W, int H, const char *L, FL_HANDLEPTR hdl)

  Create a new Fl_Free widget with type, position, size, label and handler.
- int handle (int e) override

  Handles the specified event.
- ~**Fl_Free** ()

  The destructor will call the handle function with the event FL_FREE_MEM.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ **as_gl_window** () const
- virtual Fl_Group ∗ as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const ∗ **as_group** () const
- virtual Fl_Window ∗ as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ **as_window** () const
- void bind_deimage (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗cb, void ∗p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

Checks if the widget value changed since the last callback.

- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.

- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

Sets width ww and height hh accordingly with the label size.

- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.

- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void draw () override

  Draws the widget.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 , AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.46.1 Detailed Description

Emulation of the Forms "free" widget.
This emulation allows the free demo to run, and appears to be useful for porting programs written in Forms which use the free widget or make subclasses of the Forms widgets.
There are five types of free, which determine when the handle function is called:

- `FL_NORMAL_FREE` normal event handling.
- `FL_SLEEPING_FREE` deactivates event handling (widget is inactive).
- `FL_INPUT_FREE` accepts FL_FOCUS events.
- `FL_CONTINUOUS_FREE` sets a timeout callback 100 times a second and provides an FL_STEP event. This has obvious detrimental effects on machine performance.
- `FL_ALL_FREE` same as FL_INPUT_FREE and FL_CONTINUOUS_FREE.

13.46.2 Constructor & Destructor Documentation

13.46.2.1 Fl_Free()

```
Fl_Free::Fl_Free (
            uchar t,
            int X,
            int Y,
            int W,
            int H,
            const char * L,
            FL_HANDLEPTR hdl )
```

Create a new Fl_Free widget with type, position, size, label and handler.

Parameters

in	*t*	type
in	*X,Y,W,H*	position and size
in	*L*	widget label
in	*hdl*	handler function

The constructor takes both the type and the handle function. The handle function should be declared as follows:

```
int handle_function(Fl_Widget *w,
                    int       event,
                    float     event_x,
```

```
                    float      event_y,
                    char       key)
```

This function is called from the handle() method in response to most events, and is called by the draw() method.
The event argument contains the event type:

```
// old event names for compatibility:
#define FL_MOUSE              FL_DRAG
#define FL_DRAW               0
#define FL_STEP               9
#define FL_FREEMEM            12
#define FL_FREEZE             FL_UNMAP
#define FL_THAW               FL_MAP
```

13.46.3 Member Function Documentation

13.46.3.1 draw()

```
void Fl_Free::draw ( )  [override], [protected], [virtual]
```

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();                 // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

13.46.3.2 handle()

```
int Fl_Free::handle (
            int event )  [override], [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to *override* the behavior of the base class. This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to *filter* event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you *consumed* the event by returning 1 even if you didn't *do* anything with it.

Parameters

in	*event*	the kind of event received

Return values

0	if the event was not used or understood
1	if the event was used and can be deleted

See also

Fl_Event

Reimplemented from Fl_Widget.
The documentation for this class was generated from the following files:

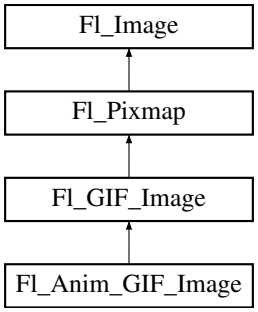
- Fl_Free.H
- forms_free.cxx

13.47 Fl_GIF_Image Class Reference

The Fl_GIF_Image class supports loading, caching, and drawing of Compuserve GIF[SM] images.

`#include <Fl_GIF_Image.H>`

Inheritance diagram for Fl_GIF_Image:



Classes

- struct GIF_FRAME

Public Member Functions

- Fl_GIF_Image (const char *filename)

  This constructor loads a GIF image from the given file.
- Fl_GIF_Image (const char *imagename, const unsigned char *data)

  This constructor loads a GIF image from memory (deprecated).
- Fl_GIF_Image (const char *imagename, const unsigned char *data, const size_t length)

  This constructor loads a GIF image from memory.

Public Member Functions inherited from Fl_Pixmap

- int **cache_h** () const
- int **cache_w** () const
- void color_average (Fl_Color c, float i) override

  The color_average() method averages the colors in the image with the provided FLTK color value.
- Fl_Image * **copy** () const
- Fl_Image * copy (int W, int H) const override

  Creates a resized copy of the image.
- void desaturate () override

  The desaturate() method converts an image to grayscale.
- void **draw** (int X, int Y)
- void draw (int X, int Y, int W, int H, int cx=0, int cy=0) override

  Draws the image to the current drawing surface with a bounding box.
- Fl_Pixmap (char *const *D)

  The constructors create a new pixmap from the specified XPM data.
- Fl_Pixmap (const char *const *D)

  The constructors create a new pixmap from the specified XPM data.
- Fl_Pixmap (const uchar *const *D)

  The constructors create a new pixmap from the specified XPM data.
- Fl_Pixmap (uchar *const *D)

  The constructors create a new pixmap from the specified XPM data.
- void label (Fl_Menu_Item *m) override

  This method is an obsolete way to set the image attribute of a menu item.

- void label (Fl_Widget *w) override

  This method is an obsolete way to set the image attribute of a widget or menu item.
- void uncache () override

  If the image has been cached for display, delete the cache data.
- virtual ~**Fl_Pixmap** ()

  The destructor frees all memory and server resources that are used by the pixmap.

Public Member Functions inherited from Fl_Image

- virtual class Fl_Shared_Image * as_shared_image ()

  Returns whether an image is an Fl_Shared_Image or not.
- Fl_Image * copy () const

  Creates a copy of the image in the same size.
- int count () const

  Returns the number of data values associated with the image.
- int d () const

  Returns the image depth.
- const char *const * data () const

  Returns a pointer to the current image data array.
- int **data_h** () const

  Returns the height of the image data.
- int **data_w** () const

  Returns the width of the image data.
- void draw (int X, int Y)

  Draws the image to the current drawing surface.
- int fail () const

  Returns a value that is not 0 if there is currently no image available.
- Fl_Image (int W, int H, int D)

  The constructor creates an empty image with the specified width, height, and depth.
- int h () const

  Returns the current image drawing height in FLTK units.
- void inactive ()

  The inactive() method calls color_average(FL_BACKGROUND_COLOR, 0.33f) to produce an image that appears grayed out.
- int ld () const

  Returns the current line data size in bytes.
- virtual void release ()

  Releases an Fl_Image - the same as '`delete this`'.
- virtual void scale (int width, int height, int proportional=1, int can_expand=0)

  Sets the drawing size of the image.
- int w () const

  Returns the current image drawing width in FLTK units.
- virtual ~**Fl_Image** ()

  The destructor is a virtual method that frees all memory used by the image.

Static Public Member Functions

- static bool **is_animated** (const char *name_)

Static Public Member Functions inherited from Fl_Image

- static Fl_Labeltype **define_FL_IMAGE_LABEL** ()
- static Fl_RGB_Scaling **RGB_scaling** ()

  Returns the currently used RGB image scaling method.
- static void RGB_scaling (Fl_RGB_Scaling)

  Sets the RGB image scaling method used for copy(int, int).
- static Fl_RGB_Scaling **scaling_algorithm** ()

  Gets what algorithm is used when resizing a source image to draw it.
- static void scaling_algorithm (Fl_RGB_Scaling algorithm)

  Sets what algorithm is used when resizing a source image to draw it.

Static Public Attributes

- static bool animate = false

  Sets how the shared image core routine should treat animated GIF files.

Static Public Attributes inherited from Fl_Image

- static const int **ERR_FILE_ACCESS** = -2
- static const int **ERR_FORMAT** = -3
- static const int **ERR_MEMORY_ACCESS** = -4
- static const int **ERR_NO_IMAGE** = -1
- static bool **register_images_done** = false

  True after fl_register_images() was called, false before.

Protected Member Functions

- **Fl_GIF_Image** ()

  The default constructor creates an empty GIF image.
- **Fl_GIF_Image** (const char ∗filename, bool anim)
- **Fl_GIF_Image** (const char ∗imagename, const unsigned char ∗data, const size_t length, bool anim)
- void **load** (const char ∗filename, bool anim)

  The protected load() methods are used by Fl_Anim_GIF_Image to request loading of animated GIF's.
- void **load** (const char ∗imagename, const unsigned char ∗data, const size_t length, bool anim)
- void **load_gif_** (class Fl_Image_Reader &rdr, bool anim=false)
- virtual void **on_extension_data** (GIF_FRAME &)
- virtual void **on_frame_data** (GIF_FRAME &)

Protected Member Functions inherited from Fl_Pixmap

- void **measure** ()

Protected Member Functions inherited from Fl_Image

- void **d** (int D)

  Sets the current image depth.
- void data (const char ∗const ∗p, int c)

  Sets the current data pointer and count of pointers in the array.
- void draw_empty (int X, int Y)

  The protected method draw_empty() draws a box with an X in it.
- int draw_scaled (int X, int Y, int W, int H)

  Draw the image to the current drawing surface rescaled to a given width and height.
- void h (int H)

  Sets the height of the image data.

- void ld (int LD)

  Sets the current line data size in bytes.
- void w (int W)

  Sets the width of the image data.

Additional Inherited Members

Public Attributes inherited from Fl_Pixmap

- int **alloc_data**

Static Protected Member Functions inherited from Fl_Image

- static void **labeltype** (const Fl_Label *lo, int lx, int ly, int lw, int lh, Fl_Align la)
- static void **measure** (const Fl_Label *lo, int &lw, int &lh)

13.47.1 Detailed Description

The Fl_GIF_Image class supports loading, caching, and drawing of Compuserve GIF[SM] images.
The class loads the first image and supports transparency.

13.47.2 Constructor & Destructor Documentation

13.47.2.1 Fl_GIF_Image() [1/3]

```
Fl_GIF_Image::Fl_GIF_Image (
            const char * filename )
```

This constructor loads a GIF image from the given file.
If a GIF image is animated, Fl_GIF_Image will only read and display the first frame of the animation.
The destructor frees all memory and server resources that are used by the image.
Use Fl_Image::fail() to check if Fl_GIF_Image failed to load. fail() returns ERR_FILE_ACCESS if the file could not be opened or read, ERR_FORMAT if the GIF format could not be decoded, and ERR_NO_IMAGE if the image could not be loaded for another reason.

Parameters

in	*filename*	a full path and name pointing to a GIF image file.

See also

Fl_GIF_Image::Fl_GIF_Image(const char *imagename, const unsigned char *data, const long length)

13.47.2.2 Fl_GIF_Image() [2/3]

```
Fl_GIF_Image::Fl_GIF_Image (
            const char * imagename,
            const unsigned char * data )
```

This constructor loads a GIF image from memory (deprecated).

Deprecated Please use Fl_GIF_Image(const char *imagename, const unsigned char *data, const size_t length) instead.

Note

Buffer overruns will not be checked.

This constructor should not be used because the caller can't supply the memory size and the image reader can't check for "end of memory" errors.

Note

A new constructor with parameter `length` is available since FLTK 1.4.0.

Parameters

in	*imagename*	A name given to this image or NULL
in	*data*	Pointer to the start of the GIF image in memory.

See also

Fl_GIF_Image(const char *filename)

Fl_GIF_Image(const char *imagename, const unsigned char *data, const size_t length)

13.47.2.3 Fl_GIF_Image() [3/3]

```
Fl_GIF_Image::Fl_GIF_Image (
            const char * imagename,
            const unsigned char * data,
            const size_t length )
```

This constructor loads a GIF image from memory.
Construct an image from a block of memory inside the application. Fluid offers "binary data" chunks as a great way to add image data into the C++ source code. `imagename` can be `NULL`. If a name is given, the image is added to the list of shared images and will be available by that name.
If a GIF image is animated, Fl_GIF_Image will only read and display the first frame of the animation.
The destructor frees all memory and server resources that are used by the image.
The third parameter `length` is used to test for buffer overruns, i.e. truncated images.
Use Fl_Image::fail() to check if Fl_GIF_Image failed to load. fail() returns ERR_FILE_ACCESS if the file could not be opened or read, ERR_FORMAT if the GIF format could not be decoded, and ERR_NO_IMAGE if the image could not be loaded for another reason.

Parameters

in	*imagename*	A name given to this image or NULL
in	*data*	Pointer to the start of the GIF image in memory.
in	*length*	Length of the GIF image in memory.

See also

Fl_GIF_Image::Fl_GIF_Image(const char *filename)

Fl_Shared_Image

Since

1.4.0

13.47.3 Member Data Documentation

13.47.3.1 animate

```
bool Fl_GIF_Image::animate = false  [static]
```

Sets how the shared image core routine should treat animated GIF files.
The default is to treat them as ordinary GIF's e.g. it creates a Fl_GIF_Image object. If this variable is set, then an animated GIF object Fl_Anim_GIF_Image is created.
The documentation for this class was generated from the following files:

- Fl_GIF_Image.H

- Fl_Anim_GIF_Image.cxx
- Fl_GIF_Image.cxx

13.48 Fl_Gl_Choice Class Reference

Public Member Functions

- **Fl_Gl_Choice** (int m, const int ∗alistp, Fl_Gl_Choice ∗n)

Friends

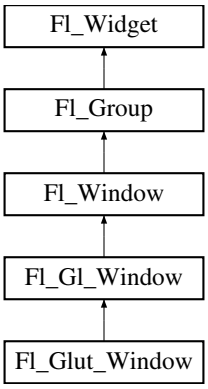
- class **Fl_Gl_Window_Driver**

The documentation for this class was generated from the following file:

- Fl_Gl_Choice.H

13.49 Fl_Gl_Window Class Reference

The Fl_Gl_Window widget sets things up so OpenGL works.
`#include <Fl_Gl_Window.H>`
Inheritance diagram for Fl_Gl_Window:



Public Member Functions

- Fl_Gl_Window const ∗ as_gl_window () const override
- Fl_Gl_Window ∗ as_gl_window () override

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- int **can_do** () const

  Returns non-zero if the hardware supports the current OpenGL mode.
- int can_do_overlay ()

  Returns true if the hardware overlay is possible.
- GLContext context () const

  Returns a pointer to the window's OpenGL rendering context.
- void context (GLContext, int destroy_flag=0)

  Sets a pointer to the GLContext that this window is using.
- char context_valid () const

  Will only be set if the OpenGL context is created or recreated.
- void **context_valid** (char v)

  See char Fl_Gl_Window::context_valid() const.
- Fl_Gl_Window (int W, int H, const char ∗l=0)

  Creates a new Fl_Gl_Window widget using the given size, and label string.

- Fl_Gl_Window (int X, int Y, int W, int H, const char *l=0)

  Creates a new Fl_Gl_Window widget using the given position, size, and label string.
- void flush () override

  Forces the window to be drawn, this window is also made current and calls draw().
- int handle (int) override

  Handle some FLTK events as needed.
- void hide () override

  Hides the window and destroys the OpenGL context.
- void **hide_overlay** ()

  Hides the window if it is not this window, does nothing in Windows.
- void **invalidate** ()

  The invalidate() method turns off valid() and is equivalent to calling value(0).
- void make_current ()

  The make_current() method selects the OpenGL context for the widget.
- void make_overlay_current ()

  Selects the OpenGL context for the widget's overlay.
- Fl_Mode mode () const

  Returns the current OpenGL capabilities of the window.
- int mode (const int *a)

  Set the OpenGL capabilities of the window using platform-specific data.
- int mode (int a)

  Set or change the OpenGL capabilities of the window.
- void ortho ()

  Sets the projection so 0,0 is in the lower left of the window and each pixel is 1 unit wide/tall.
- int pixel_h ()

  Gives the window height in OpenGL pixels.
- int pixel_w ()

  Gives the window width in OpenGL pixels.
- float pixels_per_unit ()

  The number of pixels per FLTK unit of length for the window.
- void redraw_overlay ()

  Causes draw_overlay() to be called at a later time.
- void resize (int, int, int, int) override

  Changes the size or position of the widget.
- void show () override

  Makes a widget visible.
- void **show** (int a, char **b)

  *Same as Fl_Window::show(int a, char **b)*
- void swap_buffers ()

  The swap_buffers() method swaps the back and front buffers.
- int swap_interval () const

  Gets the rate at which the GL windows swaps buffers.
- void swap_interval (int)

  Sets the rate at which the GL windows swaps buffers.
- char valid () const

  Is turned off when FLTK creates a new context for this window or when the window resizes, and is turned on after draw() is called.
- void **valid** (char v)

  See char Fl_Gl_Window::valid() const.
- ~**Fl_Gl_Window** ()

  The destructor removes the widget and destroys the OpenGL context associated with it.

Public Member Functions inherited from Fl_Window

- void allow_expand_outside_parent ()

  Allow this subwindow to expand outside the area of its parent window.
- virtual class Fl_Double_Window * as_double_window ()

  Return non-null if this is an Fl_Double_Window object.
- virtual class Fl_Overlay_Window * as_overlay_window ()

  Return non-null if this is an Fl_Overlay_Window object.
- Fl_Window const * as_window () const override
- Fl_Window * as_window () override

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- unsigned int **border** () const

  Returns whether the window possesses a border.
- void border (int b)

  Sets whether or not the window manager border is around the window.
- void clear_border ()

  Fast inline function to turn the window manager border off.
- void clear_modal_states ()

  Clears the "modal" flags and converts a "modal" or "non-modal" window back into a "normal" window.
- void **copy_label** (const char *a)

  Sets the window titlebar label to a copy of a character string.
- void cursor (const Fl_RGB_Image *, int, int)

  Changes the cursor for this window using the provided image as cursor's shape.
- void cursor (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)

  For back compatibility only.
- void cursor (Fl_Cursor)

  Changes the cursor for this window.
- int decorated_h () const

  Returns the window height including any window title bar and any frame added by the window manager.
- int decorated_w () const

  Returns the window width including any frame added by the window manager.
- void default_cursor (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)

  For back compatibility only.
- void default_cursor (Fl_Cursor)

  Sets the default window cursor.
- void **draw_backdrop** ()

  Draw the background image if one is set and is aligned inside.
- Fl_Window (int w, int h, const char *title=0)

  Creates a window from the given width `w`, height `h`, and `title`.
- Fl_Window (int x, int y, int w, int h, const char *title=0)

  Creates a window from the given position (x, y), size (w, h) and title.
- void free_position ()

  Undoes the effect of a previous resize() or show() so that the next time show() is called the window manager is free to position the window.
- void fullscreen ()

  Makes the window completely fill one or more screens, without any window manager border visible.
- unsigned int **fullscreen_active** () const

  Returns non zero if FULLSCREEN flag is set, 0 otherwise.
- void **fullscreen_off** ()

  Turns off any side effects of fullscreen()
- void **fullscreen_off** (int X, int Y, int W, int H)

Turns off any side effects of fullscreen() and does resize(X,Y,W,H).

- void fullscreen_screens (int top, int bottom, int left, int right)

  Sets which screens should be used when this window is in fullscreen mode.
- uchar get_size_range (int ∗minw, int ∗minh, int ∗maxw=NULL, int ∗maxh=NULL, int ∗dw=NULL, int ∗dh=NULL, int ∗aspect=NULL) const

  Gets the allowable range to which the user can resize this window.
- int handle (int) override

  Handles the specified event.
- void hide () override

  Removes the window from the screen.
- void **hotspot** (const Fl_Widget &p, int offscreen=0)

  See void Fl_Window::hotspot(int x, int y, int offscreen = 0)
- void **hotspot** (const Fl_Widget ∗, int offscreen=0)

  See void Fl_Window::hotspot(int x, int y, int offscreen = 0)
- void hotspot (int x, int y, int offscreen=0)

  Positions the window so that the mouse is pointing at the given position, or at the center of the given widget, which may be the window itself.
- const void ∗ icon () const

  Gets the current icon window target dependent data.
- void icon (const Fl_RGB_Image ∗)

  Sets or resets a single window icon.
- void icon (const void ∗ic)

  Platform-specific method to set the window icon usable on Windows and X11 only.
- void iconize ()

  Iconifies the window.
- const char ∗ **iconlabel** () const

  See void Fl_Window::iconlabel(const char∗)
- void **iconlabel** (const char ∗)

  Sets the icon label.
- void icons (const Fl_RGB_Image ∗[ ], int)

  Sets the window icons.
- void icons (HICON big_icon, HICON small_icon)

  Sets the window icons using HICON handles (Windows platform only).
- const char ∗ **label** () const

  See void Fl_Window::label(const char∗)
- void **label** (const char ∗)

  Sets the window title bar label.
- void **label** (const char ∗label, const char ∗iconlabel)

  Sets the icon label.
- void make_current ()

  Sets things up so that the drawing functions in <FL/fl_draw.H> will go into this window.
- void maximize ()

  Maximizes a top-level window to its current screen.
- unsigned int **maximize_active** () const

  Returns whether the window is currently maximized.
- unsigned int **menu_window** () const

  Returns true if this window is a menu window.
- unsigned int modal () const

  Returns true if this window is modal.

- unsigned int **non_modal** () const

  Returns true if this window is modal or non-modal.
- fl_uintptr_t os_id () const

  Returns a platform-specific identification of a shown window, or 0 if not shown.
- unsigned int **override** () const

  Returns non zero if OVERRIDE flag is set, 0 otherwise.
- void resize (int X, int Y, int W, int H) override

  Changes the size and position of the window.
- int screen_num () const

  The number of the screen containing the mapped window.
- void screen_num (int screen_num)

  Set the number of the screen where to map the window.
- void set_menu_window ()

  Marks the window as a menu window.
- void set_modal ()

  A "modal" window, when shown(), will prevent any events from being delivered to other windows in the same program, and will also remain on top of the other windows (if the X window manager supports the "transient for" property).
- void set_non_modal ()

  A "non-modal" window (terminology borrowed from Microsoft Windows) acts like a modal() one in that it remains on top, but it has no effect on event delivery.
- void **set_override** ()

  Activates the flags NOBORDER|OVERRIDE.
- void set_tooltip_window ()

  Marks the window as a tooltip window.
- const Fl_Image * shape () const

  Returns the image controlling the window shape or NULL.
- void shape (const Fl_Image &b)

  Set the window's shape with an Fl_Image.
- void shape (const Fl_Image *img)

  Assigns a non-rectangular shape to the window.
- void show () override

  Puts the window on the screen.
- void show (int argc, char **argv)

  Puts the window on the screen with show() and parses command-line arguments.
- int shown () const

  Returns non-zero if show() has been called (but not hide()).
- void size_range (int minw, int minh, int maxw=0, int maxh=0, int dw=0, int dh=0, int aspect=0)

  Sets the allowable range to which the user can resize this window.
- unsigned int **tooltip_window** () const

  Returns true if this window is a tooltip window.
- void un_maximize ()

  Returns a previously maximized top-level window to its previous size.
- void wait_for_expose ()

  Waits for the window to be displayed after calling show().
- int **x_root** () const

  Gets the x position of the window on the screen.
- const char * xclass () const

  Returns the xclass for this window, or a default.
- void xclass (const char *c)

  Sets the xclass for this window.
- int **y_root** () const

  Gets the y position of the window on the screen.
- virtual ~Fl_Window ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Group

- Fl_Widget *& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget *o)

  See void Fl_Group::add(Fl_Widget &w)
- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget *const * array () const

  Returns a pointer to the internal array of children.
- Fl_Group const * as_group () const override
- Fl_Group * as_group () override

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const

  Returns the n'th child.
- int **children** () const

  Returns how many child widgets the group has.
- void clear ()

  Deletes all child widgets from memory recursively.
- unsigned int clip_children () const

  Returns the current clipping mode.
- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)

  Removes the widget at `index` from the group and deletes it.
- void end ()

  Exactly the same as current(this->parent()).
- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*
- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.
- void **forms_end** ()

  This is for forms compatibility only.
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

  This does insert(w, find(before)).
- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.
- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.

- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.

- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

Calls the widget callback function with arbitrary arguments.

- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.

- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group ∗ parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group ∗p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

Finds the x/y offset of the current widget relative to the top-level window.

- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Static Public Member Functions

- static int can_do (const int ∗m)

  Returns non-zero if the hardware supports the given OpenGL mode.
- static int **can_do** (int m)

  Returns non-zero if the hardware supports the given OpenGL mode.

Static Public Member Functions inherited from Fl_Window

- static Fl_Window ∗ current ()

  Returns the last window that was made current.
- static void **default_callback** (Fl_Window ∗, void ∗v)

  Back compatibility: Sets the default callback v for win to call on close event.
- static void default_icon (const Fl_RGB_Image ∗)

  Sets a single default window icon.
- static void default_icons (const Fl_RGB_Image ∗[ ], int)

  Sets the default window icons.
- static void default_icons (HICON big_icon, HICON small_icon)

  Sets the default window icons (Windows platform only).
- static const char ∗ default_xclass ()

  Returns the default xclass.
- static void default_xclass (const char ∗)

  Sets the default window xclass.
- static bool is_a_rescale ()

  Returns true when a window is being rescaled.
- static char show_next_window_iconic ()

  Returns the static flag whether the next window should be opened iconified.
- static void show_next_window_iconic (char stat)

  Sets a static flag whether the next window should be opened iconified.

Static Public Member Functions inherited from Fl_Group

- static Fl_Group ∗ current ()

  Returns the currently active group.
- static void current (Fl_Group ∗g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget ∗widget, void ∗data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char ∗t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char ∗, const bool require_alt=false)

  Returns true if the given text `t` *contains the entered '&x' shortcut.*

Protected Member Functions

- void draw () override

  Draws the Fl_Gl_Window.
- void draw_begin ()

  Supports drawing to an Fl_Gl_Window with the FLTK 2D drawing API.
- void draw_end ()

  To be used as a match for a previous call to Fl_Gl_Window::draw_begin().

Protected Member Functions inherited from Fl_Window

- void default_size_range ()

  Protected method to calculate the default size range of a window.
- void draw () override

  Draws the widget.
- int force_position () const

  Returns the internal state of the window's `FORCE_POSITION` flag.
- void force_position (int force)

  Sets an internal flag that tells FLTK and the window manager to honor position requests.
- void free_icons ()

  Deletes all icons previously attached to the window.
- int is_resizable ()

  Protected method to determine whether a window is resizable.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()

  Returns the internal array of widget sizes and positions.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget *, int)

  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)

  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)

  Allow derived groups to act when a child widget is removed from the group.
- int * sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.

- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Friends

- class **Fl_Gl_Window_Driver**

Additional Inherited Members

Public Types inherited from Fl_Window

- typedef struct HICON__ * **HICON**

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Static Protected Attributes inherited from Fl_Window

- static Fl_Window * current_

  Stores the last window that was made current.

13.49.1 Detailed Description

The Fl_Gl_Window widget sets things up so OpenGL works.
It also keeps an OpenGL "context" for that window, so that changes to the lighting and projection may be reused between redraws. Fl_Gl_Window also flushes the OpenGL streams and swaps buffers after draw() returns.
OpenGL hardware typically provides some overlay bit planes, which are very useful for drawing UI controls atop your 3D graphics. If the overlay hardware is not provided, FLTK tries to simulate the overlay. This works pretty well if your graphics are double buffered, but not very well for single-buffered.
Please note that the FLTK drawing and clipping functions will not work inside an Fl_Gl_Window. All drawing should be done using OpenGL calls exclusively.

See also

OpenGL and support of HighDPI displays

Note

FLTK 1.4 introduces a driver system for graphic calls. It is now possible to add a selection of widgets to an OpenGL window. The widgets will draw on top of any OpenGL rendering. The number of supported widgets will increase as the driver development improves. Program test/cube.cxx illustrates how to do that.

FLTK expects that when an Fl_Gl_Window is a child of a parent Fl_Window, the child window lies entirely inside its parent window. If that's not the case, what happens to the part of the GL subwindow which leaks outside its parent is undefined and susceptible to be platform-specific.

13.49.2 Constructor & Destructor Documentation

13.49.2.1 Fl_Gl_Window() [1/2]

```
Fl_Gl_Window::Fl_Gl_Window (
            int W,
            int H,
            const char * l = 0 )  [inline]
```

Creates a new Fl_Gl_Window widget using the given size, and label string.
The default boxtype is FL_NO_BOX. The default mode is FL_RGB|FL_DOUBLE|FL_DEPTH.

13.49.2.2 Fl_Gl_Window() [2/2]

```
Fl_Gl_Window::Fl_Gl_Window (
            int X,
            int Y,
            int W,
            int H,
            const char * l = 0 )  [inline]
```

Creates a new Fl_Gl_Window widget using the given position, size, and label string.
The default boxtype is FL_NO_BOX. The default mode is FL_RGB|FL_DOUBLE|FL_DEPTH.

13.49.3 Member Function Documentation

13.49.3.1 as_gl_window() [1/2]

```
Fl_Gl_Window const * Fl_Gl_Window::as_gl_window ( ) const  [inline], [override], [virtual]
```

Reimplemented from Fl_Widget.

13.49.3.2 as_gl_window() [2/2]

```
Fl_Gl_Window * Fl_Gl_Window::as_gl_window ( )  [inline], [override], [virtual]
```

Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
Use this method if you have a widget (pointer) and need to know whether this widget is derived from Fl_Gl_Window.
If it returns non-NULL, then the widget in question is derived from Fl_Gl_Window.

Return values

NULL	if this widget is not derived from Fl_Gl_Window.

Note

This method is provided to avoid dynamic_cast.

See also

Fl_Widget::as_group(), Fl_Widget::as_window()

Reimplemented from Fl_Widget.

13.49.3.3 can_do()

```
static int Fl_Gl_Window::can_do (
            const int * m )  [inline], [static]
```

Returns non-zero if the hardware supports the given OpenGL mode.

See also

Fl_Gl_Window::mode(const int *a)

13.49.3.4 can_do_overlay()

```
int Fl_Gl_Window::can_do_overlay ( )
```

Returns true if the hardware overlay is possible.
If this is false, FLTK will try to simulate the overlay, with significant loss of update speed. Calling this will cause FLTK to open the display.

13.49.3.5 context() [1/2]

```
GLContext Fl_Gl_Window::context ( ) const  [inline]
```

Returns a pointer to the window's OpenGL rendering context.

See also

void context(GLContext c, int destroy_flag)

13.49.3.6 context() [2/2]

```
void Fl_Gl_Window::context (
            GLContext v,
            int destroy_flag = 0 )
```

Sets a pointer to the GLContext that this window is using.
This is a system-dependent structure, but it is portable to copy the context from one window to another. You can also set it to NULL, which will force FLTK to recreate the context the next time make_current() is called, this is useful for getting around bugs in OpenGL implementations.
If *destroy_flag* is true the context will be destroyed by fltk when the window is destroyed, or when the mode() is changed, or the next time context(x) is called.

13.49.3.7 context_valid()

```
char Fl_Gl_Window::context_valid ( ) const  [inline]
```

Will only be set if the OpenGL context is created or recreated.
It differs from Fl_Gl_Window::valid() which is also set whenever the context changes size.

13.49.3.8 draw()

```
void Fl_Gl_Window::draw (
            void ) [override], [protected], [virtual]
```

Draws the Fl_Gl_Window.

You ***must*** *subclass* Fl_Gl_Window and provide an implementation for draw(). You may also provide an implementation of draw_overlay() if you want to draw into the overlay planes. You can avoid reinitializing the viewport and lights and other things by checking valid() at the start of draw() and only doing the initialization if it is false.

The draw() method can *only* use OpenGL calls. Do not attempt to call X, any of the functions in <FL/fl_draw.H>, or glX directly. Do not call gl_start() or gl_finish().

If double-buffering is enabled in the window, the back and front buffers are swapped after this function is completed.

The following pseudo-code shows how to use "if (!valid())" to initialize the viewport:

```
void mywindow::draw() {
 if (!valid()) {
     glViewport(0,0,pixel_w(),pixel_h());
     glFrustum(...) or glOrtho(...)
     ...other initialization...
 }
 if (!context_valid()) {
     ...load textures, etc. ...
 }
 // clear screen
 glClearColor(...);
 glClear(...);
 ... draw your geometry here ...
}
```

Actual example code to clear screen to black and draw a 2D white "X":

```
void mywindow::draw() {
    if (!valid()) {
        glLoadIdentity();
        glViewport(0,0,pixel_w(),pixel_h());
        glOrtho(-w(),w(),-h(),h(),-1,1);
    }
    // Clear screen
    glClear(GL_COLOR_BUFFER_BIT);
    // Draw white 'X'
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_LINE_STRIP); glVertex2f(w(), h()); glVertex2f(-w(),-h()); glEnd();
    glBegin(GL_LINE_STRIP); glVertex2f(w(),-h()); glVertex2f(-w(), h()); glEnd();
}
```

Regular FLTK widgets can be added as children to the Fl_Gl_Window. To correctly overlay the widgets, Fl_Gl_Window::draw() must be called after rendering the main scene.

```
void mywindow::draw() {
  // draw 3d graphics scene
  Fl_Gl_Window::draw();
  // -- or --
  draw_begin();
  Fl_Window::draw();
  // other 2d drawing calls, overlays, etc.
  draw_end();
}
```

Implements Fl_Widget.

Reimplemented in Fl_Glut_Window.

13.49.3.9 draw_begin()

```
void Fl_Gl_Window::draw_begin ( ) [protected]
```

Supports drawing to an Fl_Gl_Window with the FLTK 2D drawing API.

See also

Using FLTK widgets in OpenGL Windows

13.49.3.10 draw_end()

```
void Fl_Gl_Window::draw_end ( ) [protected]
```

To be used as a match for a previous call to Fl_Gl_Window::draw_begin().

See also

Using FLTK widgets in OpenGL Windows

13.49.3.11 flush()

```
void Fl_Gl_Window::flush ( ) [override], [virtual]
```

Forces the window to be drawn, this window is also made current and calls draw().
Reimplemented from Fl_Window.

13.49.3.12 handle()

```
int Fl_Gl_Window::handle (
            int event ) [override], [virtual]
```

Handle some FLTK events as needed.
Reimplemented from Fl_Widget.
Reimplemented in Fl_Glut_Window.

13.49.3.13 hide()

```
void Fl_Gl_Window::hide ( ) [override], [virtual]
```

Hides the window and destroys the OpenGL context.
Reimplemented from Fl_Widget.

13.49.3.14 make_current()

```
void Fl_Gl_Window::make_current ( )
```

The make_current() method selects the OpenGL context for the widget.
It is called automatically prior to the draw() method being called and can also be used to implement feedback and/or selection within the handle() method.

13.49.3.15 make_overlay_current()

```
void Fl_Gl_Window::make_overlay_current ( )
```

Selects the OpenGL context for the widget's overlay.
This method is called automatically prior to the draw_overlay() method being called and can also be used to implement feedback and/or selection within the handle() method.

13.49.3.16 mode() [1/3]

```
Fl_Mode Fl_Gl_Window::mode ( ) const [inline]
```

Returns the current OpenGL capabilities of the window.
Don't use this if capabilities were set through Fl_Gl_Window::mode(const int *a).

13.49.3.17 mode() [2/3]

```
int Fl_Gl_Window::mode (
            const int * a ) [inline]
```

Set the OpenGL capabilities of the window using platform-specific data.

Parameters

a	zero-ending array of platform-specific attributes and attribute values

Unix/Linux platform: attributes are GLX attributes adequate for the 3rd argument of the `glXChooseVisual()` function (e.g., `GLX_DOUBLEBUFFER`, defined by including <GL/glx.h>).

Note

> What attributes are adequate here is subject to change. The preferred, stable public API is Fl_Gl_Window::mode(int a).

Windows platform: this member function is of no use.
Mac OS X platform: attributes belong to the `CGLPixelFormatAttribute` enumeration (defined by including `<OpenGL/OpenGL.h>`, e.g., `kCGLPFADoubleBuffer`) and may be followed by adequate attribute values.

13.49.3.18 mode() [3/3]

```
int Fl_Gl_Window::mode (
            int a ) [inline]
```

Set or change the OpenGL capabilities of the window.
The value can be any of the following OR'd together:

- `FL_RGB` - RGB color (not indexed)
- `FL_RGB8` - RGB color with at least 8 bits of each color
- `FL_INDEX` - Indexed mode
- `FL_SINGLE` - not double buffered
- `FL_DOUBLE` - double buffered
- `FL_ACCUM` - accumulation buffer
- `FL_ALPHA` - alpha channel in color
- `FL_DEPTH` - any depth buffer (or set FL_DEPTH32 for at least 32 bits)
- `FL_DEPTH32` - depth buffer with at least 32 bits
- `FL_STENCIL` - stencil buffer
- `FL_MULTISAMPLE` - multisample antialiasing
- `FL_OPENGL3` - use OpenGL version 3.0 or more.

FL_RGB and FL_SINGLE have a value of zero, so they are "on" unless you give FL_INDEX or FL_DOUBLE.
If the desired combination cannot be done, FLTK will try turning off FL_MULTISAMPLE. If this also fails the show() will call Fl::error() and not show the window.
You can change the mode while the window is displayed. This is most useful for turning double-buffering on and off. Under X this will cause the old X window to be destroyed and a new one to be created. If this is a top-level window this will unfortunately also cause the window to blink, raise to the top, and be de-iconized, and the xid() will change, possibly breaking other code. It is best to make the GL window a child of another window if you wish to do this!
mode() must not be called within draw() since it changes the current context.
The FL_OPENGL3 flag is recommended to use OpenGL version 3 or more. This flag is required (not just recommended) if GL 3.0 is in use and at least one of these conditions applies:

- the program runs on the macOS platform;
- the Fl_Gl_Window has child widgets.

See more details in Using OpenGL 3.0 (or higher versions).

Version

the `FL_OPENGL3` flag appeared in version 1.3.4

13.49.3.19 ortho()

```
void Fl_Gl_Window::ortho ( )
```

Sets the projection so 0,0 is in the lower left of the window and each pixel is 1 unit wide/tall.
If you are drawing 2D images, your draw() method may want to call this if valid() is false.

13.49.3.20 pixel_h()

```
int Fl_Gl_Window::pixel_h ( ) [inline]
```

Gives the window height in OpenGL pixels.

When an Fl_Gl_Window is mapped to a HighDPI display, the value given by Fl_Gl_Window::h() which is expressed in FLTK units, may differ from the window height in pixels. Calls to OpenGL functions expecting pixel values (*e.g.*, `glViewport`) should therefore use pixel_h() rather than h(). Method pixel_h() detects when the GUI is rescaled or when the window has been moved between low and high resolution displays and automatically adjusts the returned value.

Version

1.3.4

13.49.3.21 pixel_w()

```
int Fl_Gl_Window::pixel_w ( ) [inline]
```

Gives the window width in OpenGL pixels.

When an Fl_Gl_Window is mapped to a HighDPI display, the value given by Fl_Gl_Window::w() which is expressed in FLTK units, may differ from the window width in pixels. Calls to OpenGL functions expecting pixel values (*e.g.*, `glViewport`) should therefore use pixel_w() rather than w(). Method pixel_w() detects when the GUI is rescaled or when the window has been moved between low and high resolution displays and automatically adjusts the returned value.

Version

1.3.4

13.49.3.22 pixels_per_unit()

```
float Fl_Gl_Window::pixels_per_unit ( )
```

The number of pixels per FLTK unit of length for the window.

This method dynamically adjusts its value when the GUI is rescaled or when the window is moved to/from displays of distinct resolutions. This method is useful, e.g., to convert, in a window's handle() method, the FLTK units returned by Fl::event_x() and Fl::event_y() to the pixel units used by the OpenGL source code.

Version

1.3.4

13.49.3.23 redraw_overlay()

```
void Fl_Gl_Window::redraw_overlay ( )
```

Causes draw_overlay() to be called at a later time.

Initially the overlay is clear. If you want the window to display something in the overlay when it first appears, you must call this immediately after you show() your window.

13.49.3.24 resize()

```
void Fl_Gl_Window::resize (
            int x,
            int y,
            int w,
            int h ) [override], [virtual]
```

Changes the size or position of the widget.

This is a virtual function so that the widget may implement its own handling of resizing. The default version does *not* call the redraw() method, but instead relies on the parent widget to do so because the parent may know a faster way to update the display, such as scrolling from the old position.

Some window managers under X11 call resize() a lot more often than needed. Please verify that the position or size of a widget did actually change before doing any extensive calculations.

position(X, Y) is a shortcut for resize(X, Y, w(), h()), and size(W, H) is a shortcut for resize(x(), y(), W, H).

Parameters

in	*x,y*	new position relative to the parent window
in	*w,h*	new size

See also

position(int,int), size(int,int)

Reimplemented from Fl_Widget.

13.49.3.25 show()

`void Fl_Gl_Window::show ( ) [override], [virtual]`

Makes a widget visible.
An invisible widget never gets redrawn and does not get keyboard or mouse events, but can receive a few other events like FL_SHOW.
The visible() method returns true if the widget is set to be visible. The visible_r() method returns true if the widget and all of its parents are visible. A widget is only visible if visible() is true on it *and all of its parents*.
Changing it will send FL_SHOW or FL_HIDE events to the widget. *Do not change it if the parent is not visible, as this will send false FL_SHOW or FL_HIDE events to the widget*. redraw() is called if necessary on this or the parent.

See also

hide(), visible(), visible_r()

Reimplemented from Fl_Widget.

13.49.3.26 swap_buffers()

`void Fl_Gl_Window::swap_buffers ( )`

The swap_buffers() method swaps the back and front buffers.
It is called automatically after the draw() method is called.

13.49.3.27 swap_interval() [1/2]

`int Fl_Gl_Window::swap_interval ( ) const`

Gets the rate at which the GL windows swaps buffers.
This method can be called after the OpenGL context was created, typically within the user overridden `Fl_Gl_Window::draw()` method that will be overridden by the user.

Note

This method depends highly on the underlying OpenGL contexts and driver implementation. Some drivers return no information, most drivers don't support intervals with multiple frames and return only 0 or 1.

Some drivers have the ability to set the swap interval but no way to query it, hence this method may return -1 even though the interval was set correctly. Conversely a return value greater zero does not guarantee that the driver actually honors the setting.

Returns

an integer greater zero if vertical blanking is taken into account when swapping OpenGL buffers

0 if the vertical blanking is ignored

-1 if the information can not be retrieved

13.49.3.28 swap_interval() [2/2]

```
void Fl_Gl_Window::swap_interval (
            int frames )
```

Sets the rate at which the GL windows swaps buffers.
This method can be called after the OpenGL context was created, typically within the user overridden `Fl_Gl_Window::draw()` method that will be overridden by the user.

Note

This method depends highly on the underlying OpenGL contexts and driver implementation. Most driver seem to accept only 0 and 1 to swap buffer asynchronously or in sync with the vertical blank.

Parameters

in	*frames*	set the number of vertical frame blanks between OpenGL buffer swaps

13.49.3.29 valid()

```
char Fl_Gl_Window::valid ( ) const  [inline]
```

Is turned off when FLTK creates a new context for this window or when the window resizes, and is turned on *after* draw() is called.
You can use this inside your draw() method to avoid unnecessarily initializing the OpenGL context. Just do this:

```
void mywindow::draw() {
 if (!valid()) {
   glViewport(0,0,pixel_w(),pixel_h());
   glFrustum(...);
   ...other initialization...
 }
 if (!context_valid()) {
   ...load textures, etc. ...
 }
 ... draw your geometry here ...
}
```

You can turn valid() on by calling valid(1). You should only do this after fixing the transformation inside a draw() or after make_current(). This is done automatically after draw() returns.
The documentation for this class was generated from the following files:

- Fl_Gl_Window.H
- Fl_Gl_Overlay.cxx
- Fl_Gl_Window.cxx

13.50 Fl_Glut_Bitmap_Font Struct Reference

fltk glut font/size attributes used in the glutXXX functions
`#include <glut.H>`

Public Attributes

- Fl_Font **font**
- Fl_Fontsize **size**

13.50.1 Detailed Description

fltk glut font/size attributes used in the glutXXX functions
The documentation for this struct was generated from the following file:

- glut.H

13.51 Fl_Glut_StrokeChar Struct Reference

Public Attributes

- int **Number**
- GLfloat **Right**
- const Fl_Glut_StrokeStrip * **Strips**

The documentation for this struct was generated from the following file:

- glut.H

13.52 Fl_Glut_StrokeFont Struct Reference

Public Attributes

- const Fl_Glut_StrokeChar ** **Characters**
- GLfloat **Height**
- char * **Name**
- int **Quantity**

The documentation for this struct was generated from the following file:

- glut.H

13.53 Fl_Glut_StrokeStrip Struct Reference

Public Attributes

- int **Number**
- const Fl_Glut_StrokeVertex * **Vertices**

The documentation for this struct was generated from the following file:

- glut.H

13.54 Fl_Glut_StrokeVertex Struct Reference

Public Attributes

- GLfloat **X**
- GLfloat **Y**

The documentation for this struct was generated from the following file:

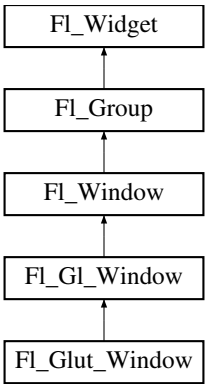
- glut.H

13.55 Fl_Glut_Window Class Reference

GLUT is emulated using this window class and these static variables (plus several more static variables hidden in glut_compatibility.cxx):

```
#include <glut.H>
```

Inheritance diagram for Fl_Glut_Window:



Public Member Functions

- **Fl_Glut_Window** (int w, int h, const char ∗t=0)

  Creates a glut window, registers to the glut windows list.
- **Fl_Glut_Window** (int x, int y, int w, int h, const char ∗t=0)

  Creates a glut window, registers to the glut windows list.
- void **make_current** ()
- ∼**Fl_Glut_Window** ()

  Destroys the glut window, first unregister it from the glut windows list.

Public Member Functions inherited from Fl_Gl_Window

- Fl_Gl_Window const ∗ as_gl_window () const override
- Fl_Gl_Window ∗ as_gl_window () override

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- int **can_do** () const

  Returns non-zero if the hardware supports the current OpenGL mode.
- int can_do_overlay ()

  Returns true if the hardware overlay is possible.
- GLContext context () const

  Returns a pointer to the window's OpenGL rendering context.
- void context (GLContext, int destroy_flag=0)

  Sets a pointer to the GLContext that this window is using.
- char context_valid () const

  Will only be set if the OpenGL context is created or recreated.
- void **context_valid** (char v)

  See char Fl_Gl_Window::context_valid() const.
- Fl_Gl_Window (int W, int H, const char ∗l=0)

  Creates a new Fl_Gl_Window widget using the given size, and label string.
- Fl_Gl_Window (int X, int Y, int W, int H, const char ∗l=0)

  Creates a new Fl_Gl_Window widget using the given position, size, and label string.
- void flush () override

  Forces the window to be drawn, this window is also made current and calls draw().
- void hide () override

  Hides the window and destroys the OpenGL context.
- void **hide_overlay** ()

  Hides the window if it is not this window, does nothing in Windows.
- void **invalidate** ()

  The invalidate() method turns off valid() and is equivalent to calling value(0).
- void make_current ()

The make_current() method selects the OpenGL context for the widget.

- void make_overlay_current ()

  Selects the OpenGL context for the widget's overlay.
- Fl_Mode mode () const

  Returns the current OpenGL capabilities of the window.
- int mode (const int ∗a)

  Set the OpenGL capabilities of the window using platform-specific data.
- int mode (int a)

  Set or change the OpenGL capabilities of the window.
- void ortho ()

  Sets the projection so 0,0 is in the lower left of the window and each pixel is 1 unit wide/tall.
- int pixel_h ()

  Gives the window height in OpenGL pixels.
- int pixel_w ()

  Gives the window width in OpenGL pixels.
- float pixels_per_unit ()

  The number of pixels per FLTK unit of length for the window.
- void redraw_overlay ()

  Causes draw_overlay() to be called at a later time.
- void resize (int, int, int, int) override

  Changes the size or position of the widget.
- void show () override

  Makes a widget visible.
- void **show** (int a, char ∗∗b)

  Same as Fl_Window::show(int a, char ∗∗b)
- void swap_buffers ()

  The swap_buffers() method swaps the back and front buffers.
- int swap_interval () const

  Gets the rate at which the GL windows swaps buffers.
- void swap_interval (int)

  Sets the rate at which the GL windows swaps buffers.
- char valid () const

  Is turned off when FLTK creates a new context for this window or when the window resizes, and is turned on after draw() is called.
- void **valid** (char v)

  See char Fl_Gl_Window::valid() const.
- ∼**Fl_Gl_Window** ()

  The destructor removes the widget and destroys the OpenGL context associated with it.

Public Member Functions inherited from Fl_Window

- void allow_expand_outside_parent ()

  Allow this subwindow to expand outside the area of its parent window.
- virtual class Fl_Double_Window ∗ as_double_window ()

  Return non-null if this is an Fl_Double_Window object.
- virtual class Fl_Overlay_Window ∗ as_overlay_window ()

  Return non-null if this is an Fl_Overlay_Window object.
- Fl_Window const ∗ as_window () const override
- Fl_Window ∗ as_window () override

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- unsigned int **border** () const

Returns whether the window possesses a border.

- void border (int b)

  Sets whether or not the window manager border is around the window.
- void clear_border ()

  Fast inline function to turn the window manager border off.
- void clear_modal_states ()

  Clears the "modal" flags and converts a "modal" or "non-modal" window back into a "normal" window.
- void **copy_label** (const char *a)

  Sets the window titlebar label to a copy of a character string.
- void cursor (const Fl_RGB_Image *, int, int)

  Changes the cursor for this window using the provided image as cursor's shape.
- void cursor (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)

  For back compatibility only.
- void cursor (Fl_Cursor)

  Changes the cursor for this window.
- int decorated_h () const

  Returns the window height including any window title bar and any frame added by the window manager.
- int decorated_w () const

  Returns the window width including any frame added by the window manager.
- void default_cursor (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)

  For back compatibility only.
- void default_cursor (Fl_Cursor)

  Sets the default window cursor.
- void **draw_backdrop** ()

  Draw the background image if one is set and is aligned inside.
- Fl_Window (int w, int h, const char *title=0)

  Creates a window from the given width `w`, height `h`, and `title`.
- Fl_Window (int x, int y, int w, int h, const char *title=0)

  Creates a window from the given position (x, y), size (w, h) and title.
- void free_position ()

  Undoes the effect of a previous resize() or show() so that the next time show() is called the window manager is free to position the window.
- void fullscreen ()

  Makes the window completely fill one or more screens, without any window manager border visible.
- unsigned int **fullscreen_active** () const

  Returns non zero if FULLSCREEN flag is set, 0 otherwise.
- void **fullscreen_off** ()

  Turns off any side effects of fullscreen()
- void **fullscreen_off** (int X, int Y, int W, int H)

  Turns off any side effects of fullscreen() and does resize(X,Y,W,H).
- void fullscreen_screens (int top, int bottom, int left, int right)

  Sets which screens should be used when this window is in fullscreen mode.
- uchar get_size_range (int *minw, int *minh, int *maxw=NULL, int *maxh=NULL, int *dw=NULL, int *dh=NULL, int *aspect=NULL) const

  Gets the allowable range to which the user can resize this window.
- int handle (int) override

  Handles the specified event.
- void hide () override

  Removes the window from the screen.
- void **hotspot** (const Fl_Widget &p, int offscreen=0)

  See void Fl_Window::hotspot(int x, int y, int offscreen = 0)

- void **hotspot** (const Fl_Widget ∗, int offscreen=0)

  See void Fl_Window::hotspot(int x, int y, int offscreen = 0)
- void hotspot (int x, int y, int offscreen=0)

  Positions the window so that the mouse is pointing at the given position, or at the center of the given widget, which may be the window itself.
- const void ∗ icon () const

  Gets the current icon window target dependent data.
- void icon (const Fl_RGB_Image ∗)

  Sets or resets a single window icon.
- void icon (const void ∗ic)

  Platform-specific method to set the window icon usable on Windows and X11 only.
- void iconize ()

  Iconifies the window.
- const char ∗ **iconlabel** () const

  See void Fl_Window::iconlabel(const char∗)
- void **iconlabel** (const char ∗)

  Sets the icon label.
- void icons (const Fl_RGB_Image ∗[ ], int)

  Sets the window icons.
- void icons (HICON big_icon, HICON small_icon)

  Sets the window icons using HICON handles (Windows platform only).
- const char ∗ **label** () const

  See void Fl_Window::label(const char∗)
- void **label** (const char ∗)

  Sets the window title bar label.
- void **label** (const char ∗label, const char ∗iconlabel)

  Sets the icon label.
- void make_current ()

  Sets things up so that the drawing functions in <FL/fl_draw.H> will go into this window.
- void maximize ()

  Maximizes a top-level window to its current screen.
- unsigned int **maximize_active** () const

  Returns whether the window is currently maximized.
- unsigned int **menu_window** () const

  Returns true if this window is a menu window.
- unsigned int modal () const

  Returns true if this window is modal.
- unsigned int **non_modal** () const

  Returns true if this window is modal or non-modal.
- fl_uintptr_t os_id () const

  Returns a platform-specific identification of a shown window, or 0 if not shown.
- unsigned int **override** () const

  Returns non zero if OVERRIDE flag is set, 0 otherwise.
- void resize (int X, int Y, int W, int H) override

  Changes the size and position of the window.
- int screen_num () const

  The number of the screen containing the mapped window.
- void screen_num (int screen_num)

  Set the number of the screen where to map the window.

- void set_menu_window ()

  Marks the window as a menu window.
- void set_modal ()

  A "modal" window, when shown(), will prevent any events from being delivered to other windows in the same program, and will also remain on top of the other windows (if the X window manager supports the "transient for" property).
- void set_non_modal ()

  A "non-modal" window (terminology borrowed from Microsoft Windows) acts like a modal() one in that it remains on top, but it has no effect on event delivery.
- void **set_override** ()

  Activates the flags NOBORDER|OVERRIDE.
- void set_tooltip_window ()

  Marks the window as a tooltip window.
- const Fl_Image ∗ shape () const

  Returns the image controlling the window shape or NULL.
- void shape (const Fl_Image &b)

  Set the window's shape with an Fl_Image.
- void shape (const Fl_Image ∗img)

  Assigns a non-rectangular shape to the window.
- void show () override

  Puts the window on the screen.
- void show (int argc, char ∗∗argv)

  Puts the window on the screen with show() and parses command-line arguments.
- int shown () const

  Returns non-zero if show() has been called (but not hide()).
- void size_range (int minw, int minh, int maxw=0, int maxh=0, int dw=0, int dh=0, int aspect=0)

  Sets the allowable range to which the user can resize this window.
- unsigned int **tooltip_window** () const

  Returns true if this window is a tooltip window.
- void un_maximize ()

  Returns a previously maximized top-level window to its previous size.
- void wait_for_expose ()

  Waits for the window to be displayed after calling show().
- int **x_root** () const

  Gets the x position of the window on the screen.
- const char ∗ xclass () const

  Returns the xclass for this window, or a default.
- void xclass (const char ∗c)

  Sets the xclass for this window.
- int **y_root** () const

  Gets the y position of the window on the screen.
- virtual ∼Fl_Window ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Group

- Fl_Widget ∗& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget ∗o)

  See void Fl_Group::add(Fl_Widget &w)

- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget *const * array () const

  Returns a pointer to the internal array of children.
- Fl_Group const * as_group () const override
- Fl_Group * as_group () override

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const

  Returns the n'th child.
- int **children** () const

  Returns how many child widgets the group has.
- void clear ()

  Deletes all child widgets from memory recursively.
- unsigned int clip_children () const

  Returns the current clipping mode.
- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)

  Removes the widget at `index` from the group and deletes it.
- void end ()

  Exactly the same as current(this->parent()).
- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*
- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.
- void **forms_end** ()

  This is for forms compatibility only.
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

  This does insert(w, find(before)).
- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.
- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.
- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

Sets a widget to accept input.

- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.

- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

Internal use only - "for hacks only".

- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.

- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Public Attributes

- void(* **display**)()
- void(* **entry**)(int)
- void(* **keyboard**)(uchar, int x, int y)
- int **menu** [3]
- void(* **motion**)(int x, int y)
- void(* **mouse**)(int b, int state, int x, int y)
- int **number**
- void(* **overlaydisplay**)()
- void(* **passivemotion**)(int x, int y)
- void(* **reshape**)(int w, int h)
- void(* **special**)(int, int x, int y)
- void(* **visibility**)(int)

Protected Member Functions

- void draw () override

  Draws the Fl_Gl_Window.
- void draw_overlay () override

  You must implement this virtual function if you want to draw into the overlay.
- int handle (int) override

  Handle some FLTK events as needed.

Protected Member Functions inherited from Fl_Gl_Window

- void draw_begin ()

  Supports drawing to an Fl_Gl_Window with the FLTK 2D drawing API.
- void draw_end ()

  To be used as a match for a previous call to Fl_Gl_Window::draw_begin().

Protected Member Functions inherited from Fl_Window

- void default_size_range ()

  Protected method to calculate the default size range of a window.
- void draw () override

  Draws the widget.
- int force_position () const

  Returns the internal state of the window's `FORCE_POSITION` flag.
- void force_position (int force)

  Sets an internal flag that tells FLTK and the window manager to honor position requests.
- void free_icons ()

  Deletes all icons previously attached to the window.
- int is_resizable ()

  Protected method to determine whether a window is resizable.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()

  Returns the internal array of widget sizes and positions.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget *, int)

  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)

  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)

  Allow derived groups to act when a child widget is removed from the group.
- int * sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

Draws a box of type t, of color c at the widget's position and size.

- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Public Types inherited from Fl_Window

- typedef struct HICON__ * **HICON**

Static Public Member Functions inherited from Fl_Gl_Window

- static int can_do (const int *m)

  Returns non-zero if the hardware supports the given OpenGL mode.
- static int **can_do** (int m)

  Returns non-zero if the hardware supports the given OpenGL mode.

Static Public Member Functions inherited from Fl_Window

- static Fl_Window * current ()

  Returns the last window that was made current.
- static void **default_callback** (Fl_Window *, void *v)

  Back compatibility: Sets the default callback v for win to call on close event.
- static void default_icon (const Fl_RGB_Image *)

  Sets a single default window icon.
- static void default_icons (const Fl_RGB_Image *[ ], int)

  Sets the default window icons.
- static void default_icons (HICON big_icon, HICON small_icon)

  Sets the default window icons (Windows platform only).

- static const char * default_xclass ()

  Returns the default xclass.
- static void default_xclass (const char *)

  Sets the default window xclass.
- static bool is_a_rescale ()

  Returns true when a window is being rescaled.
- static char show_next_window_iconic ()

  Returns the static flag whether the next window should be opened iconified.
- static void show_next_window_iconic (char stat)

  Sets a static flag whether the next window should be opened iconified.

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()

  Returns the currently active group.
- static void current (Fl_Group *g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Static Protected Attributes inherited from Fl_Window

- static Fl_Window * current_

  Stores the last window that was made current.

13.55.1 Detailed Description

GLUT is emulated using this window class and these static variables (plus several more static variables hidden in glut_compatibility.cxx):

13.55.2 Member Function Documentation

13.55.2.1 draw()

```
void Fl_Glut_Window::draw (
            void ) [override], [protected], [virtual]
```

Draws the Fl_Gl_Window.

You ***must*** *subclass* Fl_Gl_Window and provide an implementation for draw(). You may also provide an implementation of draw_overlay() if you want to draw into the overlay planes. You can avoid reinitializing the viewport and lights and other things by checking valid() at the start of draw() and only doing the initialization if it is false.

The draw() method can *only* use OpenGL calls. Do not attempt to call X, any of the functions in <FL/fl_draw.H>, or glX directly. Do not call gl_start() or gl_finish().

If double-buffering is enabled in the window, the back and front buffers are swapped after this function is completed.

The following pseudo-code shows how to use "if (!valid())" to initialize the viewport:

```
void mywindow::draw() {
 if (!valid()) {
     glViewport(0,0,pixel_w(),pixel_h());
     glFrustum(...) or glOrtho(...)
     ...other initialization...
 }
 if (!context_valid()) {
     ...load textures, etc. ...
 }
 // clear screen
 glClearColor(...);
 glClear(...);
 ... draw your geometry here ...
}
```

Actual example code to clear screen to black and draw a 2D white "X":

```
void mywindow::draw() {
    if (!valid()) {
        glLoadIdentity();
        glViewport(0,0,pixel_w(),pixel_h());
        glOrtho(-w(),w(),-h(),h(),-1,1);
    }
    // Clear screen
    glClear(GL_COLOR_BUFFER_BIT);
    // Draw white 'X'
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_LINE_STRIP); glVertex2f(w(), h()); glVertex2f(-w(),-h()); glEnd();
    glBegin(GL_LINE_STRIP); glVertex2f(w(),-h()); glVertex2f(-w(), h()); glEnd();
}
```

Regular FLTK widgets can be added as children to the Fl_Gl_Window. To correctly overlay the widgets, Fl_Gl_Window::draw() must be called after rendering the main scene.

```
void mywindow::draw() {
  // draw 3d graphics scene
  Fl_Gl_Window::draw();
  // -- or --
  draw_begin();
  Fl_Window::draw();
  // other 2d drawing calls, overlays, etc.
  draw_end();
}
```

Reimplemented from Fl_Gl_Window.

13.55.2.2 draw_overlay()

```
void Fl_Glut_Window::draw_overlay ( ) [override], [protected], [virtual]
```

You must implement this virtual function if you want to draw into the overlay.

The overlay is cleared before this is called. You should draw anything that is not clear using OpenGL. You must use gl_color(i) to choose colors (it allocates them from the colormap using system-specific calls), and remember that you are in an indexed OpenGL mode and drawing anything other than flat-shaded will probably not work.

Both this function and Fl_Gl_Window::draw() should check Fl_Gl_Window::valid() and set the same transformation. If you don't your code may not work on other systems. Depending on the OS, and on whether overlays are real or simulated, the OpenGL context may be the same or different between the overlay and main window.

Reimplemented from Fl_Gl_Window.

13.55.2.3 handle()

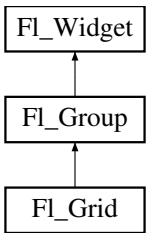
```
int Fl_Glut_Window::handle (
            int event ) [override], [protected], [virtual]
```

Handle some FLTK events as needed.
Reimplemented from Fl_Gl_Window.
The documentation for this class was generated from the following files:

- glut.H
- glut_compatibility.cxx

13.56 Fl_Grid Class Reference

Fl_Grid is a container (layout) widget with multiple columns and rows.
`#include <Fl_Grid.H>`
Inheritance diagram for Fl_Grid:



Classes

- class Cell

Public Member Functions

- Fl_Grid::Cell * cell (Fl_Widget *widget) const

  Get the grid cell of widget `widget`.
- Fl_Grid::Cell * cell (int row, int col) const

  Get the grid cell of row `row` and column `col`.
- virtual void clear_layout ()

  Reset the layout w/o removing widgets.
- void col_gap (const int *value, size_t size)

  Set more than one column gaps at once.
- int **col_gap** (int col) const
- void col_gap (int col, int value)

  Set the gap of column `col`.
- void col_weight (const int *value, size_t size)

  Set the weight of more than one column.
- int **col_weight** (int col) const
- void col_weight (int col, int value)

  Set the weight of a column.
- void col_width (const int *value, size_t size)

  Set minimal widths of more than one column.
- int **col_width** (int col) const
- void col_width (int col, int value)

  Set the minimal width of a column.
- short **cols** () const
- int **computed_col_width** (int col) const
- int **computed_row_height** (int row) const
- void debug (int level=127)

  Output layout information of this Fl_Grid to stderr.

- Fl_Grid (int X, int Y, int W, int H, const char *L=0)

  Create a new Fl_Grid widget.
- void gap (int *row_gap, int *col_gap) const

  Get the default gaps for rows and columns.
- virtual void gap (int row_gap, int col_gap=-1)

  Set default gaps for rows and columns.
- virtual void layout ()

  Calculate the grid layout and resize and position all widgets.
- virtual void layout (int rows, int cols, int margin=-1, int gap=-1)

  Set the basic layout parameters of the Fl_Grid widget.
- int margin (int *left, int *top, int *right, int *bottom) const

  Returns all outside margin sizes of the grid.
- virtual void margin (int left, int top=-1, int right=-1, int bottom=-1)

  Set all margins (left, top, right, bottom).
- bool **need_layout** () const

  Return whether layout calculation is required.
- void need_layout (int set)

  Request or reset the request to calculate the layout of children.
- virtual void resize (int X, int Y, int W, int H) override

  Recalculate the layout and position and resize all widgets.
- void row_gap (const int *value, size_t size)

  Set more than one row gaps at once.
- int **row_gap** (int row) const
- void row_gap (int row, int value)

  Set the gap of row `row`.
- void row_height (const int *value, size_t size)

  Set the minimal row height of more than one row.
- int **row_height** (int row) const
- void row_height (int row, int value)

  Set the minimal row height of row `row`.
- void row_weight (const int *value, size_t size)

  Set the weight of more than one row.
- int **row_weight** (int row) const
- void row_weight (int row, int value)

  Set the row weight of row `row`.
- short **rows** () const
- void show_grid (int set)

  Enable or disable drawing of the grid helper lines for visualization.
- void show_grid (int set, Fl_Color col)

  Enable or disable drawing of the grid helper lines for visualization.
- Fl_Grid::Cell * widget (Fl_Widget *wi, int row, int col, Fl_Grid_Align align=FL_GRID_FILL)

  Assign a widget to a grid cell and set its alignment.
- Fl_Grid::Cell * widget (Fl_Widget *wi, int row, int col, int rowspan, int colspan, Fl_Grid_Align align=FL_GRID_FILL)

  Assign a widget to a grid cell and set cell spanning and alignment.

Public Member Functions inherited from Fl_Group

- Fl_Widget *& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget *o)

  See void Fl_Group::add(Fl_Widget &w)
- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget *const * array () const

  Returns a pointer to the internal array of children.
- Fl_Group const * as_group () const override
- Fl_Group * as_group () override

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const

  Returns the n'th child.
- int **children** () const

  Returns how many child widgets the group has.
- void clear ()

  Deletes all child widgets from memory recursively.
- unsigned int clip_children () const

  Returns the current clipping mode.
- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)

  Removes the widget at `index` from the group and deletes it.
- void end ()

  Exactly the same as current(this->parent()).
- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*
- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.
- void **forms_end** ()

  This is for forms compatibility only.
- int handle (int) override

  Handles the specified event.
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

  This does insert(w, find(before)).
- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.

- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.
- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- void resize (int, int, int, int) override

  Resizes the Fl_Group widget and all of its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.

- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

Sets the image to use as part of the widget label when in the inactive state.

- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.

- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

Changes the size of the widget.

- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- Cell ∗ **add_cell** (int row, int col)
- virtual void draw () override

  Draws the Fl_Grid widget and all children.
- virtual void draw_grid ()

  Draws the grid helper lines for design and debugging purposes.
- void **init** ()
- void on_remove (int) override

  Fl_Group calls this method when a child widget is about to be removed.
- void **remove_cell** (int row, int col)

Protected Member Functions inherited from Fl_Group

- Fl_Rect ∗ bounds ()

  Returns the internal array of widget sizes and positions.
- void draw () override

  Draws the widget.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget ∗, int)

  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)

  Allow derived groups to act when a widget is moved within the group.
- int ∗ sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

Draws the widget's label at the defined label position.

- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Protected Attributes

- bool **draw_grid_**
- Fl_Color **grid_color**

Friends

- class **Fl_Grid_Type**

Additional Inherited Members

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()

  Returns the currently active group.
- static void current (Fl_Group *g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,

GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU = 1<<19 ,
NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 , AUTO_DELETE_USER_DATA = 1<<23 ,
MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
USERFLAG1 = 1<<31 }

flags possible values enumeration.

13.56.1 Detailed Description

Fl_Grid is a container (layout) widget with multiple columns and rows.
This container widget features very flexible layouts in columns and rows w/o the need to position each child widget in x/y coordinates.
Widgets are assigned to grid cells (column, row) with their minimal sizes in `w()` and `h()`. The `x()` and `y()` positions are ignored and can be (0, 0). Fl_Grid calculates widget positions and resizes the widgets to fit into the grid. It is possible to create a single row or column of widgets with Fl_Grid.
You should design your grid with the smallest possible sizes of all widgets in mind. Fl_Grid will automatically assign additional space to cells according to some rules (described later) when resizing the Fl_Grid widget.
Hint: You should set a minimum window size to make sure the Fl_Grid is never resized below its minimal sizes. Resizing below the given widget sizes results in undefined behavior.
Fl_Grid and other container widgets (e.g. Fl_Group) can be nested. One main advantage of this usage is that widget coordinates in embedded Fl_Group widgets become relative to the group and will be positioned as expected.

Todo This (relative group coordinates of nested groups of Fl_Grid) needs explanation and maybe an example.

Fl_Grid child widgets are handled by its base class Fl_Group but Fl_Grid stores additional data corresponding to each widget in internal grid cells.
Fl_Grid children are allowed to span multiple columns and rows like HTML <table> cells. Individual children can have fixed sizes or be aligned inside their cells (left, right, top, bottom, and more) and/or follow their cell sizes when the Fl_Grid container is resized.
Note to resizing: since Fl_Grid uses its own layout algorithm the normal Fl_Group::resizable() widget is ignored (if set). Calling init_sizes() is not necessary.

Note

Fl_Grid is, as of FLTK 1.4.x, still in experimental state and should be used with caution. The API can still be changed although it is assumed to be almost stable - as stable as possible for a first release.

Example: Simple 3x3 Fl_Grid with five buttons:

```
#include <FL/Fl.H>
#include <FL/Fl_Double_Window.H>
#include <FL/Fl_Grid.H>
#include <FL/Fl_Button.H>

int main(int argc, char **argv) {
  Fl_Double_Window *win = new Fl_Double_Window(320, 180, "3x3 Fl_Grid with Buttons");
  // create the Fl_Grid container with five buttons
  Fl_Grid *grid = new Fl_Grid(0, 0, win->w(), win->h());
  grid->layout(3, 3, 10, 10);
  grid->color(FL_WHITE);
  Fl_Button *b0 = new Fl_Button(0, 0, 0, 0, "New");
  Fl_Button *b1 = new Fl_Button(0, 0, 0, 0, "Options");
  Fl_Button *b3 = new Fl_Button(0, 0, 0, 0, "About");
  Fl_Button *b4 = new Fl_Button(0, 0, 0, 0, "Help");
  Fl_Button *b6 = new Fl_Button(0, 0, 0, 0, "Quit");
  // assign buttons to grid positions
  grid->widget(b0, 0, 0);
  grid->widget(b1, 0, 2);
  grid->widget(b3, 1, 1);
  grid->widget(b4, 2, 0);
  grid->widget(b6, 2, 2);
  grid->show_grid(0);     // 1 to display grid helper lines
  grid->end();
  win->end();
  win->resizable(grid);
  win->size_range(300, 100);
  win->show(argc, argv);
  return Fl::run();
}
```

Figure 13.19 Simple 3x3 Fl_Grid

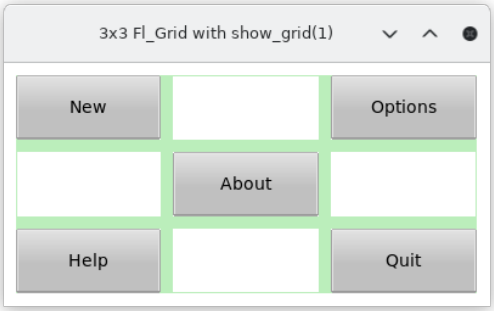


Figure 13.20 show_grid() set to 1

13.56.2 Constructor & Destructor Documentation

13.56.2.1 Fl_Grid()

```
Fl_Grid::Fl_Grid (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

Create a new Fl_Grid widget.

Todo More documentation of Fl_Grid constructor?

13.56.3 Member Function Documentation

13.56.3.1 cell() [1/2]

```
Fl_Grid::Cell * Fl_Grid::cell (
            Fl_Widget * widget ) const
```

Get the grid cell of widget `widget`.
The pointer to the cell can be used for further assignment of properties like alignment etc.
Hint: If you know the row and column index of the cell you should use Fl_Grid::cell(int row, int col) instead because it is **much** faster.
Please see Fl_Grid::cell(int row, int col) for details and the validity of cell pointers.

Parameters

in	*widget*	widget whose cell is requested

Return values

NULL	if `widget` is not assigned to a cell

13.56.3.2 cell() [2/2]

```
Fl_Grid::Cell * Fl_Grid::cell (
            int row,
            int col ) const
```

Get the grid cell of row `row` and column `col`.
Widgets and other attributes are organized in cells (Fl_Grid::Cell).
This cell is an opaque structure (class) with some public methods. **Don't** assume anything about grid cell sizes and ordering in memory. These are implementation details that can be changed without notice.
The validity of an Fl_Grid::Cell pointer is limited. It will definitely be invalidated when the overall grid layout is changed, for instance by calling layout(int, int).
Adding new cells beyond the current layout limits will also invalidate cell pointers but this is not (yet) implemented.
Attempts to assign widgets to out-of-bounds cells are currently ignored.
The only well-defined usage of cell pointers is to set one or more properties like widget alignment of a cell after retrieving the cell pointer. Don't store cell pointers in your program for later reference.

Parameters

in	*row*	row index
in	*col*	column index

Returns

pointer to cell

Return values

NULL	if `row` or `col` is out of bounds or no widget was assigned

13.56.3.3 clear_layout()

```
void Fl_Grid::clear_layout ( ) [virtual]
```

Reset the layout w/o removing widgets.
Removes all cells and sets rows and cols to zero. Existing widgets are kept as children of the Fl_Group (base class) but are hidden.
This method should be rarely used. You may want to call Fl_Grid::clear() to remove all widgets and reset the layout to zero rows and columns.
You must call layout(int rows, int cols, ...) to set a new layout, allocate new cells, and assign widgets to new cells.

Todo Fl_Grid::clear() needs to be implemented as documented above!

13.56.3.4 col_gap() [1/2]

```
void Fl_Grid::col_gap (
            const int * value,
            size_t size )
```

Set more than one column gaps at once.

See also

Fl_Grid::col_weight(const int *value, size_t size) for handling of the value array and `size`.

13.56.3.5 col_gap() [2/2]

```
void Fl_Grid::col_gap (
            int col,
            int value )
```

Set the gap of column `col`.
Note that the gap is right of each column except the last one which is ignored. Use margin() for the right most column.

Parameters

in	*col*	column
in	*value*	gap size after the column

13.56.3.6 col_weight() [1/2]

```
void Fl_Grid::col_weight (
            const int * value,
            size_t size )
```

Set the weight of more than one column.
The values are taken from the array `value` and assigned sequentially to columns, starting from column 0. If the array `size` is too large extraneous values are ignored.
Negative values in the `array` are not assigned to their columns, i.e. the existing value for the corresponding column is not changed.
Example:

```
int val[] = { 0, 0, 50, -1, -1, 50, 0 };
grid->col_weight(val, sizeof(val)/sizeof(val[0]));
```

Parameters

in	*value*	an array of column weights
in	*size*	the size of the array (number of values)

13.56.3.7 col_weight() [2/2]

```
void Fl_Grid::col_weight (
            int col,
            int value )
```

Set the weight of a column.
Column and row weights are used to distribute additional space when the grid is resized beyond its defined (minimal) size. All weight values are relative and can be chosen freely. Suggested weights are in the range {0 .. 100}, 0 (zero) disables resizing of the column.
How does it work?
Whenever additional space (say: `SPACE` in pixels) is to be distributed to a set of columns the weights of all columns are added to a value `SUM`, then every single column width is increased by the value (in pseudo code):

```
col.width += SPACE * col.weight / SUM
```

Resulting pixel values are rounded to the next integer and rounding differences are added to or subtracted from the column with the highest weight. If more columns have the same weight one of them is chosen.

Note

> If none of the columns considered for resizing have weights $>$ 0 then Fl_Grid assigns the remaining space to an arbitrary column or to all considered columns evenly. This is implementation defined and can be changed without notice. You can avoid this situation by designing your grid with sensible sizes and weights.

Parameters

in	*col*	column number (counting from 0)
in	*value*	weight, must be $>=$ 0

13.56.3.8 col_width() [1/2]

```
void Fl_Grid::col_width (
            const int * value,
            size_t size )
```

Set minimal widths of more than one column.
The values are taken from the array `value` and assigned sequentially to columns, starting from column 0. If the array `size` is too large extraneous values are ignored.
Negative values in the `array` are not assigned to their columns, i.e. the existing value for the corresponding column is not changed.
Example:

```
int widths[] = { 0, 0, 50, -1, -1, 50, 0 };
grid->col_width(widths, sizeof(width)/sizeof(width[0]));
```

Parameters

in	*value*	an array of column widths
in	*size*	the size of the array (number of values)

13.56.3.9 col_width() [2/2]

```
void Fl_Grid::col_width (
            int col,
            int value )
```

Set the minimal width of a column.
Column widths are calculated by using the maximum of all widget widths in that column and the given column width.
After calculating the width additional space is added when resizing according to the `weight` of the column.
You can set one or more column widths in one call by using Fl_Grid::col_width(const int *value, size_t size).

Parameters

in	*col*	column number (counting from 0)
in	*value*	minimal column width, must be >= 0

See also

Fl_Grid::col_width(const int *value, size_t size)

13.56.3.10 debug()

```
void Fl_Grid::debug (
            int level = 127 )
```

Output layout information of this Fl_Grid to stderr.
Parameter `level` will be used to define the amount of output.

- 0 = nothing
- 127 = everything
- other values not yet defined

Note

It is not yet defined which kind of values `level` will have, either a numerical value (127 = maximum, 0 = nothing) or a bit mask that determines what to output.

Todo Add more information about cells and children.

Control output by using `level`.

Parameters

in	*level*	not yet used (0-127, default = 127)

13.56.3.11 draw()

```
void Fl_Grid::draw (
            void ) [override], [protected], [virtual]
```

Draws the Fl_Grid widget and all children.
If the layout has been changed layout() is called before the widget is drawn so all children are arranged as designed.

See also

layout()

need_layout()

Implements Fl_Widget.

13.56.3.12 draw_grid()

```
void Fl_Grid::draw_grid ( ) [protected], [virtual]
```

Draws the grid helper lines for design and debugging purposes.
This method is protected so it can be modified in subclasses.

13.56.3.13 gap() [1/2]

```
void Fl_Grid::gap (
            int * row_gap,
            int * col_gap ) const
```

Get the default gaps for rows and columns.

Parameters

out	*row_gap*	pointer to int to receive column gap, may be NULL
out	*col_gap*	pointer to int to receive column gap, may be NULL

13.56.3.14 gap() [2/2]

```
void Fl_Grid::gap (
            int row_gap,
            int col_gap = -1 ) [virtual]
```

Set default gaps for rows and columns.
All gaps are positioned below the rows and right of their columns.
The bottom row and the right-most column don't have a gap, i.e. the gap sizes of these columns and rows are ignored. You can use a right or bottom margin instead.
You have to specify at least one argument, `col_gap` is optional. If you don't specify an argument or use a negative value (e.g. -1) then that margin is not affected.
You can also initialize the default gaps with layout(int, int, int, int).

Parameters

in	*row_gap*	default gap for all rows
in	*col_gap*	default gap for all columns

See also

Fl_Grid::layout(int rows, int cols, int margin, int gap)

13.56.3.15 layout() [1/2]

```
void Fl_Grid::layout ( ) [virtual]
```

Calculate the grid layout and resize and position all widgets.
This is called automatically when the Fl_Grid is resized. You need to call it once after you added widgets or moved widgets between cells.
Calling it once after all modifications are completed is enough.

Todo Document when and why to call layout() w/o args. See Fl_Flex::layout()

See also

Fl_Grid::layout(int rows, int cols, int margin, int gap)

13.56.3.16 layout() [2/2]

```
void Fl_Grid::layout (
            int rows,
            int cols,
            int margin = -1,
            int gap = -1 ) [virtual]
```

Set the basic layout parameters of the Fl_Grid widget.
You need to specify at least `rows` and `cols` to define a layout before you can add widgets to the grid.
Parameters `margin` and `gap` are optional.
You can call layout(int rows, int cols, int margin, int gap) again to change the layout but this is inefficient since all cells are reallocated if the layout changed.
Calling this with the same values of `rows` and `cols` is fast and can be used to change `margin` and `gap` w/o reallocating the cells.
`margin` sets all margins (left, top, right, bottom) to the same value. Negative values (e.g. -1) don't change the established margins. The default value set by the constructor is 0.
`gap` sets row and column gaps to the same value. Negative values (e.g. -1) do not affect the established gaps. The default value set by the constructor is 0.
After you added all widgets you must call layout() once without arguments to calculate the actual layout and to position and resize all widgets.

Todo Document when and why to call layout() w/o args. See Fl_Flex::layout()

Parameters

in	*rows*	number of rows
in	*cols*	number of columns
in	*margin*	margin size inside the Fl_Grid's border
in	*gap*	gap size between cells

See also

Fl_Grid::layout()

13.56.3.17 margin() [1/2]

```
int Fl_Grid::margin (
            int * left,
```

```
int * top,
int * right,
int * bottom ) const
```

Returns all outside margin sizes of the grid.
All margin sizes are returned in the given arguments. If any argument is NULL the respective value is not returned.

Parameters

out	*left*	returns left margin if not NULL
out	*top*	returns top margin if not NULL
out	*right*	returns right margin if not NULL
out	*bottom*	returns bottom margin if not NULL

Returns

whether all margins are equal

Return values

1	all margins have the same size
0	at least one margin has a different size

13.56.3.18 margin() [2/2]

```
void Fl_Grid::margin (
            int left,
            int top = -1,
            int right = -1,
            int bottom = -1 )  [virtual]
```

Set all margins (left, top, right, bottom).
All margins are measured in pixels inside the box borders. You need to specify at least one argument, all other arguments are optional. If you don't specify an argument or use a negative value (e.g. -1) then that particular margin is not affected.

Parameters

in	*left*	left margin
in	*top*	top margin
in	*right*	right margin
in	*bottom*	bottom margin

13.56.3.19 need_layout()

```
void Fl_Grid::need_layout (
            int set )  [inline]
```

Request or reset the request to calculate the layout of children.
If called with true (1) this calls redraw() to schedule a full draw(). When draw is eventually called, the layout is (re)calculated before actually drawing the widget.

Parameters

in	*set*	1 to request layout calculation, 0 to reset the request

13.56.3.20 on_remove()

```
void Fl_Grid::on_remove (
            int index )  [override], [protected], [virtual]
```

Fl_Group calls this method when a child widget is about to be removed.
Make sure that the widget is also removed from our internal list of children.
Reimplemented from Fl_Group.

13.56.3.21 resize()

```
void Fl_Grid::resize (
            int X,
            int Y,
            int W,
            int H )  [override], [virtual]
```

Recalculate the layout and position and resize all widgets.
This method overrides Fl_Group::resize() and calculates all positions and sizes of its children according to its own rules.

Parameters

in	*X,Y*	new widget position
in	*W,H*	new widget size

Reimplemented from Fl_Widget.

13.56.3.22 row_gap() [1/2]

```
void Fl_Grid::row_gap (
            const int * value,
            size_t size )
```

Set more than one row gaps at once.

See also

Fl_Grid::col_weight(const int *value, size_t size) for handling of the value array and `size`.

13.56.3.23 row_gap() [2/2]

```
void Fl_Grid::row_gap (
            int row,
            int value )
```

Set the gap of row `row`.
Note that the gap is below each row except the last one which is ignored. Use margin() for the bottom row.

Parameters

in	*row*	row
in	*value*	gap size below the row

13.56.3.24 row_height() [1/2]

```
void Fl_Grid::row_height (
            const int * value,
            size_t size )
```

Set the minimal row height of more than one row.

Parameters

in	*value*	array of height values
in	*size*	size of array `value`

See also

Fl_Grid::col_weight(const int *value, size_t size) for handling of the value array and `size`.

13.56.3.25 row_height() [2/2]

```
void Fl_Grid::row_height (
            int row,
            int value )
```

Set the minimal row height of row `row`.

Parameters

in	*row*	row
in	*value*	minimal height of the row

13.56.3.26 row_weight() [1/2]

```
void Fl_Grid::row_weight (
            const int * value,
            size_t size )
```

Set the weight of more than one row.

Parameters

in	*value*	array of height values
in	*size*	size of array `value`

See also

Fl_Grid::col_weight(const int *value, size_t size) for handling of the `value` array and `size`.

13.56.3.27 row_weight() [2/2]

```
void Fl_Grid::row_weight (
            int row,
            int value )
```

Set the row weight of row `row`.

Parameters

in	*row*	row
in	*value*	weight of the row

13.56.3.28 show_grid() [1/2]

```
void Fl_Grid::show_grid (
            int set )  [inline]
```

Enable or disable drawing of the grid helper lines for visualization.
Use this method during the design stage of your Fl_Grid widget or for debugging if widgets are not positioned as intended.
The default is a light green color but you can change it for better contrast if needed, see show_grid(int set, Fl_Color col).

Note

You can define the environment variable `FLTK_GRID_DEBUG=1` to set show_grid(1) for all Fl_Grid widgets at construction time. This enables you to debug the grid layout w/o changing code.

Parameters

in	*set*	1 (true) = draw, 0 = don't draw the grid

See also

show_grid(int set, Fl_Color col)

13.56.3.29 show_grid() [2/2]

```
void Fl_Grid::show_grid (
            int set,
            Fl_Color col )  [inline]
```

Enable or disable drawing of the grid helper lines for visualization.
This method also sets the color used for the helper lines.
The default is a light green color but you can change it to any color for better contrast if needed.

Parameters

in	*set*	1 (true) = draw, 0 = don't draw the grid
in	*col*	color to use for the grid helper lines

See also

show_grid(int set)

13.56.3.30 widget() [1/2]

```
Fl_Grid::Cell * Fl_Grid::widget (
            Fl_Widget * wi,
            int row,
            int col,
            Fl_Grid_Align align = FL_GRID_FILL )
```

Assign a widget to a grid cell and set its alignment.
This short form sets row and column spanning attributes to (1, 1).
For more information see Fl_Grid::widget(Fl_Widget *wi, int row, int col, int rowspan, int colspan, Fl_Grid_Align align)

Parameters

in	*wi*	widget to be assigned to the cell
in	*row*	row
in	*col*	column
in	*align*	widget alignment inside the cell

Returns

assigned cell

Return values

NULL	if `row` or `col` is out of bounds

See also

Fl_Grid::widget(Fl_Widget *wi, int row, int col, int rowspan, int colspan, Fl_Grid_Align align)

13.56.3.31 widget() [2/2]

```
Fl_Grid::Cell * Fl_Grid::widget (
            Fl_Widget * wi,
            int row,
            int col,
            int rowspan,
            int colspan,
            Fl_Grid_Align align = FL_GRID_FILL )
```

Assign a widget to a grid cell and set cell spanning and alignment.
Default alignment is `FL_GRID_FILL` which stretches the widget in horizontal and vertical directions to fill the whole cell(s) given by `colspan` and `rowspan`.
You can use this method to move a widget from one cell to another; it is automatically removed from its old cell. If the new cell is already assigned to another widget that widget is deassigned but kept as a child of the group.
Before you can assign a widget to a cell it must have been created as a child of the Fl_Grid widget (i.e. its Fl_Group).

Parameters

in	*wi*	widget to be assigned to the cell
in	*row*	row
in	*col*	column
in	*rowspan*	vertical span in cells, default 1
in	*colspan*	horizontal span in cells, default 1
in	*align*	widget alignment inside the cell

Returns

assigned cell

Return values

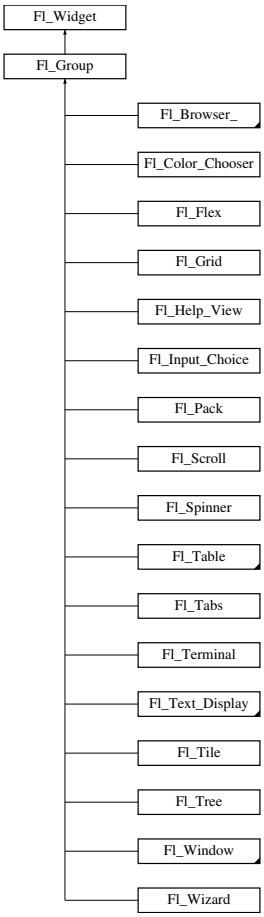
NULL	if `row` or `col` is out of bounds or `wi` is not a child

The documentation for this class was generated from the following files:

- Fl_Grid.H
- Fl_Grid.cxx

13.57 Fl_Group Class Reference

The Fl_Group class is the main FLTK container widget.
`#include <Fl_Group.H>`
Inheritance diagram for Fl_Group:



Public Member Functions

- Fl_Widget *& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget *o)

  See void Fl_Group::add(Fl_Widget &w)
- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget *const * array () const

  Returns a pointer to the internal array of children.
- Fl_Group const * as_group () const override
- Fl_Group * as_group () override

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const

  Returns the n'th child.
- int **children** () const

  Returns how many child widgets the group has.
- void clear ()

  Deletes all child widgets from memory recursively.
- unsigned int clip_children () const

  Returns the current clipping mode.

- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)

  Removes the widget at `index` from the group and deletes it.
- void end ()

  Exactly the same as current(this->parent()).
- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*
- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.
- void **forms_end** ()

  This is for forms compatibility only.
- int handle (int) override

  Handles the specified event.
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

  This does insert(w, find(before)).
- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.
- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.
- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- void resize (int, int, int, int) override

  Resizes the Fl_Group widget and all of its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.

- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

Gets the background color of the widget.

- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.

- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

Schedules the drawing of the widget.

- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.

- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Static Public Member Functions

- static Fl_Group * current ()

  Returns the currently active group.
- static void current (Fl_Group *g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Member Functions

- Fl_Rect * bounds ()

  Returns the internal array of widget sizes and positions.
- void draw () override

  Draws the widget.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget *, int)

  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)

  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)

  Allow derived groups to act when a child widget is removed from the group.
- int * sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

Sets a flag in the flags mask.

- void w (int v)

  Internal use only.

- void x (int v)

  Internal use only.

- void y (int v)

  Internal use only.

Additional Inherited Members

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.57.1 Detailed Description

The Fl_Group class is the main FLTK container widget.
It maintains an ordered list of child widgets. These children can themselves be any widget including Fl_Group. The most important subclass of Fl_Group is Fl_Window, however groups can also be used to control radio buttons, to enforce resize behavior, or to manage layout of widgets, see Fl_Pack, Fl_Flex, Fl_Grid.
The tab and arrow keys are used to move the focus between widgets of this group, and to other groups. The only modifier grabbed is shift (for shift-tab), so that ctrl-tab, alt-up, and such are free for the app to use as shortcuts.
To remove a widget from the group and destroy it, in 1.3.x (and up) you can simply use:

```
delete some_widget;
```

..and this will trigger proper scheduling of the widget's removal from its parent group.
If used as a child of `Fl_Tabs`, setting `when(FL_WHEN_CLOSED)` will enable the Close button in the corresponding tab. If the user clicks the Close button, the callback of this group will be called with the callback reason `FL_REASON_CLOSED`.

13.57.2 Constructor & Destructor Documentation

13.57.2.1 Fl_Group()

```
Fl_Group::Fl_Group (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

Creates a new Fl_Group widget using the given position, size, and label string.
The default boxtype is FL_NO_BOX.

13.57.2.2 ~Fl_Group()

```
Fl_Group::~Fl_Group ( ) [virtual]
```

The destructor *also deletes all the children*.
This allows a whole tree to be deleted at once, without having to keep a pointer to all the children in the user code.
It is allowed that the Fl_Group and all of its children are automatic (local) variables, but you must declare the Fl_Group *first*, so that it is destroyed last.
If you add static or automatic (local) variables to an Fl_Group, then it is your responsibility to remove (or delete) all such static or automatic child widgets ***before*** *destroying* the group - otherwise the group will attempt to call delete operator on them leading to undefined behavior!

13.57.3 Member Function Documentation

13.57.3.1 array()

```
Fl_Widget *const * Fl_Group::array ( ) const
```

Returns a pointer to the internal array of children.

Note

This pointer is only valid until the next time a child is added or removed.

13.57.3.2 as_group() [1/2]

```
Fl_Group const * Fl_Group::as_group ( ) const [inline], [override], [virtual]
```

Reimplemented from Fl_Widget.

13.57.3.3 as_group() [2/2]

```
Fl_Group * Fl_Group::as_group ( ) [inline], [override], [virtual]
```

Returns an Fl_Group pointer if this widget is an Fl_Group.
Use this method if you have a widget (pointer) and need to know whether this widget is derived from Fl_Group. If it returns non-NULL, then the widget in question is derived from Fl_Group, and you can use the returned pointer to access its children or other Fl_Group-specific methods.
Example:

```
void my_callback (Fl_Widget *w, void *) {
  Fl_Group *g = w->as_group();
  if (g)
    printf ("This group has %d children\n", g->children());
  else
    printf ("This widget is not a group!\n");
}
```

Return values

NULL	if this widget is not derived from Fl_Group.

Note

This method is provided to avoid dynamic_cast.

See also

Fl_Widget::as_window(), Fl_Widget::as_gl_window()

Reimplemented from Fl_Widget.

13.57.3.4 begin()

```
void Fl_Group::begin ( )
```

Sets the current group so you can build the widget tree by just constructing the widgets.
begin() is automatically called by the constructor for Fl_Group (and thus for Fl_Window as well). begin() *is exactly the same as* current(this). *Don't forget to end() the group or window!*

13.57.3.5 bounds()

```
Fl_Rect * Fl_Group::bounds ( ) [protected]
```

Returns the internal array of widget sizes and positions.
If the bounds() array does not exist, it will be allocated and filled with the current widget sizes and positions.
The bounds() array stores the initial positions of widgets as Fl_Rect's. The size of the array is children() + 2.

- The first Fl_Rect is the group,
- the second is the resizable (clipped to the group),
- the rest are the children.

This is a convenient order for the resize algorithm.
If the group and/or the resizable() is a Fl_Window (or subclass) then the x() and y() coordinates of their respective Fl_Rect's are zero.

Note

You should never need to use this *protected* method directly, unless you have special needs to rearrange the children of a Fl_Group. Fl_Tile uses this to rearrange its widget positions. The returned array should be considered read-only. Do not change its contents. If you need to rearrange children in a group, do so by resizing the children and call init_sizes().

#include <FL/Fl_Rect.H> if you want to access the bounds() array in your derived class. Fl_Rect.H is intentionally not included by Fl_Group.H to avoid unnecessary dependencies.

Returns

Array of Fl_Rect's with widget positions and sizes. The returned array is only valid until init_sizes() is called or widgets are added to or removed from the group.

See also

init_sizes()

Since

FLTK 1.4.0

13.57.3.6 child()

```
Fl_Widget * Fl_Group::child (
            int n ) const [inline]
```

Returns the n'th child.
Returns `nullptr` if `n` is out of range (since FLTK 1.4.0).
No range checking was done in FLTK 1.3 and older versions!

Parameters

in	*n*	index of child (0 .. children() - 1)

Returns

pointer to the n'th child or nullptr if out of range

13.57.3.7 clear()

```
void Fl_Group::clear (
            void )
```

Deletes all child widgets from memory recursively.
This method differs from the remove() method in that it affects all child widgets and deletes them from memory.
The resizable() widget of the Fl_Group is set to the Fl_Group itself.

13.57.3.8 clip_children() [1/2]

```
unsigned int Fl_Group::clip_children ( ) const  [inline]
```

Returns the current clipping mode.

Returns

true, if clipping is enabled, false otherwise.

See also

void Fl_Group::clip_children(int c)

13.57.3.9 clip_children() [2/2]

```
void Fl_Group::clip_children (
            int c )  [inline]
```

Controls whether the group widget clips the drawing of child widgets to its bounding box.
Set c to 1 if you want to clip the child widgets to the bounding box.
The default is to not clip (0) the drawing of child widgets.

13.57.3.10 current() [1/2]

```
Fl_Group * Fl_Group::current ( )  [static]
```

Returns the currently active group.
The Fl_Widget constructor automatically does current()->add(widget) if this is not null. To prevent new widgets from being added to a group, call Fl_Group::current(0).

13.57.3.11 current() [2/2]

```
void Fl_Group::current (
            Fl_Group * g )  [static]
```

Sets the current group.

See also

Fl_Group::current()

13.57.3.12 delete_child()

```
int Fl_Group::delete_child (
            int index )  [virtual]
```

Removes the widget at `index` from the group and deletes it.
This method does nothing if `index` is out of bounds.
This method differs from the remove() method in that it deletes the widget from memory. Since this method is virtual it can be reimplemented in subclasses with additional requirements and consequences. See the documentation of subclasses.
Many subclasses don't need to reimplement this method.

Note

This method **may** refuse to remove and delete the widget if it is an essential part of the Fl_Group, for instance a scrollbar in an Fl_Scroll group. In this case the widget is neither removed nor deleted.

This method does not call init_sizes() or redraw(). This is left to user code if necessary.
Returns 0 if the widget was removed and deleted. Return values $>$ 0 are reserved for use by FLTK core widgets. Return values $<$ 0 are free to be used by user defined widgets.

Todo Reimplementation of Fl_Group::delete_child(int) in more FLTK subclasses. This is not yet complete.

Parameters

in	*index*	index of child to be removed

Returns

success (0) or error code

Return values

0	success
1	index out of range
2	widget not allowed to be removed (see note)
>2	reserved for FLTK use

Since

FLTK 1.4.0

Reimplemented in Fl_Scroll.

13.57.3.13 draw()

```
void Fl_Group::draw ( ) [override], [protected], [virtual]
```

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();                 // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

Reimplemented in Fl_Help_View, Fl_Input_Choice, Fl_Pack, Fl_Scroll, Fl_Spinner, Fl_Table, Fl_Tabs, Fl_Text_Display, Fl_Tree, Fl_Window, Fl_Wizard, and Fl_Terminal.

13.57.3.14 draw_child()

```
void Fl_Group::draw_child (
            Fl_Widget & widget ) const [protected]
```

Forces a child to redraw.

This draws a child widget, if it is not clipped. The damage bits are cleared after drawing.

13.57.3.15 draw_children()

```
void Fl_Group::draw_children ( ) [protected]
```

Draws all children of the group.

This is useful, if you derived a widget from Fl_Group and want to draw a special border or background. You can call draw_children() from the derived draw() method after drawing the box, border, or background.

13.57.3.16 end()

```
void Fl_Group::end ( )
```

Exactly the same as current(this->parent()).

Any new widgets added to the widget tree will be added to the parent of the group.

13.57.3.17 find()

```
int Fl_Group::find (
            const Fl_Widget * o ) const
```

Searches the children for the widget and returns the index.
Returns children() if the widget is NULL or not found.

13.57.3.18 handle()

```
int Fl_Group::handle (
            int event )  [override], [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to *override* the behavior of the base class. This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to *filter* event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you *consumed* the event by returning 1 even if you didn't *do* anything with it.

Parameters

in	*event*	the kind of event received

Return values

0	if the event was not used or understood
1	if the event was used and can be deleted

See also

Fl_Event

Reimplemented from Fl_Widget.
Reimplemented in Fl_Table, Fl_Terminal, Fl_Text_Display, Fl_Text_Editor, Fl_Tree, Fl_Spinner, Fl_Table_Row, Fl_Tile, Fl_Help_View, Fl_Scroll, Fl_Tabs, and Fl_Window.

13.57.3.19 init_sizes()

```
void Fl_Group::init_sizes ( )
```

Resets the internal array of widget sizes and positions.
The Fl_Group widget keeps track of the original widget sizes and positions when resizing occurs so that if you resize a window back to its original size the widgets will be in the correct places. If you rearrange the widgets in your group, call this method to register the new arrangement with the Fl_Group that contains them.
If you add or remove widgets, this will be done automatically.

Note

The internal array of widget sizes and positions will be allocated and filled when the next resize() occurs. For more information on the contents and structure of the bounds() array see bounds().

See also

bounds()

sizes() (deprecated)

13.57.3.20 insert() [1/2]

```
void Fl_Group::insert (
            Fl_Widget & o,
            int index )
```

The widget is removed from its current group (if any) and then inserted into this group.
It is put at index n - or at the end, if n >= children(). This can also be used to rearrange the widgets inside a group.

13.57.3.21 insert() [2/2]

```
void Fl_Group::insert (
            Fl_Widget & o,
            Fl_Widget * before )  [inline]
```

This does insert(w, find(before)).
This will append the widget if `before` is not in the group.

13.57.3.22 on_insert()

```
int Fl_Group::on_insert (
            Fl_Widget * candidate,
            int index )  [protected], [virtual]
```

Allow derived groups to act when a widget is added as a child.
Widgets derived from Fl_Group may store additional data for their children. Overriding this method will allow derived classes to generate these data structures just before the child is added.
This method usually returns the same index that was given in the parameters. By setting a new index, the position of other widgets in the list of children can be preserved (e.g. Fl_Scroll keeps its scroll bars as the last two children).
By returning -1, Fl_Group::insert will not add the child to the group. This is not recommended, but Fl_Table does something similar to forward children to a hidden group.

Parameters

candidate	the candidate will be added to the child vector after this method returns.
index	add the child at this position in the list of children

Returns

index to position the child as planned

a new index to force the child to a different position

-1 to keep the group from adding the candidate

Reimplemented in Fl_Scroll, Fl_Tabs, and Fl_Tile.

13.57.3.23 on_move()

```
int Fl_Group::on_move (
            int oldIndex,
            int newIndex )  [protected], [virtual]
```

Allow derived groups to act when a widget is moved within the group.
Widgets derived from Fl_Group may store additional data for their children. Overriding this method will allow derived classes to move these data structures just before the child itself is moved.
This method usually returns the new index that was given in the parameters. By setting a different destination index, the position of other widgets in the list of children can be preserved.
By returning -1, Fl_Group::insert will not move the child.

Parameters

oldIndex	the current index of the child that will be moved
newIndex	the new index of the child

Returns

`newIndex` to position the child as planned

a different index to force the child to a different position

-1 to keep the group from moving the child

Reimplemented in Fl_Scroll, Fl_Tabs, and Fl_Tile.

13.57.3.24 on_remove()

```
void Fl_Group::on_remove (
            int index )  [protected], [virtual]
```

Allow derived groups to act when a child widget is removed from the group.
Widgets derived from Fl_Group may store additional data for their children. Overriding this method will allow derived classes to remove these data structures just before the child is removed.

Parameters

index	remove the child at this position

Reimplemented in Fl_Flex, Fl_Grid, Fl_Tabs, and Fl_Tile.

13.57.3.25 remove() [1/3]

```
void Fl_Group::remove (
            Fl_Widget & o )
```

Removes a widget from the group but does not delete it.
This method does nothing if the widget is not a child of the group.
This method differs from the clear() method in that it only affects a single widget and does not delete it from memory.

Note

If you have the child's index anyway, use remove(int index) instead, because this doesn't need a child lookup in the group's table of children. This can be much faster, if there are lots of children.

13.57.3.26 remove() [2/3]

```
void Fl_Group::remove (
            Fl_Widget * o )  [inline]
```

Removes the widget `o` from the group.

See also

void remove(Fl_Widget&)

13.57.3.27 remove() [3/3]

```
void Fl_Group::remove (
            int index )
```

Removes the widget at `index` from the group but does not delete it.
This method does nothing if `index` is out of bounds.
This method differs from the clear() method in that it only affects a single widget and does not delete it from memory.

Since

FLTK 1.3.0

13.57.3.28 resizable() [1/3]

```
Fl_Widget * Fl_Group::resizable ( ) const  [inline]
```

Returns the group's resizable widget.
See void Fl_Group::resizable(Fl_Widget *o)

13.57.3.29 resizable() [2/3]

```
void Fl_Group::resizable (
            Fl_Widget & o )  [inline]
```

Sets the group's resizable widget.
See void Fl_Group::resizable(Fl_Widget *o)

13.57.3.30 resizable() [3/3]

```
void Fl_Group::resizable (
            Fl_Widget * o )  [inline]
```

The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
If the resizable is NULL the group's size is fixed and all of the widgets in the group remain a fixed size and distance from the top-left corner. This is the default for groups derived from Fl_Window and Fl_Pack.
The resizable may be set to the group itself, in which case all of the widgets that are its direct children are resized proportionally. This is the default value for Fl_Group.
The resizable widget defines the resizing box for the group, which could be the group itself or one of the group's direct children. When the group is resized it calculates a new size and position for all of its children. Widgets that are horizontally or vertically inside the dimensions of the box are scaled to the new size. Widgets outside the box are moved.

Note

The resizable of a group **must** be one of

- NULL
- the group itself
- a direct child of the group.

If you set any other widget that is not a direct child of the group as its resizable then the behavior is undefined. This is **not** checked by Fl_Group for historical reasons.

In these examples the gray area is the resizable:

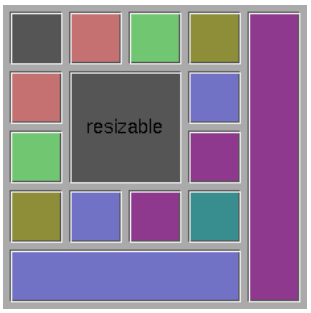


Figure 13.21 before resize

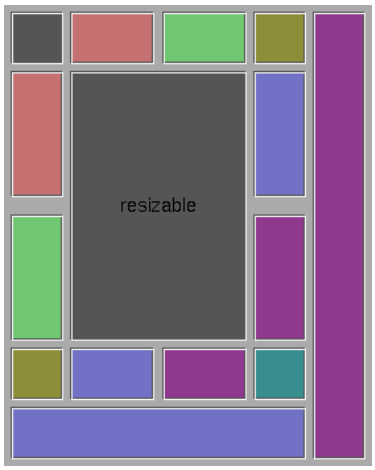


Figure 13.22 after resize

It is possible to achieve any type of resize behavior by using an invisible Fl_Box as the resizable and/or by using a hierarchy of Fl_Group widgets, each with their own resizing strategies.
See the How Does Resizing Work? chapter for more examples and detailed explanation.

Note

The resizable() widget of a window can also affect the window's resizing behavior if Fl_Window::size_range() is not called. Please see Fl_Window::default_size_range() for more information on how the default size range is calculated.

See also

Fl_Window::size_range()

Fl_Window::default_size_range()

13.57.3.31 resize()

```
void Fl_Group::resize (
            int X,
            int Y,
            int W,
            int H )  [override], [virtual]
```

Resizes the Fl_Group widget and all of its children.
The Fl_Group widget first resizes itself, and then it moves and resizes all its children according to the rules documented for Fl_Group::resizable(Fl_Widget*)

See also

Fl_Group::resizable(Fl_Widget*)

Fl_Group::resizable()

Fl_Widget::resize(int,int,int,int)

Reimplemented from Fl_Widget.
Reimplemented in Fl_Input_Choice, Fl_Pack, Fl_Scroll, Fl_Spinner, Fl_Table, Fl_Terminal, Fl_Text_Display, Fl_Tile, Fl_Window, Fl_Help_View, Fl_Overlay_Window, Fl_Tabs, and Fl_Tree.

13.57.3.32 sizes()

```
int * Fl_Group::sizes ( )  [protected]
```

Returns the internal array of widget sizes and positions.
For backward compatibility with FLTK versions before 1.4.
The sizes() array stores the initial positions of widgets as (left, right, top, bottom) quads. The first quad is the group, the second is the resizable (clipped to the group), and the rest are the children. If the group and/or the resizable() is a Fl_Window, then the first (left) and third (top) entries of their respective quads (x,y) are zero.

Deprecated Deprecated since 1.4.0. Please use bounds() instead.

Note

This method will be removed in a future FLTK version (1.5.0 or higher).

Returns

Array of int's with widget positions and sizes. The returned array is only valid until init_sizes() is called or widgets are added to or removed from the group.

Note

Since FLTK 1.4.0 the returned array is a **read-only** and re-ordered copy of the internal bounds() array. Do not change its contents. If you need to rearrange children in a group, do so by resizing the children and call init_sizes().

See also

bounds()

13.57.3.33 update_child()

```
void Fl_Group::update_child (
            Fl_Widget & widget ) const  [protected]
```

Draws a child only if it needs it.
This draws a child widget, if it is not clipped *and* if any damage() bits are set. The damage bits are cleared after drawing.

See also

Fl_Group::draw_child(Fl_Widget& widget) const

The documentation for this class was generated from the following files:

- Fl_Group.H
- Fl_Group.cxx
- forms_compatibility.cxx

13.58 Fl_Help_Dialog Class Reference

The Fl_Help_Dialog widget displays a standard help dialog window using the Fl_Help_View widget.

Public Member Functions

- **Fl_Help_Dialog** ()

  The constructor creates the dialog pictured above.
- int **h** ()

  Returns the position and size of the help dialog.
- void **hide** ()

  Hides the Fl_Help_Dialog window.
- int load (const char *f)

  Loads the specified HTML file into the Fl_Help_View widget.
- void **position** (int xx, int yy)

  Set the screen position of the dialog.
- void **resize** (int xx, int yy, int ww, int hh)

  Change the position and size of the dialog.

- void show ()

  Shows the Fl_Help_Dialog window.
- void **show** (int argc, char **argv)

  *Shows the main Help Dialog Window Delegates call to encapsulated window_ void Fl_Window::show(int argc, char **argv) instance method.*
- Fl_Fontsize **textsize** ()

  Sets or gets the default text size for the help view.
- void textsize (Fl_Fontsize s)

  Sets or gets the default text size for the help view.
- void **topline** (const char *n)

  Sets the top line in the Fl_Help_View widget to the named or numbered line.
- void **topline** (int n)

  Sets the top line in the Fl_Help_View widget to the named or numbered line.
- const char * value () const

  The first form sets the current buffer to the string provided and reformats the text.
- void value (const char *f)

  The first form sets the current buffer to the string provided and reformats the text.
- int **visible** ()

  Returns 1 if the Fl_Help_Dialog window is visible.
- int **w** ()

  Returns the position and size of the help dialog.
- int **x** ()

  Returns the position and size of the help dialog.
- int **y** ()

  Returns the position and size of the help dialog.
- ~**Fl_Help_Dialog** ()

  The destructor destroys the widget and frees all memory that has been allocated for the current file.

13.58.1 Detailed Description

The Fl_Help_Dialog widget displays a standard help dialog window using the Fl_Help_View widget.
The Fl_Help_Dialog class is not part of the FLTK core library, but instead of *fltk_images*. Use `--use-images` when compiling with `fltk-config` .
P

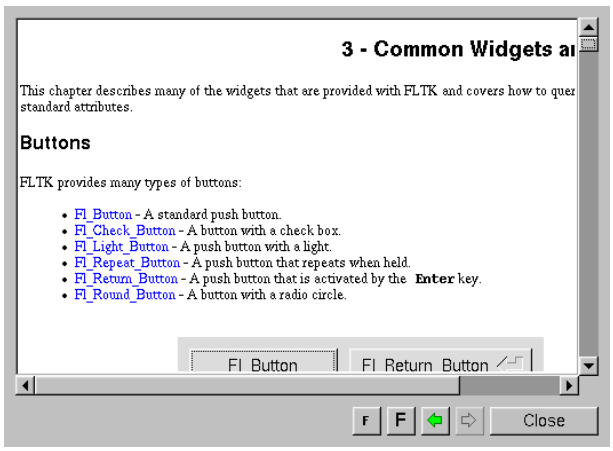


Figure 13.23 Fl_Help_Dialog

13.58.2 Member Function Documentation

13.58.2.1 load()

```
int Fl_Help_Dialog::load (
            const char * f )
```

Loads the specified HTML file into the Fl_Help_View widget.
The filename can also contain a target name ("filename.html#target"). Always use forward slashes as path delimiters, Windows-style backslashes are not supported here.

Parameters

in	*f*	the name and path of an HTML file

Returns

0 on success, -1 on error

See also

Fl_Help_View::load(), fl_load_uri()

13.58.2.2 show()

```
void Fl_Help_Dialog::show ( )
```

Shows the Fl_Help_Dialog window.
Shows the main Help Dialog Window Delegates call to encapsulated window_ void Fl_Window::show() method.

13.58.2.3 textsize()

```
void Fl_Help_Dialog::textsize (
            Fl_Fontsize s )
```

Sets or gets the default text size for the help view.
Sets the internal Fl_Help_View instance text size.
Delegates call to encapsulated view_ void Fl_Help_View::textsize(Fl_Fontsize s) instance method

13.58.2.4 value() [1/2]

```
const char * Fl_Help_Dialog::value ( ) const
```

The first form sets the current buffer to the string provided and reformats the text.
It also clears the history of the "back" and "forward" buttons. The second form returns the current buffer contents.

13.58.2.5 value() [2/2]

```
void Fl_Help_Dialog::value (
            const char * v )
```

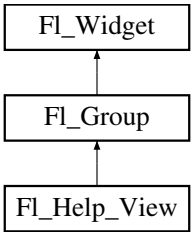
The first form sets the current buffer to the string provided and reformats the text.
It also clears the history of the "back" and "forward" buttons. The second form returns the current buffer contents.
The documentation for this class was generated from the following files:

- Fl_Help_Dialog.H
- Fl_Help_Dialog.cxx
- Fl_Help_Dialog_Dox.cxx

13.59 Fl_Help_View Class Reference

A widget to display formatted text, formatted in a subset of HTML.
`#include <Fl_Help_View.H>`

Inheritance diagram for Fl_Help_View:



Public Member Functions

- void **clear_selection** ()

  Removes the current text selection.
- int copy (int clipboard=1)

  If text is selected in this view, copy it to a clipboard.
- const char ∗ directory () const

  Return the current directory for the text in the buffer.
- const char ∗ filename () const

  Return the current filename for the text in the buffer.
- int find (const char ∗s, int p=0)

  Finds the specified string `s` at starting position `p`.
- Fl_Help_View (int xx, int yy, int ww, int hh, const char ∗l=0)

  Creates the Fl_Help_View widget at the specified position and size.
- int handle (int) override

  Handles events in the widget.
- Fl_Scrollbar ∗ **hscrollbar** ()

  Return pointer to horizontal scrollbar.
- int **leftline** () const

  Get the left position in pixels.
- void leftline (int)

  Scrolls the text to the indicated position, given a pixel column.
- void link (Fl_Help_Func ∗fn)

  Set a callback function for following links.
- int load (const char ∗f)

  Loads the specified file.
- void resize (int, int, int, int) override

  Override the superclass's resize method.
- Fl_Scrollbar ∗ **scrollbar** ()

  Return pointer to vertical scrollbar.
- int scrollbar_size () const

  Get the current size of the scrollbars' troughs, in pixels.
- void scrollbar_size (int newSize)

  Set the pixel size of the scrollbars' troughs to `newSize`, in pixels.
- void **select_all** ()

  Selects all the text in the view.
- int **size** () const

  Return the document height in pixels.
- void **size** (int W, int H)
- int text_selected () const

  Check if the user selected text in this view.

- Fl_Color **textcolor** () const

  Return the current default text color.
- void **textcolor** (Fl_Color c)

  Set the default text color.
- Fl_Font **textfont** () const

  Return the default text font.
- void **textfont** (Fl_Font f)

  Set the default text font.
- Fl_Fontsize **textsize** () const

  Get the default text size.
- void **textsize** (Fl_Fontsize s)

  Set the default text size.
- const char ∗ title () const

  Return the title of the current document.
- int **topline** () const

  Get the current top line in pixels.
- void topline (const char ∗n)

  Scroll the text to the given anchor.
- void topline (int)

  Scrolls the text to the indicated position, given a pixel line.
- const char ∗ **value** () const

  Return a pointer to the internal text buffer.
- void value (const char ∗val)

  Sets the current help text buffer to the string provided and reformats the text.
- ∼Fl_Help_View () override

  Destroys the Fl_Help_View widget.

Public Member Functions inherited from Fl_Group

- Fl_Widget ∗& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget ∗o)

  See void Fl_Group::add(Fl_Widget &w)
- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget ∗const ∗ array () const

  Returns a pointer to the internal array of children.
- Fl_Group const ∗ as_group () const override
- Fl_Group ∗ as_group () override

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget ∗ child (int n) const

  Returns the n'th child.
- int **children** () const

  Returns how many child widgets the group has.
- void clear ()

  Deletes all child widgets from memory recursively.
- unsigned int clip_children () const

Returns the current clipping mode.

- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)

  Removes the widget at `index` from the group and deletes it.
- void end ()

  Exactly the same as current(this->parent()).
- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*
- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.
- void **forms_end** ()

  This is for forms compatibility only.
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

  This does insert(w, find(before)).
- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.
- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.
- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

Gets the current user data (long) argument that is passed to the callback function.

- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ **as_gl_window** () const
- virtual Fl_Window ∗ as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ **as_window** () const
- void bind_deimage (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗cb, void ∗p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.

- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

Sets the image to use as part of the widget label when in the active state.

- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.

- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

Get the spacing between the label and the vertical edge of the widget.

- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Static Public Attributes

- static const char * **copy_menu_text** = "Copy"

  This text may be customized at run-time.

Protected Member Functions

- void draw () override

  Draws the Fl_Help_View widget.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()

  Returns the internal array of widget sizes and positions.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget *, int)

  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)

  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)

  Allow derived groups to act when a child widget is removed from the group.

- int * sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()

  Returns the currently active group.
- static void current (Fl_Group *g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.59.1 Detailed Description

A widget to display formatted text, formatted in a subset of HTML.
The Fl_Help_View widget displays HTML text. Most HTML 2.0 elements are supported, as well as a primitive implementation of tables. GIF, JPEG, and PNG images are displayed inline.
Supported HTML tags:

- A: HREF/NAME
- B
- BODY: BGCOLOR/TEXT/LINK
- BR
- CENTER
- CODE
- DD
- DL
- DT
- EM
- FONT: COLOR/SIZE/FACE=(helvetica/arial/sans/times/serif/symbol/courier)
- H1/H2/H3/H4/H5/H6
- HEAD
- HR
- I

- IMG: SRC/WIDTH/HEIGHT/ALT
- KBD
- LI
- OL
- P
- PRE
- STRONG
- TABLE: TH/TD/TR/BORDER/BGCOLOR/COLSPAN/ALIGN=CENTER|RIGHT|LEFT
- TITLE
- TT
- U
- UL
- VAR

Supported color names:

- black,red,green,yellow,blue,magenta,fuchsia,cyan,aqua,white,gray,grey,lime,maroon,navy,olive,purple,silver,teal.

Supported urls:

- Internal: file:
- External: http: ftp: https: ipp: mailto: news:

Quoted char names:

- Aacute aacute Acirc acirc acute AElig aelig Agrave agrave amp Aring aring Atilde atilde Auml auml
- brvbar bull
- Ccedil ccedil cedil cent copy curren
- dagger deg divide
- Eacute eacute Ecirc ecirc Egrave egrave ETH eth Euml euml euro
- frac12 frac14 frac34
- gt
- Iacute iacute Icirc icirc iexcl Igrave igrave iquest Iuml iuml
- laquo lt
- macr micro middot
- nbsp not Ntilde ntilde
- Oacute oacute Ocirc ocirc Ograve ograve ordf ordm Oslash oslash Otilde otilde Ouml ouml
- para permil plusmn pound
- quot
- raquo reg
- sect shy sup1 sup2 sup3 szlig
- THORN thorn times trade

- Uacute uacute Ucirc ucirc Ugrave ugrave uml Uuml uuml
- Yacute yacute
- yen Yuml yuml

Note

You can't effectively set the box() to FL_NO_BOX, this would result in FL_DOWN_BOX being used as the boxtype of the widget. This is unexpected but can't be changed for backwards compatibility. If you don't want a frame around the widget you can use FL_FLAT_BOX instead.

13.59.2 Constructor & Destructor Documentation

13.59.2.1 Fl_Help_View()

```
Fl_Help_View::Fl_Help_View (
            int xx,
            int yy,
            int ww,
            int hh,
            const char * l = 0 )
```

Creates the Fl_Help_View widget at the specified position and size.

Parameters

in	*xx,yy,ww,hh*	Position and size of the widget
in	*l*	Label for the widget, can be nullptr

13.59.2.2 ~Fl_Help_View()

```
Fl_Help_View::~Fl_Help_View ( ) [override]
```

Destroys the Fl_Help_View widget.
The destructor destroys the widget and frees all memory that has been allocated for the current document.

13.59.3 Member Function Documentation

13.59.3.1 copy()

```
int Fl_Help_View::copy (
            int clipboard = 1 )
```

If text is selected in this view, copy it to a clipboard.

Parameters

in	*clipboard*	for x11 only, 0=selection buffer, 1=clipboard, 2=both

Returns

1 if text is selected, 0 if no text is selected

13.59.3.2 directory()

```
const char * Fl_Help_View::directory ( ) const
```

Return the current directory for the text in the buffer.
Fl_Help_View remains the owner of the allocated memory. If the directory changes, the returned pointer will become stale.

Returns

nullptr if the directory name is empty

13.59.3.3 draw()

```
void Fl_Help_View::draw (
            void ) [override], [protected], [virtual]
```

Draws the Fl_Help_View widget.
Reimplemented from Fl_Group.

13.59.3.4 filename()

```
const char * Fl_Help_View::filename ( ) const
```

Return the current filename for the text in the buffer.
Fl_Help_View remains the owner of the allocated memory. If the filename changes, the returned pointer will become stale.

Returns

nullptr if the filename is empty

13.59.3.5 find()

```
int Fl_Help_View::find (
            const char * s,
            int p = 0 )
```

Finds the specified string `s` at starting position `p`.
The argument `p` and the return value are offsets in Fl_Help_View::value(), counting from 0. If `p` is out of range, 0 is used.
The string comparison is simple but honors some special cases:

- the specified string `s` must be in UTF-8 encoding
- HTML tags in value() are filtered (not compared as such, they never match)
- HTML entities like '<' or '&x#20ac;' are converted to Unicode (UTF-8)
- ASCII characters (7-bit, $<$ 0x80) are compared case insensitive
- every newline (LF, '\n') in value() is treated like a single space
- all other strings are compared as-is (byte by byte)

Parameters

in	*s*	search string in UTF-8 encoding
in	*p*	starting position for search (0,...), Default = 0

Returns

the matching position or -1 if not found

13.59.3.6 handle()

```
int Fl_Help_View::handle (
            int event ) [override], [virtual]
```

Handles events in the widget.

Parameters

in	*event*	Event to handle.

Returns

1 if the event was handled, 0 otherwise.

Reimplemented from Fl_Group.

13.59.3.7 leftline()

```
void Fl_Help_View::leftline (
            int left )
```

Scrolls the text to the indicated position, given a pixel column.
If the given pixel value `left` is out of range, then the text is scrolled to the left or right side of the document, resp.

Parameters

in	*left*	left column number in pixels (0 = left side)

13.59.3.8 link()

```
void Fl_Help_View::link (
            Fl_Help_Func * fn )
```

Set a callback function for following links.
This method assigns a callback function to use when a link is followed or a file is loaded (via Fl_Help_View::load()) that requires a different file or path.
The callback function receives a pointer to the Fl_Help_View widget and the URI or full pathname for the file in question. It must return a pathname that can be opened as a local file or nullptr:
`const char *fn(Fl_Widget *w, const char *uri);`
The link function can be used to retrieve remote or virtual documents, returning a temporary file that contains the actual data. If the link function returns nullptr, the value of the Fl_Help_View widget will remain unchanged.
If the link callback cannot handle the URI scheme, it should return the uri value unchanged or set the value() of the widget before returning nullptr.

Parameters

in	*fn*	Pointer to the callback function

13.59.3.9 load()

```
int Fl_Help_View::load (
            const char * f )
```

Loads the specified file.
This method loads the specified file or URL. The filename may end in a `#name` style target.
If the URL starts with *ftp*, *http*, *https*, *ipp*, *mailto*, or *news*, followed by a colon, FLTK will use fl_open_uri() to show the requested page in an external browser.
In all other cases, the URL is interpreted as a filename. The file is read and displayed in this browser. Note that Windows style backslashes are not supported in the file name.

Parameters

in	*f*	filename or URL

Returns

0 on success, -1 on error

See also

fl_open_uri()

13.59.3.10 resize()

```
void Fl_Help_View::resize (
            int xx,
            int yy,
            int ww,
            int hh )  [override], [virtual]
```

Override the superclass's resize method.

Parameters

in	*xx,yy,ww,hh*	New position and size of the widget

Reimplemented from Fl_Group.

13.59.3.11 scrollbar_size() [1/2]

```
int Fl_Help_View::scrollbar_size ( ) const
```

Get the current size of the scrollbars' troughs, in pixels.
If this value is zero (default), this widget will use the Fl::scrollbar_size() value as the scrollbar's width.

Returns

Scrollbar size in pixels, or 0 if the global Fl::scrollbar_size() is being used.

See also

Fl::scrollbar_size(int)

13.59.3.12 scrollbar_size() [2/2]

```
void Fl_Help_View::scrollbar_size (
            int newSize )
```

Set the pixel size of the scrollbars' troughs to `newSize`, in pixels.
Normally you should not need this method, and should use Fl::scrollbar_size(int) instead to manage the size of ALL your widgets' scrollbars. This ensures your application has a consistent UI, is the default behavior, and is normally what you want.
Only use THIS method if you really need to override the global scrollbar size. The need for this should be rare.
Setting `newSize` to the special value of 0 causes the widget to track the global Fl::scrollbar_size(), which is the default.

Parameters

in	*newSize*	Sets the scrollbar size in pixels. If 0 (default), scrollbar size tracks the global Fl::scrollbar_size()

See also

Fl::scrollbar_size()

13.59.3.13 text_selected()

```
int Fl_Help_View::text_selected ( ) const
```

Check if the user selected text in this view.

Returns

1 if text is selected, 0 if no text is selected

13.59.3.14 title()

```
const char * Fl_Help_View::title ( ) const
```

Return the title of the current document.
Fl_Help_View remains the owner of the allocated memory. If the document changes, the returned pointer will become stale.

Returns

empty string if the directory name is empty

13.59.3.15 topline() [1/2]

```
void Fl_Help_View::topline (
            const char * anchor )
```

Scroll the text to the given anchor.

Parameters

in	*anchor*	scroll to this named anchor

13.59.3.16 topline() [2/2]

```
void Fl_Help_View::topline (
            int top )
```

Scrolls the text to the indicated position, given a pixel line.
If the given pixel value `top` is out of range, then the text is scrolled to the top or bottom of the document, resp.

Parameters

in	*top*	top line number in pixels (0 = start of document)

13.59.3.17 value()

```
void Fl_Help_View::value (
            const char * val )
```

Sets the current help text buffer to the string provided and reformats the text.
The provided character string `val` is copied internally and will be freed when value() is called again, or when the widget is destroyed.
If `val` is nullptr, then the widget is cleared.

Parameters

in	*val*	Text to view, or nullptr to clear the widget, Fl_Help_View will creat a local copy of the string.

The documentation for this class was generated from the following files:

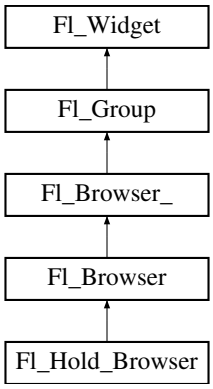
- Fl_Help_View.H
- Fl_Help_View.cxx

13.60 Fl_Hold_Browser Class Reference

The Fl_Hold_Browser is a subclass of Fl_Browser which lets the user select a single item, or no items by clicking on the empty space.

`#include <Fl_Hold_Browser.H>`

Inheritance diagram for Fl_Hold_Browser:



Public Member Functions

- Fl_Hold_Browser (int X, int Y, int W, int H, const char *L=0)

  Creates a new Fl_Hold_Browser widget using the given position, size, and label string.

Public Member Functions inherited from Fl_Browser

- void add (const char *newtext, void *d=0)

  Adds a new line to the end of the browser.
- void bottomline (int line)

  Scrolls the browser so the bottom item in the browser is showing the specified `line`.
- void clear ()

  Removes all the lines in the browser.
- char column_char () const

  Gets the current column separator character.
- void column_char (char c)

  Sets the column separator to c.
- const int * column_widths () const

  Gets the current column width array.
- void column_widths (const int *arr)

  Sets the current array to `arr`.
- void * data (int line) const

  Returns the user data() for specified `line`.
- void data (int line, void *d)

  Sets the user data for specified `line` to `d`.
- void display (int line, int val=1)

  For back compatibility.
- int displayed (int line) const

  Returns non-zero if `line` has been scrolled to a position where it is being displayed.
- Fl_Browser (int X, int Y, int W, int H, const char *L=0)

  The constructor makes an empty browser.
- char format_char () const

  Gets the current format code prefix character, which by default is '@'.

- void format_char (char c)

  Sets the current format code prefix character to `c`.
- void hide () override

  Hides the entire Fl_Browser widget – opposite of show().
- void hide (int line)

  Makes `line` *invisible, preventing selection by the user.*
- Fl_Image * icon (int line) const

  Returns the icon currently defined for `line`.
- void icon (int line, Fl_Image *icon)

  Set the image icon for `line` *to the value* `icon`.
- void insert (int line, const char *newtext, void *d=0)

  Insert a new entry whose label is `newtext` *above given* `line`*, optional data* `d`.
- void lineposition (int line, Fl_Line_Position pos)

  Updates the browser so that `line` *is shown at position* `pos`.
- int load (const char *filename)

  Clears the browser and reads the file, adding each line from the file to the browser.
- void make_visible (int line)

  Make the item at the specified `line` *visible().*
- void middleline (int line)

  Scrolls the browser so the middle item in the browser is showing the specified `line`.
- void move (int to, int from)

  Line `from` *is removed and reinserted at* `to`.
- void remove (int line)

  Remove entry for given `line` *number, making the browser one line shorter.*
- void remove_icon (int line)

  Removes the icon for `line`.
- void **replace** (int a, const char *b)

  For back compatibility only.
- int select (int line, int val=1)

  Sets the selection state of the item at `line` *to the value* `val`.
- int selected (int line) const

  Returns 1 if specified `line` *is selected, 0 if not.*
- void show () override

  Shows the entire Fl_Browser widget – opposite of hide().
- void show (int line)

  Makes `line` *visible, and available for selection by user.*
- int size () const

  Returns how many lines are in the browser.
- void **size** (int W, int H)
- void swap (int a, int b)

  Swaps two browser lines `a` *and* `b`.
- const char * text (int line) const

  Returns the label text for the specified `line`.
- void text (int line, const char *newtext)

  Sets the text for the specified `line` *to* `newtext`.
- Fl_Fontsize **textsize** () const

  Gets the default text size (in pixels) for the lines in the browser.
- void textsize (Fl_Fontsize newSize)

  Sets the default text size (in pixels) for the lines in the browser to `newSize`.
- int topline () const

  Returns the line that is currently visible at the top of the browser.

- void topline (int line)

  Scrolls the browser so the top item in the browser is showing the specified `line`.
- int value () const

  Returns the line number of the currently selected line, or 0 if none selected.
- void value (int line)

  Sets the browser's value(), which selects the specified `line`.
- int visible (int line) const

  Returns non-zero if the specified `line` is visible, 0 if hidden.
- ~**Fl_Browser** ()

  The destructor deletes all list items and destroys the browser.

Public Member Functions inherited from Fl_Browser_

- int deselect (int docallbacks=0)

  Deselects all items in the list and returns 1 if the state changed or 0 if it did not.
- void display (void *item)

  Displays the `item`, scrolling the list as necessary.
- int handle (int event) override

  Handles the `event` within the normal widget bounding box.
- uchar **has_scrollbar** () const

  Returns the current scrollbar mode, see Fl_Browser_::has_scrollbar(uchar)
- void has_scrollbar (uchar mode)

  Sets whether the widget should have scrollbars or not (default Fl_Browser_::BOTH).
- int hposition () const

  Gets the horizontal scroll position of the list as a pixel position `pos`.
- void hposition (int)

  Sets the horizontal scroll position of the list to pixel position `pos`.
- int linespacing () const

  Return the height of additional spacing between browser lines.
- void linespacing (int pixels)

  Add some space between browser lines.
- int position () const
- void position (int pos)
- void **position** (int x, int y)
- void resize (int X, int Y, int W, int H) override

  Repositions and/or resizes the browser.
- void scrollbar_left ()

  Moves the vertical scrollbar to the lefthand side of the list.
- void scrollbar_right ()

  Moves the vertical scrollbar to the righthand side of the list.
- int scrollbar_size () const

  Gets the current size of the scrollbars' troughs, in pixels.
- void scrollbar_size (int newSize)

  Sets the pixel size of the scrollbars' troughs to `newSize`, in pixels.
- int select (void *item, int val=1, int docallbacks=0)

  Sets the selection state of `item` to `val`, and returns 1 if the state changed or 0 if it did not.
- int select_only (void *item, int docallbacks=0)

  Selects `item` and returns 1 if the state changed or 0 if it did not.
- void sort (int flags=0)

  Sort the items in the browser based on `flags`.
- Fl_Color **textcolor** () const

Gets the default text color for the lines in the browser.

- void **textcolor** (Fl_Color col)

  Sets the default text color for the lines in the browser to color `col`.
- Fl_Font textfont () const

  Gets the default text font for the lines in the browser.
- void **textfont** (Fl_Font font)

  Sets the default text font for the lines in the browser to `font`.
- Fl_Fontsize **textsize** () const

  Gets the default text size (in pixels) for the lines in the browser.
- void **textsize** (Fl_Fontsize newSize)

  Sets the default text size (in pixels) for the lines in the browser to `size`.
- int vposition () const

  Gets the vertical scroll position of the list as a pixel position `pos`.
- void vposition (int pos)

  Sets the vertical scroll position of the list to pixel position `pos`.

Public Member Functions inherited from Fl_Group

- Fl_Widget *& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget *o)

  See void Fl_Group::add(Fl_Widget &w)
- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget *const * array () const

  Returns a pointer to the internal array of children.
- Fl_Group const * as_group () const override
- Fl_Group * as_group () override

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const

  Returns the n'th child.
- int **children** () const

  Returns how many child widgets the group has.
- void clear ()

  Deletes all child widgets from memory recursively.
- unsigned int clip_children () const

  Returns the current clipping mode.
- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)

  Removes the widget at `index` from the group and deletes it.
- void end ()

  Exactly the same as current(this->parent()).
- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*
- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.

- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.
- void **forms_end** ()

  This is for forms compatibility only.
- int handle (int) override

  Handles the specified event.
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

  This does insert(w, find(before)).
- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.
- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.
- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- void resize (int, int, int, int) override

  Resizes the Fl_Group widget and all of its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Window * as_window ()

Returns an Fl_Window pointer if this widget is an Fl_Window.

- virtual Fl_Window const ∗ **as_window** () const
- void bind_deimage (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗cb, void ∗p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget ∗w) const

  Checks if w is a child of this widget.
- void copy_label (const char ∗new_label)

  Sets the current label.
- void copy_tooltip (const char ∗text)

Sets the current tooltip text.

- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.

- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

Marks the value of the widget as changed.

- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.

- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Public Types inherited from Fl_Browser

- enum Fl_Line_Position { **TOP** , **BOTTOM** , **MIDDLE** }

  For internal use only?

Public Types inherited from Fl_Browser_

- enum {
  HORIZONTAL = 1 , VERTICAL = 2 , BOTH = 3 , ALWAYS_ON = 4 ,
  HORIZONTAL_ALWAYS = 5 , VERTICAL_ALWAYS = 6 , BOTH_ALWAYS = 7 }

  Values for has_scrollbar().

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()

  Returns the currently active group.
- static void current (Fl_Group *g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Public Attributes inherited from Fl_Browser_

- Fl_Scrollbar hscrollbar

  Horizontal scrollbar.
- Fl_Scrollbar scrollbar

  Vertical scrollbar.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Browser

- FL_BLINE * _remove (int line)

  Removes the item at the specified `line`.
- const void * **bline_data** (const FL_BLINE *b) const

  Get FL_BLINE data.
- void *& **bline_data** (FL_BLINE *b) const

  Get writable reference to FL_BLINE data.
- char **bline_flags** (const FL_BLINE *b) const

  Get FL_BLINE flags.
- char & **bline_flags** (FL_BLINE *b) const

  Get writable reference to FL_BLINE flags.
- const Fl_Image * **bline_icon** (const FL_BLINE *b) const

  Get FL_BLINE icon.
- Fl_Image *& **bline_icon** (FL_BLINE *b) const

  Get writable reference to FL_BLINE icon.
- short bline_length (const FL_BLINE *b) const

  Get FL_BLINE allocated text buffer size (excl.
- const char * **bline_txt** (const FL_BLINE *b) const

  Get FL_BLINE text.
- char * **bline_txt** (FL_BLINE *b) const

  Get writable reference to FL_BLINE text.
- FL_BLINE * find_line (int line) const

  Returns the item for specified `line`.
- int full_height () const override

  The height of the entire list of all visible() items in pixels.
- int incr_height () const override

  The default 'average' item height (including inter-item spacing) in pixels.
- void insert (int line, FL_BLINE *item)

  Insert specified `item` above `line`.
- void * item_at (int line) const override

  Return the item at specified `line`.
- void item_draw (void *item, int X, int Y, int W, int H) const override

  Draws `item` at the position specified by `X Y W H`.
- void * item_first () const override

Returns the very first item in the list.

- int item_height (void *item) const override

  Returns height of `item` in pixels.
- void * item_last () const override

  Returns the very last item in the list.
- void * item_next (void *item) const override

  Returns the next item after `item`.
- void * item_prev (void *item) const override

  Returns the previous item before `item`.
- void item_select (void *item, int val) override

  Change the selection state of `item` to the value `val`.
- int item_selected (void *item) const override

  See if `item` is selected.
- void item_swap (void *a, void *b) override

  Swap the items `a` and `b`.
- const char * item_text (void *item) const override

  Returns the label text for `item`.
- int item_width (void *item) const override

  Returns width of `item` in pixels.
- int lineno (void *item) const

  Returns line number corresponding to `item`, or zero if not found.
- void swap (FL_BLINE *a, FL_BLINE *b)

  Swap the two items `a` and `b`.

Protected Member Functions inherited from Fl_Browser_

- void bbox (int &X, int &Y, int &W, int &H) const

  Returns the bounding box for the interior of the list's display window, inside the scrollbars.
- void deleting (void *item)

  This method should be used when `item` is being deleted from the list.
- int displayed (void *item) const

  Returns non-zero if `item` has been scrolled to a position where it is being displayed.
- void draw () override

  Draws the list within the normal widget bounding box.
- void * find_item (int ypos)

  This method returns the item under mouse y position `ypos`.
- Fl_Browser_ (int X, int Y, int W, int H, const char *L=0)

  The constructor makes an empty browser.
- virtual int full_width () const

  This method may be provided by the subclass to indicate the full width of the item list, in pixels.
- void inserting (void *a, void *b)

  This method should be used when an item is in the process of being inserted into the list.
- virtual int item_quick_height (void *item) const

  This method may be provided by the subclass to return the height of the `item`, in pixels.
- int leftedge () const

  This method returns the X position of the left edge of the list area after adjusting for the scrollbar and border, if any.
- void new_list ()

  This method should be called when the list data is completely replaced or cleared.
- void redraw_line (void *item)

  This method should be called when the contents of `item` has changed, but not its height.
- void redraw_lines ()

This method will cause the entire list to be redrawn.

- void replacing (void *a, void *b)

  This method should be used when item a is being replaced by item b.
- void * selection () const

  Returns the item currently selected, or NULL if there is no selection.
- void swapping (void *a, void *b)

  This method should be used when two items a and b are being swapped.
- void * **top** () const

  Returns the item that appears at the top of the list.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()

  Returns the internal array of widget sizes and positions.
- void draw () override

  Draws the widget.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget *, int)

  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)

  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)

  Allow derived groups to act when a child widget is removed from the group.
- int * sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

Draws the widget's label at the defined label position.

- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Static Protected Attributes inherited from Fl_Browser

- static constexpr char **BLINE_NOTDISPLAYED** = 2
- static constexpr char **BLINE_SELECTED** = 1

13.60.1 Detailed Description

The Fl_Hold_Browser is a subclass of Fl_Browser which lets the user select a single item, or no items by clicking on the empty space.

Figure 13.24 Fl_Hold_Browser

As long as the mouse button is held down the item pointed to by it is highlighted, and this highlighting remains on when the mouse button is released. Normally the callback is done when the user releases the mouse, but you can change this with when().
See Fl_Browser for methods to add and remove lines from the browser.

13.60.2 Constructor & Destructor Documentation

13.60.2.1 Fl_Hold_Browser()

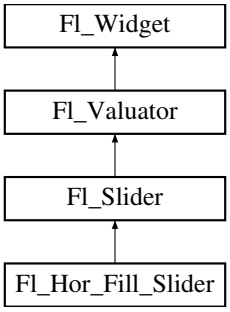
```
Fl_Hold_Browser::Fl_Hold_Browser (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

Creates a new Fl_Hold_Browser widget using the given position, size, and label string.
The default boxtype is FL_DOWN_BOX. The constructor specializes Fl_Browser() by setting the type to FL_↩ HOLD_BROWSER. The destructor destroys the widget and frees all memory that has been allocated.
The documentation for this class was generated from the following files:

- Fl_Hold_Browser.H
- Fl_Browser.cxx

13.61 Fl_Hor_Fill_Slider Class Reference

Inheritance diagram for Fl_Hor_Fill_Slider:



Public Member Functions

- **Fl_Hor_Fill_Slider** (int X, int Y, int W, int H, const char ∗L=0)

Public Member Functions inherited from Fl_Slider

- void bounds (double a, double b)

  Sets the minimum (a) and maximum (b) values for the valuator widget.
- Fl_Slider (int X, int Y, int W, int H, const char ∗L=0)

  Creates a new Fl_Slider widget using the given position, size, and label string.
- **Fl_Slider** (uchar t, int X, int Y, int W, int H, const char ∗L)

  Creates a new Fl_Slider widget using the given type, position, size, and label string.
- int handle (int) override

  Handles the specified event.
- uchar num_ticks () const

  Gets the number of tick marks.
- Scale_Type scale () const

  Gets the scale type.
- void scale (Scale_Type s)

  Sets the scale type.
- int scrollvalue (int pos, int size, int first, int total)

  Sets the size and position of the sliding knob in the box.
- Fl_Boxtype **slider** () const

  Gets the slider box type.
- void **slider** (Fl_Boxtype c)

  Sets the slider box type.
- float **slider_size** () const

  Get the dimensions of the moving piece of slider.
- void slider_size (double v)

  Set the dimensions of the moving piece of slider.
- uchar ticks () const

  Gets the tick mark position.
- void ticks (uchar t, uchar num_ticks=11)

  Sets the tick mark position for Fl_Nice_Slider.

Public Member Functions inherited from Fl_Valuator

- void **bounds** (double a, double b)

  Sets the minimum (a) and maximum (b) values for the valuator widget.
- double **clamp** (double)

  Clamps the passed value to the valuator range.
- virtual int format (char ∗)

  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.
- virtual std::string format_str ()

  C++11 API for Fl_Valuator::format(char∗ buffer).
- double increment (double, int)

  Adds n times the step value to the passed value.
- double maximum () const

  Gets the maximum value for the valuator.
- void maximum (double a)

  Sets the maximum value for the valuator.
- double minimum () const

  Gets the minimum value for the valuator.
- void minimum (double a)

  Sets the minimum value for the valuator.
- void precision (int digits)

  Sets the step value to 1.0 / 10^{digits}.
- void range (double a, double b)

  Sets the minimum and maximum values for the valuator.
- double round (double)

  Round the passed value to the nearest step increment.
- double step () const

  Gets or sets the step value.
- void **step** (double a, int b)

  See double Fl_Valuator::step() const
- void **step** (double s)

  See double Fl_Valuator::step() const.
- void **step** (int a)

  See double Fl_Valuator::step() const
- double value () const

  Gets the floating point(double) value.
- int value (double)

  Sets the current value.
- ∼**Fl_Valuator** () override

  Destructor is accessible despite protected constructor.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

Returns whether the widget and all of its parents are active.

- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ **as_gl_window** () const
- virtual Fl_Group ∗ as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const ∗ **as_group** () const
- virtual Fl_Window ∗ as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ **as_window** () const
- void bind_deimage (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗cb, void ∗p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

Sets a widget to accept input.

- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.

- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

Returns a pointer to the parent widget.

- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.

- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Public Types inherited from Fl_Slider

- enum Scale_Type : uchar { LINEAR_SCALE = 0 , LOG_SCALE }

  Bitset for tick mark positions for Fl_Nice_Slider.
- enum Tick_Position : uchar {
  TICKS_NONE = 0x00 , TICKS_BELOW = 0x01 , TICKS_ABOVE = 0x02 , TICKS_LEFT = 0x01 ,
  TICKS_RIGHT = 0x02 }

  Bitset for tick mark positions for Fl_Nice_Slider.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Slider

- void draw () override

  Draws the widget.
- void **draw** (int, int, int, int)
- void draw_ticks (const Fl_Rect &r, int S)

  Draw tick marks.
- int **handle** (int, int, int, int, int)
- double increment_lin_log (double v, int n, int range)

  Adds a sensible step value to the passed value.
- double position_to_value (double pos) const

  Converts a normalized position (0.0 to 1.0) to a value based on the current scale type.
- double value_to_position (double val) const

  Converts a value to a normalized position (0.0 to 1.0) based on the current scale type.

Protected Member Functions inherited from Fl_Valuator

- Fl_Valuator (int X, int Y, int W, int H, const char *L)

  Creates a new Fl_Valuator widget using the given position, size, and label string.
- void **handle_drag** (double newvalue)

  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.
- void **handle_push** ()

  Stores the current value in the previous value.
- void **handle_release** ()

  Called after an FL_WHEN_RELEASE event is received and before the callback.
- int **horizontal** () const

  Tells if the valuator is an FL_HORIZONTAL one.

- double **previous_value** () const

  Gets the previous floating point value before an event changed it.
- void **set_value** (double v)

  Sets the current floating point value.
- double **softclamp** (double)

  Clamps the value, but accepts v if the previous value is not already out of range.
- virtual void value_damage ()

  Asks for partial redraw.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

The documentation for this class was generated from the following files:

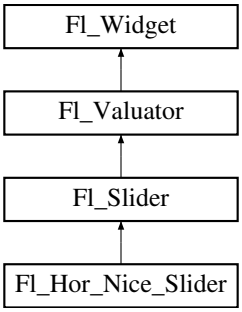
- Fl_Hor_Fill_Slider.H
- Fl_Slider.cxx

13.62 Fl_Hor_Nice_Slider Class Reference

Single thumb tab slider.

`#include <Fl_Hor_Nice_Slider.H>`

Inheritance diagram for Fl_Hor_Nice_Slider:



Public Member Functions

- **Fl_Hor_Nice_Slider** (int X, int Y, int W, int H, const char ∗L=0)

Public Member Functions inherited from Fl_Slider

- void bounds (double a, double b)

  Sets the minimum (a) and maximum (b) values for the valuator widget.
- Fl_Slider (int X, int Y, int W, int H, const char ∗L=0)

  Creates a new Fl_Slider widget using the given position, size, and label string.
- **Fl_Slider** (uchar t, int X, int Y, int W, int H, const char ∗L)

  Creates a new Fl_Slider widget using the given type, position, size, and label string.
- int handle (int) override

  Handles the specified event.
- uchar num_ticks () const

  Gets the number of tick marks.
- Scale_Type scale () const

  Gets the scale type.
- void scale (Scale_Type s)

  Sets the scale type.
- int scrollvalue (int pos, int size, int first, int total)

  Sets the size and position of the sliding knob in the box.
- Fl_Boxtype **slider** () const

  Gets the slider box type.
- void **slider** (Fl_Boxtype c)

  Sets the slider box type.
- float **slider_size** () const

  Get the dimensions of the moving piece of slider.
- void slider_size (double v)

  Set the dimensions of the moving piece of slider.
- uchar ticks () const

  Gets the tick mark position.
- void ticks (uchar t, uchar num_ticks=11)

  Sets the tick mark position for Fl_Nice_Slider.

Public Member Functions inherited from Fl_Valuator

- void **bounds** (double a, double b)

  Sets the minimum (a) and maximum (b) values for the valuator widget.
- double **clamp** (double)

  Clamps the passed value to the valuator range.
- virtual int format (char ∗)

  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.
- virtual std::string format_str ()

  C++11 API for Fl_Valuator::format(char∗ buffer).
- double increment (double, int)

  Adds n times the step value to the passed value.
- double maximum () const

  Gets the maximum value for the valuator.
- void maximum (double a)

  Sets the maximum value for the valuator.
- double minimum () const

  Gets the minimum value for the valuator.
- void minimum (double a)

  Sets the minimum value for the valuator.
- void precision (int digits)

  Sets the step value to 1.0 / 10^{digits}.
- void range (double a, double b)

  Sets the minimum and maximum values for the valuator.
- double round (double)

  Round the passed value to the nearest step increment.
- double step () const

  Gets or sets the step value.
- void **step** (double a, int b)

  See double Fl_Valuator::step() const
- void **step** (double s)

  See double Fl_Valuator::step() const.
- void **step** (int a)

  See double Fl_Valuator::step() const
- double value () const

  Gets the floating point(double) value.
- int value (double)

  Sets the current value.
- ∼**Fl_Valuator** () override

  Destructor is accessible despite protected constructor.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

Returns whether the widget and all of its parents are active.

- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ **as_gl_window** () const
- virtual Fl_Group ∗ as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const ∗ **as_group** () const
- virtual Fl_Window ∗ as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ **as_window** () const
- void bind_deimage (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗cb, void ∗p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

Sets a widget to accept input.

- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.

- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

Returns a pointer to the parent widget.

- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.

- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Public Types inherited from Fl_Slider

- enum Scale_Type : uchar { LINEAR_SCALE = 0 , LOG_SCALE }

  Bitset for tick mark positions for Fl_Nice_Slider.
- enum Tick_Position : uchar {
  TICKS_NONE = 0x00 , TICKS_BELOW = 0x01 , TICKS_ABOVE = 0x02 , TICKS_LEFT = 0x01 ,
  TICKS_RIGHT = 0x02 }

  Bitset for tick mark positions for Fl_Nice_Slider.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Slider

- void draw () override

  Draws the widget.
- void **draw** (int, int, int, int)
- void draw_ticks (const Fl_Rect &r, int S)

  Draw tick marks.
- int **handle** (int, int, int, int, int)
- double increment_lin_log (double v, int n, int range)

  Adds a sensible step value to the passed value.
- double position_to_value (double pos) const

  Converts a normalized position (0.0 to 1.0) to a value based on the current scale type.
- double value_to_position (double val) const

  Converts a value to a normalized position (0.0 to 1.0) based on the current scale type.

Protected Member Functions inherited from Fl_Valuator

- Fl_Valuator (int X, int Y, int W, int H, const char *L)

  Creates a new Fl_Valuator widget using the given position, size, and label string.
- void **handle_drag** (double newvalue)

  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.
- void **handle_push** ()

  Stores the current value in the previous value.
- void **handle_release** ()

  Called after an FL_WHEN_RELEASE event is received and before the callback.
- int **horizontal** () const

  Tells if the valuator is an FL_HORIZONTAL one.

- double **previous_value** () const

  Gets the previous floating point value before an event changed it.
- void **set_value** (double v)

  Sets the current floating point value.
- double **softclamp** (double)

  Clamps the value, but accepts v if the previous value is not already out of range.
- virtual void value_damage ()

  Asks for partial redraw.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

13.62.1 Detailed Description

Single thumb tab slider.

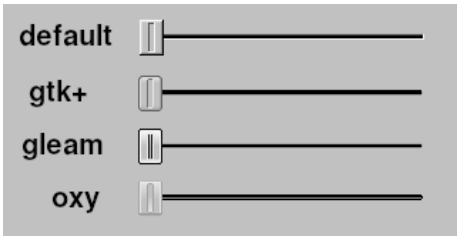


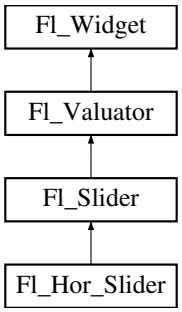
Figure 13.25 Fl_Hor_Nice_Slider with various Fl::scheme() values

The documentation for this class was generated from the following files:

- Fl_Hor_Nice_Slider.H
- Fl_Slider.cxx

13.63 Fl_Hor_Slider Class Reference

Horizontal Slider class.
`#include <Fl_Hor_Slider.H>`
Inheritance diagram for Fl_Hor_Slider:



Public Member Functions

- **Fl_Hor_Slider** (int X, int Y, int W, int H, const char ∗l=0)

  Creates a new Fl_Hor_Slider widget using the given position, size, and label string.

Public Member Functions inherited from Fl_Slider

- void bounds (double a, double b)

  Sets the minimum (a) and maximum (b) values for the valuator widget.
- Fl_Slider (int X, int Y, int W, int H, const char ∗L=0)

  Creates a new Fl_Slider widget using the given position, size, and label string.
- **Fl_Slider** (uchar t, int X, int Y, int W, int H, const char ∗L)

  Creates a new Fl_Slider widget using the given type, position, size, and label string.
- int handle (int) override

  Handles the specified event.
- uchar num_ticks () const

  Gets the number of tick marks.
- Scale_Type scale () const

  Gets the scale type.
- void scale (Scale_Type s)

Sets the scale type.

- int scrollvalue (int pos, int size, int first, int total)

  Sets the size and position of the sliding knob in the box.
- Fl_Boxtype **slider** () const

  Gets the slider box type.
- void **slider** (Fl_Boxtype c)

  Sets the slider box type.
- float **slider_size** () const

  Get the dimensions of the moving piece of slider.
- void slider_size (double v)

  Set the dimensions of the moving piece of slider.
- uchar ticks () const

  Gets the tick mark position.
- void ticks (uchar t, uchar num_ticks=11)

  Sets the tick mark position for Fl_Nice_Slider.

Public Member Functions inherited from Fl_Valuator

- void **bounds** (double a, double b)

  Sets the minimum (a) and maximum (b) values for the valuator widget.
- double **clamp** (double)

  Clamps the passed value to the valuator range.
- virtual int format (char ∗)

  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.
- virtual std::string format_str ()

  C++11 API for Fl_Valuator::format(char∗ buffer).
- double increment (double, int)

  Adds n times the step value to the passed value.
- double maximum () const

  Gets the maximum value for the valuator.
- void maximum (double a)

  Sets the maximum value for the valuator.
- double minimum () const

  Gets the minimum value for the valuator.
- void minimum (double a)

  Sets the minimum value for the valuator.
- void precision (int digits)

  Sets the step value to $1.0 / 10^{digits}$.
- void range (double a, double b)

  Sets the minimum and maximum values for the valuator.
- double round (double)

  Round the passed value to the nearest step increment.
- double step () const

  Gets or sets the step value.
- void **step** (double a, int b)

  See double Fl_Valuator::step() const
- void **step** (double s)

  See double Fl_Valuator::step() const.
- void **step** (int a)

  See double Fl_Valuator::step() const

- double value () const

  Gets the floating point(double) value.
- int value (double)

  Sets the current value.
- ~**Fl_Valuator** () override

  Destructor is accessible despite protected constructor.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

Sets the current callback function and managed user data for the widget.

- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.

- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

Gets the font size in pixels.

- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.

- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Public Types inherited from Fl_Slider

- enum Scale_Type : uchar { LINEAR_SCALE = 0 , LOG_SCALE }

  Bitset for tick mark positions for Fl_Nice_Slider.
- enum Tick_Position : uchar {
  TICKS_NONE = 0x00 , TICKS_BELOW = 0x01 , TICKS_ABOVE = 0x02 , TICKS_LEFT = 0x01 ,
  TICKS_RIGHT = 0x02 }

  Bitset for tick mark positions for Fl_Nice_Slider.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget ∗widget, void ∗data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char ∗t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char ∗, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Slider

- void draw () override

  Draws the widget.
- void **draw** (int, int, int, int)
- void draw_ticks (const Fl_Rect &r, int S)

  Draw tick marks.
- int **handle** (int, int, int, int, int)
- double increment_lin_log (double v, int n, int range)

  Adds a sensible step value to the passed value.
- double position_to_value (double pos) const

  Converts a normalized position (0.0 to 1.0) to a value based on the current scale type.
- double value_to_position (double val) const

  Converts a value to a normalized position (0.0 to 1.0) based on the current scale type.

Protected Member Functions inherited from Fl_Valuator

- Fl_Valuator (int X, int Y, int W, int H, const char *L)

  Creates a new Fl_Valuator widget using the given position, size, and label string.
- void **handle_drag** (double newvalue)

  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.
- void **handle_push** ()

  Stores the current value in the previous value.
- void **handle_release** ()

  Called after an FL_WHEN_RELEASE event is received and before the callback.
- int **horizontal** () const

  Tells if the valuator is an FL_HORIZONTAL one.
- double **previous_value** () const

  Gets the previous floating point value before an event changed it.
- void **set_value** (double v)

  Sets the current floating point value.
- double **softclamp** (double)

  Clamps the value, but accepts v if the previous value is not already out of range.
- virtual void value_damage ()

  Asks for partial redraw.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

Internal use only.

- void x (int v)

  Internal use only.

- void y (int v)

  Internal use only.

13.63.1 Detailed Description

Horizontal Slider class.

See also

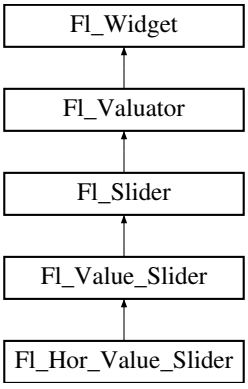
class Fl_Slider.

The documentation for this class was generated from the following files:

- Fl_Hor_Slider.H
- Fl_Slider.cxx

13.64 Fl_Hor_Value_Slider Class Reference

Inheritance diagram for Fl_Hor_Value_Slider:



Public Member Functions

- **Fl_Hor_Value_Slider** (int X, int Y, int W, int H, const char ∗l=0)

Public Member Functions inherited from Fl_Value_Slider

- Fl_Value_Slider (int x, int y, int w, int h, const char ∗l=0)

  Creates a new Fl_Value_Slider widget using the given position, size, and label string.

- int handle (int) override

  Handles the specified event.

- Fl_Color **textcolor** () const

  Gets the color of the text in the value box.

- void **textcolor** (Fl_Color s)

  Sets the color of the text in the value box.

- Fl_Font **textfont** () const

  Gets the typeface of the text in the value box.

- void **textfont** (Fl_Font s)

  Sets the typeface of the text in the value box.

- Fl_Fontsize **textsize** () const

  Gets the size of the text in the value box.

- void **textsize** (Fl_Fontsize s)

  Sets the size of the text in the value box.
- int value_height () const

  Gets the height of the value box in pixels (vertical mode only).
- void value_height (int s)

  Sets the height of the value box in pixels (vertical mode only).
- int value_width () const

  Gets the width of the value box in pixels (horizontal mode only).
- void value_width (int s)

  Sets the width of the value box in pixels (horizontal mode only).

Public Member Functions inherited from Fl_Slider

- void bounds (double a, double b)

  Sets the minimum (a) and maximum (b) values for the valuator widget.
- Fl_Slider (int X, int Y, int W, int H, const char ∗L=0)

  Creates a new Fl_Slider widget using the given position, size, and label string.
- **Fl_Slider** (uchar t, int X, int Y, int W, int H, const char ∗L)

  Creates a new Fl_Slider widget using the given type, position, size, and label string.
- uchar num_ticks () const

  Gets the number of tick marks.
- Scale_Type scale () const

  Gets the scale type.
- void scale (Scale_Type s)

  Sets the scale type.
- int scrollvalue (int pos, int size, int first, int total)

  Sets the size and position of the sliding knob in the box.
- Fl_Boxtype **slider** () const

  Gets the slider box type.
- void **slider** (Fl_Boxtype c)

  Sets the slider box type.
- float **slider_size** () const

  Get the dimensions of the moving piece of slider.
- void slider_size (double v)

  Set the dimensions of the moving piece of slider.
- uchar ticks () const

  Gets the tick mark position.
- void ticks (uchar t, uchar num_ticks=11)

  Sets the tick mark position for Fl_Nice_Slider.

Public Member Functions inherited from Fl_Valuator

- void **bounds** (double a, double b)

  Sets the minimum (a) and maximum (b) values for the valuator widget.
- double **clamp** (double)

  Clamps the passed value to the valuator range.
- virtual int format (char ∗)

  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.
- virtual std::string format_str ()

  C++11 API for Fl_Valuator::format(char∗ buffer).
- double increment (double, int)

Adds n times the step value to the passed value.

- double maximum () const

  Gets the maximum value for the valuator.
- void maximum (double a)

  Sets the maximum value for the valuator.
- double minimum () const

  Gets the minimum value for the valuator.
- void minimum (double a)

  Sets the minimum value for the valuator.
- void precision (int digits)

  Sets the step value to 1.0 / 10^{digits}.
- void range (double a, double b)

  Sets the minimum and maximum values for the valuator.
- double round (double)

  Round the passed value to the nearest step increment.
- double step () const

  Gets or sets the step value.
- void **step** (double a, int b)

  See double Fl_Valuator::step() const
- void **step** (double s)

  See double Fl_Valuator::step() const.
- void **step** (int a)

  See double Fl_Valuator::step() const
- double value () const

  Gets the floating point(double) value.
- int value (double)

  Sets the current value.
- ~**Fl_Valuator** () override

  Destructor is accessible despite protected constructor.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

- virtual class Fl_Gl_Window const ∗ **as_gl_window** () const
- virtual Fl_Group ∗ as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const ∗ **as_group** () const
- virtual Fl_Window ∗ as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ **as_window** () const
- void bind_deimage (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗cb, void ∗p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.

- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

Returns whether the image is managed by the widget.

- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.

- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

Get the spacing between the label and the vertical edge of the widget.

- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Public Types inherited from Fl_Slider

- enum Scale_Type : uchar { LINEAR_SCALE = 0 , LOG_SCALE }

  Bitset for tick mark positions for Fl_Nice_Slider.
- enum Tick_Position : uchar {
  TICKS_NONE = 0x00 , TICKS_BELOW = 0x01 , TICKS_ABOVE = 0x02 , TICKS_LEFT = 0x01 ,
  TICKS_RIGHT = 0x02 }

  Bitset for tick mark positions for Fl_Nice_Slider.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget ∗widget, void ∗data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char ∗t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char ∗, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Value_Slider

- void draw () override

  Draws the widget.

Protected Member Functions inherited from Fl_Slider

- void **draw** (int, int, int, int)
- void draw_ticks (const Fl_Rect &r, int S)

  Draw tick marks.
- int **handle** (int, int, int, int, int)
- double increment_lin_log (double v, int n, int range)

  Adds a sensible step value to the passed value.
- double position_to_value (double pos) const

  Converts a normalized position (0.0 to 1.0) to a value based on the current scale type.
- double value_to_position (double val) const

  Converts a value to a normalized position (0.0 to 1.0) based on the current scale type.

Protected Member Functions inherited from Fl_Valuator

- Fl_Valuator (int X, int Y, int W, int H, const char *L)

  Creates a new Fl_Valuator widget using the given position, size, and label string.
- void **handle_drag** (double newvalue)

  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.
- void **handle_push** ()

  Stores the current value in the previous value.
- void **handle_release** ()

  Called after an FL_WHEN_RELEASE event is received and before the callback.
- int **horizontal** () const

  Tells if the valuator is an FL_HORIZONTAL one.
- double **previous_value** () const

  Gets the previous floating point value before an event changed it.
- void **set_value** (double v)

  Sets the current floating point value.
- double **softclamp** (double)

  Clamps the value, but accepts v if the previous value is not already out of range.
- virtual void value_damage ()

  Asks for partial redraw.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

The documentation for this class was generated from the following files:

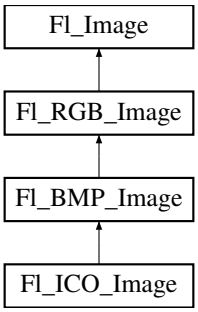
- Fl_Hor_Value_Slider.H
- Fl_Value_Slider.cxx

13.65 Fl_ICO_Image Class Reference

The Fl_ICO_Image class supports loading, caching, and drawing of Windows icon (.ico) files.

`#include <Fl_ICO_Image.H>`

Inheritance diagram for Fl_ICO_Image:



Classes

- struct IconDirEntry

  Windows ICONDIRENTRY structure

Public Member Functions

- Fl_ICO_Image (const char *filename, int id=-1, const unsigned char *data=NULL, const size_t datasize=0)

  Loads the named icon image from the given .ico filename or from memory.
- const IconDirEntry * **icondirentry** () const

  Returns the array of idcount() loaded IconDirEntry structures.
- int **idcount** () const

  Returns the number of icons of various resolutions present in the ICO object.
- ~**Fl_ICO_Image** ()

  Destructor.

Public Member Functions inherited from Fl_BMP_Image

- Fl_BMP_Image (const char *filename)

  This constructor loads the named BMP image from the given BMP filename.
- Fl_BMP_Image (const char *imagename, const unsigned char *data, const long length=-1)

  This constructor loads a BMP image from memory.

Public Member Functions inherited from Fl_RGB_Image

- virtual Fl_SVG_Image * as_svg_image ()

  Returns whether an RGB image is an Fl_SVG_Image or not.
- virtual const Fl_SVG_Image * as_svg_image () const

  Returns whether an RGB image is an Fl_SVG_Image or not.
- int **cache_h** () const
- int **cache_w** () const
- void color_average (Fl_Color c, float i) override

  The color_average() method averages the colors in the image with the provided FLTK color value.
- Fl_Image * **copy** () const
- Fl_Image * copy (int W, int H) const override

  Creates a resized copy of the image.
- void desaturate () override

  The desaturate() method converts an image to grayscale.
- void **draw** (int X, int Y)
- void draw (int X, int Y, int W, int H, int cx=0, int cy=0) override

  Draws the image to the current drawing surface with a bounding box.
- Fl_RGB_Image (const Fl_Pixmap *pxm, Fl_Color bg=FL_GRAY)

The constructor creates a new RGBA image from the specified Fl_Pixmap.

- Fl_RGB_Image (const uchar *bits, int bits_length, int W, int H, int D, int LD)

  The constructor creates a new image from the specified data.
- Fl_RGB_Image (const uchar *bits, int W, int H, int D=3, int LD=0)

  The constructor creates a new image from the specified data.
- void label (Fl_Menu_Item *m) override

  This method is an obsolete way to set the image attribute of a menu item.
- void label (Fl_Widget *w) override

  This method is an obsolete way to set the image attribute of a widget or menu item.
- virtual void normalize ()

  Makes sure the object is fully initialized.
- void uncache () override

  If the image has been cached for display, delete the cache data.
- ~**Fl_RGB_Image** () override

  The destructor frees all memory and server resources that are used by the image.

Public Member Functions inherited from Fl_Image

- virtual class Fl_Shared_Image * as_shared_image ()

  Returns whether an image is an Fl_Shared_Image or not.
- Fl_Image * copy () const

  Creates a copy of the image in the same size.
- int count () const

  Returns the number of data values associated with the image.
- int d () const

  Returns the image depth.
- const char *const * data () const

  Returns a pointer to the current image data array.
- int **data_h** () const

  Returns the height of the image data.
- int **data_w** () const

  Returns the width of the image data.
- void draw (int X, int Y)

  Draws the image to the current drawing surface.
- int fail () const

  Returns a value that is not 0 if there is currently no image available.
- Fl_Image (int W, int H, int D)

  The constructor creates an empty image with the specified width, height, and depth.
- int h () const

  Returns the current image drawing height in FLTK units.
- void inactive ()

  The inactive() method calls color_average(FL_BACKGROUND_COLOR, 0.33f) to produce an image that appears grayed out.
- int ld () const

  Returns the current line data size in bytes.
- virtual void release ()

  Releases an Fl_Image - the same as '`delete this`'.
- virtual void scale (int width, int height, int proportional=1, int can_expand=0)

  Sets the drawing size of the image.
- int w () const

  Returns the current image drawing width in FLTK units.
- virtual ~**Fl_Image** ()

  The destructor is a virtual method that frees all memory used by the image.

Additional Inherited Members

Static Public Member Functions inherited from Fl_RGB_Image

- static size_t max_size ()

  Returns the maximum allowed image size in bytes when creating an Fl_RGB_Image object.
- static void max_size (size_t size)

  Sets the maximum allowed image size in bytes when creating an Fl_RGB_Image object.

Static Public Member Functions inherited from Fl_Image

- static Fl_Labeltype **define_FL_IMAGE_LABEL** ()
- static Fl_RGB_Scaling **RGB_scaling** ()

  Returns the currently used RGB image scaling method.
- static void RGB_scaling (Fl_RGB_Scaling)

  Sets the RGB image scaling method used for copy(int, int).
- static Fl_RGB_Scaling **scaling_algorithm** ()

  Gets what algorithm is used when resizing a source image to draw it.
- static void scaling_algorithm (Fl_RGB_Scaling algorithm)

  Sets what algorithm is used when resizing a source image to draw it.

Public Attributes inherited from Fl_RGB_Image

- int **alloc_array**

  If non-zero, the object's data array is delete[]'d when deleting the object.
- const uchar ∗ array

  Points to the start of the object's data array.

Static Public Attributes inherited from Fl_Image

- static const int **ERR_FILE_ACCESS** = -2
- static const int **ERR_FORMAT** = -3
- static const int **ERR_MEMORY_ACCESS** = -4
- static const int **ERR_NO_IMAGE** = -1
- static bool **register_images_done** = false

  True after fl_register_images() was called, false before.

Protected Member Functions inherited from Fl_BMP_Image

- void **load_bmp_** (class Fl_Image_Reader &rdr, int ico_height=0, int ico_width=0)

Protected Member Functions inherited from Fl_Image

- void **d** (int D)

  Sets the current image depth.
- void data (const char ∗const ∗p, int c)

  Sets the current data pointer and count of pointers in the array.
- void draw_empty (int X, int Y)

  The protected method draw_empty() draws a box with an X in it.
- int draw_scaled (int X, int Y, int W, int H)

  Draw the image to the current drawing surface rescaled to a given width and height.
- void h (int H)

  Sets the height of the image data.
- void ld (int LD)

Sets the current line data size in bytes.

- void w (int W)

  Sets the width of the image data.

Static Protected Member Functions inherited from Fl_Image

- static void **labeltype** (const Fl_Label *lo, int lx, int ly, int lw, int lh, Fl_Align la)
- static void **measure** (const Fl_Label *lo, int &lw, int &lh)

13.65.1 Detailed Description

The Fl_ICO_Image class supports loading, caching, and drawing of Windows icon (.ico) files.

13.65.2 Constructor & Destructor Documentation

13.65.2.1 Fl_ICO_Image()

```
Fl_ICO_Image::Fl_ICO_Image (
            const char * filename,
            int id = -1,
            const unsigned char * data = NULL,
            const size_t datasize = 0 )
```

Loads the named icon image from the given .ico filename or from memory.

Parameters

filename	Name of a .ico file, or of the in-memory image
id	When id is -1 (default), the highest-resolution icon is loaded; when id 0, load the icon with this ID; when id = -2, load all IconDirEntry structures but no image.
data	NULL, or in-memory icon data
datasize	Size in bytes of the `data` byte array (used when `data` is not NULL)

The documentation for this class was generated from the following files:

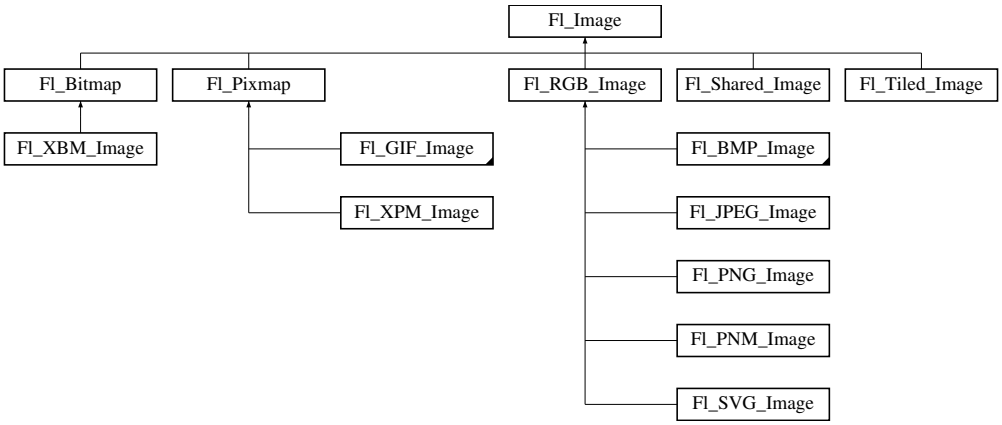
- Fl_ICO_Image.H
- Fl_ICO_Image.cxx

13.66 Fl_Image Class Reference

Base class for image caching, scaling and drawing.

`#include <Fl_Image.H>`

Inheritance diagram for Fl_Image:



Public Member Functions

- virtual class Fl_Shared_Image ∗ as_shared_image ()

  Returns whether an image is an Fl_Shared_Image or not.
- virtual void color_average (Fl_Color c, float i)

  The color_average() method averages the colors in the image with the provided FLTK color value.
- Fl_Image ∗ copy () const

  Creates a copy of the image in the same size.
- virtual Fl_Image ∗ copy (int W, int H) const

  Creates a resized copy of the image.
- int count () const

  Returns the number of data values associated with the image.
- int d () const

  Returns the image depth.
- const char ∗const ∗ data () const

  Returns a pointer to the current image data array.
- int **data_h** () const

  Returns the height of the image data.
- int **data_w** () const

  Returns the width of the image data.
- virtual void desaturate ()

  The desaturate() method converts an image to grayscale.
- void draw (int X, int Y)

  Draws the image to the current drawing surface.
- virtual void draw (int X, int Y, int W, int H, int cx=0, int cy=0)

  Draws the image to the current drawing surface with a bounding box.
- int fail () const

  Returns a value that is not 0 if there is currently no image available.
- Fl_Image (int W, int H, int D)

  The constructor creates an empty image with the specified width, height, and depth.
- int h () const

  Returns the current image drawing height in FLTK units.
- void inactive ()

  The inactive() method calls color_average(FL_BACKGROUND_COLOR, 0.33f) to produce an image that appears grayed out.
- virtual void label (Fl_Menu_Item ∗m)

  This method is an obsolete way to set the image attribute of a menu item.
- virtual void label (Fl_Widget ∗w)

  This method is an obsolete way to set the image attribute of a widget or menu item.
- int ld () const

  Returns the current line data size in bytes.
- virtual void release ()

  Releases an Fl_Image - the same as '`delete this`'.
- virtual void scale (int width, int height, int proportional=1, int can_expand=0)

  Sets the drawing size of the image.
- virtual void uncache ()

  If the image has been cached for display, delete the cache data.
- int w () const

  Returns the current image drawing width in FLTK units.
- virtual ∼**Fl_Image** ()

  The destructor is a virtual method that frees all memory used by the image.

Static Public Member Functions

- static Fl_Labeltype **define_FL_IMAGE_LABEL** ()
- static Fl_RGB_Scaling **RGB_scaling** ()

  Returns the currently used RGB image scaling method.
- static void RGB_scaling (Fl_RGB_Scaling)

  Sets the RGB image scaling method used for copy(int, int).
- static Fl_RGB_Scaling **scaling_algorithm** ()

  Gets what algorithm is used when resizing a source image to draw it.
- static void scaling_algorithm (Fl_RGB_Scaling algorithm)

  Sets what algorithm is used when resizing a source image to draw it.

Static Public Attributes

- static const int **ERR_FILE_ACCESS** = -2
- static const int **ERR_FORMAT** = -3
- static const int **ERR_MEMORY_ACCESS** = -4
- static const int **ERR_NO_IMAGE** = -1
- static bool **register_images_done** = false

  True after fl_register_images() was called, false before.

Protected Member Functions

- void **d** (int D)

  Sets the current image depth.
- void data (const char ∗const ∗p, int c)

  Sets the current data pointer and count of pointers in the array.
- void draw_empty (int X, int Y)

  The protected method draw_empty() draws a box with an X in it.
- int draw_scaled (int X, int Y, int W, int H)

  Draw the image to the current drawing surface rescaled to a given width and height.
- void h (int H)

  Sets the height of the image data.
- void ld (int LD)

  Sets the current line data size in bytes.
- void w (int W)

  Sets the width of the image data.

Static Protected Member Functions

- static void **labeltype** (const Fl_Label ∗lo, int lx, int ly, int lw, int lh, Fl_Align la)
- static void **measure** (const Fl_Label ∗lo, int &lw, int &lh)

Friends

- class **Fl_Graphics_Driver**

13.66.1 Detailed Description

Base class for image caching, scaling and drawing.
Fl_Image is the base class used for caching, scaling and drawing all kinds of images in FLTK. This class keeps track of common image data such as the pixels, colormap, width, height, and depth. Virtual methods are used to provide type-specific image handling.
Each image possesses two (width, height) pairs:

1. The width and height of the raw image data are returned by data_w() and data_h(). These values are set when the image is created and remain unchanged.
2. The width and height of the area filled by the image when it gets drawn are returned by w() and h(). These values are equal to data_w() and data_h() when the image is created and can be changed by the scale() member function.

Since the Fl_Image class does not support image drawing by itself, calling the Fl_Image::draw() method results in a box with an X in it being drawn instead.

13.66.2 Constructor & Destructor Documentation

13.66.2.1 Fl_Image()

```
Fl_Image::Fl_Image (
            int W,
            int H,
            int D )
```

The constructor creates an empty image with the specified width, height, and depth.
The width and height are in pixels. The depth is 0 for bitmaps, 1 for pixmap (colormap) images, and 1 to 4 for color images.

13.66.3 Member Function Documentation

13.66.3.1 as_shared_image()

```
virtual class Fl_Shared_Image * Fl_Image::as_shared_image ( )  [inline], [virtual]
```

Returns whether an image is an Fl_Shared_Image or not.
This virtual method returns a pointer to an Fl_Shared_Image if this object is an instance of Fl_Shared_Image or NULL if not. This can be used to detect if a given Fl_Image object is a shared image, i.e. derived from Fl_Shared_Image.

Since

1.4.0

Reimplemented in Fl_Shared_Image.

13.66.3.2 color_average()

```
void Fl_Image::color_average (
            Fl_Color c,
            float i )  [virtual]
```

The color_average() method averages the colors in the image with the provided FLTK color value.
The first argument specifies the FLTK color to be used.
The second argument specifies the amount of the original image to combine with the color, so a value of 1.0 results in no color blend, and a value of 0.0 results in a constant image of the specified color.
An internal copy is made of the original image data before changes are applied, to avoid modifying the original image data in memory.
Reimplemented in Fl_Anim_GIF_Image, Fl_RGB_Image, Fl_Pixmap, Fl_Shared_Image, Fl_SVG_Image, and Fl_Tiled_Image.

13.66.3.3 copy() [1/2]

```
Fl_Image * Fl_Image::copy ( ) const  [inline]
```

Creates a copy of the image in the same size.
The new image should be released when you are done with it.
This does exactly the same as 'Fl_Image::copy(int W, int H) const' where `W` and `H` are the width and height of the source image, respectively. This applies also to all subclasses of Fl_Image in the FLTK library.
The following two copy() calls are equivalent:

```
Fl_Image *img1 = new Fl_Image(...);
// ...
```

```
Fl_Image *img2 = img1->copy();
Fl_Image *img3 = img1->copy(img1->w(), img1->h())
```

For details see 'Fl_Image::copy(int w, int h) const'.

See also

Fl_Image::release()

Note

Since FLTK 1.4.0 this method is 'const'. If you derive your own class from Fl_Image or any subclass your overridden methods of 'Fl_Image::copy() const' and 'Fl_Image::copy(int, int) const' **must** also be 'const' for inheritance to work properly. This is different than in FLTK 1.3.x and earlier where these methods have not been 'const'.

13.66.3.4 copy() [2/2]

```
Fl_Image * Fl_Image::copy (
            int W,
            int H ) const  [virtual]
```

Creates a resized copy of the image.
It is recommended not to call this member function to reduce the size of an image to the size of the area where this image will be drawn, and to use Fl_Image::scale() instead.
The new image should be released when you are done with it.
Note: since FLTK 1.4.0 you can use Fl_Image::release() for all types of images (i.e. all subclasses of Fl_Image) instead of operator *delete* for Fl_Image's and Fl_Image::release() for Fl_Shared_Image's.
The new image data will be converted to the requested size. RGB images are resized using the algorithm set by Fl_Image::RGB_scaling().
For the new image the following equations are true:

- w() == data_w() == `W`
- h() == data_h() == `H`

Parameters

in	*W,H*	Requested width and height of the new image

Note

The returned image can be safely cast to the same image type as that of the source image provided this type is one of Fl_RGB_Image, Fl_SVG_Image, Fl_Pixmap, Fl_Bitmap, Fl_Tiled_Image, Fl_Anim_GIF_Image and Fl_Shared_Image. Returned objects copied from images of other, derived, image classes belong to the parent class appearing in this list. For example, the copy of an Fl_GIF_Image is an object of class Fl_Pixmap.

Since FLTK 1.4.0 this method is 'const'. If you derive your own class from Fl_Image or any subclass your overridden methods of 'Fl_Image::copy() const' and 'Fl_Image::copy(int, int) const' **must** also be 'const' for inheritance to work properly. This is different than in FLTK 1.3.x and earlier where these methods have not been 'const'.

Reimplemented in Fl_Anim_GIF_Image, Fl_Bitmap, Fl_RGB_Image, Fl_Pixmap, Fl_Shared_Image, Fl_SVG_Image, and Fl_Tiled_Image.

13.66.3.5 count()

```
int Fl_Image::count ( ) const  [inline]
```

Returns the number of data values associated with the image.
The value will be 0 for images with no associated data, 1 for bitmap and color images, and greater than 2 for pixmap images.

See also

data()

13.66.3.6 d()

```
int Fl_Image::d ( ) const  [inline]
```

Returns the image depth.
The return value will be 0 for bitmaps, 1 for pixmaps, and 1 to 4 for color images.

13.66.3.7 data() [1/2]

```
const char *const * Fl_Image::data ( ) const  [inline]
```

Returns a pointer to the current image data array.
There can be 0, 1, or more pointers to actual image data in an image.
Use the count() method to find the size of the data array. You must not dereference the data() pointer if count() equals zero.

Note

data() **may** return NULL.

Example:
Fl_RGB_Image has exactly one pointer which points at the R, G, B [, A] data array of the image. The total size of this array depends on several attributes like data_w(), data_h(), d() and ld() and is basically data_w() ∗ data_h() ∗ d() but there are exceptions if ld() is non-zero: see description of ld(). Since FLTK 1.4.0 w() and h() are no longer significant for the image data size if scale() has been called on the image to set a different display size.
Other image types have different numbers and types of data pointers which are implementation details and not documented here.

See also

count(), w(), h(), data_w(), data_h(), d(), ld()

13.66.3.8 data() [2/2]

```
void Fl_Image::data (
            const char *const * p,
            int c )  [inline], [protected]
```

Sets the current data pointer and count of pointers in the array.
There can be 0, 1, or more pointers to actual image data in an image.

See also

const char∗ const∗ data(), count(), w(), h(), data_w(), data_h(), d(), ld()

13.66.3.9 desaturate()

```
void Fl_Image::desaturate ( )  [virtual]
```

The desaturate() method converts an image to grayscale.
If the image contains an alpha channel (depth = 4), the alpha channel is preserved.
An internal copy is made of the original image data before changes are applied, to avoid modifying the original image data in memory.
Reimplemented in Fl_Anim_GIF_Image, Fl_RGB_Image, Fl_Pixmap, Fl_Shared_Image, Fl_SVG_Image, and Fl_Tiled_Image.

13.66.3.10 draw() [1/2]

```
void Fl_Image::draw (
            int X,
            int Y )  [inline]
```

Draws the image to the current drawing surface.

Parameters

X,Y	specify the upper-lefthand corner of the image.

13.66.3.11 draw() [2/2]

```
void Fl_Image::draw (
            int X,
            int Y,
            int W,
            int H,
            int cx = 0,
            int cy = 0 )  [virtual]
```

Draws the image to the current drawing surface with a bounding box.
Arguments `X,Y,W,H` specify a bounding box for the image, with the origin (upper-left corner) of the image offset by the `cx` and `cy` arguments.
Another way to see what part of the image gets drawn and where, is to consider this alternative writing producing the same output:

```
fl_push_clip(X,Y,W,H);
this->draw(X - cx, Y - cy);
fl_pop_clip();
```

Repeated calls to this member function with the same image but varying `W, H, cx`, or `cy` arguments can be more efficiently processed using the above alternative writing.
Reimplemented in Fl_Anim_GIF_Image, Fl_Bitmap, Fl_RGB_Image, Fl_Pixmap, Fl_Shared_Image, Fl_SVG_Image, and Fl_Tiled_Image.

13.66.3.12 draw_empty()

```
void Fl_Image::draw_empty (
            int X,
            int Y )  [protected]
```

The protected method draw_empty() draws a box with an X in it.
It can be used to draw any image that lacks image data.

13.66.3.13 draw_scaled()

```
int Fl_Image::draw_scaled (
            int X,
            int Y,
            int W,
            int H )  [protected]
```

Draw the image to the current drawing surface rescaled to a given width and height.
Intended for internal use by the FLTK library.

Parameters

X,Y	position of the image's top-left
W,H	width and height for the drawn image

Returns

1

Deprecated Only for API compatibility with FLTK 1.3.4.

13.66.3.14 fail()

```
int Fl_Image::fail ( ) const
```

Returns a value that is not 0 if there is currently no image available.
Example use:

```
// [..]
  Fl_Box box(X, Y, W, H);
  Fl_JPEG_Image jpg("/tmp/foo.jpg");
  switch (jpg.fail()) {
    case Fl_Image::ERR_NO_IMAGE:
    case Fl_Image::ERR_FILE_ACCESS:
      fl_alert("/tmp/foo.jpg: %s", strerror(errno));    // shows actual os error to user
      exit(1);
    case Fl_Image::ERR_FORMAT:
      fl_alert("/tmp/foo.jpg: couldn't decode image");
      exit(1);
  }
  box.image(jpg);
```

Returns

Image load failure if non-zero

Return values

0	the image was loaded successfully
ERR_NO_IMAGE	no image was found
ERR_FILE_ACCESS	there was a file access related error (errno should be set)
ERR_FORMAT	image decoding failed
ERR_MEMORY_ACCESS	image decoder tried to access memory outside of given memory block

13.66.3.15 h() [1/2]

```
int Fl_Image::h ( ) const  [inline]
```

Returns the current image drawing height in FLTK units.
The values of h() and data_h() are identical unless scale() has been called after which they may differ.

13.66.3.16 h() [2/2]

```
void Fl_Image::h (
            int H )  [inline], [protected]
```

Sets the height of the image data.
This protected function sets both image heights: the height of the image data returned by data_h() and the image drawing height in FLTK units returned by h().

13.66.3.17 inactive()

```
void Fl_Image::inactive ( )  [inline]
```

The inactive() method calls color_average(FL_BACKGROUND_COLOR, 0.33f) to produce an image that appears grayed out.
An internal copy is made of the original image before changes are applied, to avoid modifying the original image.

Note

The RGB color of FL_BACKGROUND_COLOR may change when the connection to the display is made. See fl_open_display().

13.66.3.18 label() [1/2]

```
void Fl_Image::label (
            Fl_Menu_Item * m )  [virtual]
```

This method is an obsolete way to set the image attribute of a menu item.

Deprecated Please use Fl_Menu_Item::image() instead.

Reimplemented in Fl_Bitmap, Fl_RGB_Image, and Fl_Pixmap.

13.66.3.19 label() [2/2]

```
void Fl_Image::label (
            Fl_Widget * widget )  [virtual]
```

This method is an obsolete way to set the image attribute of a widget or menu item.

Deprecated Please use Fl_Widget::image() or Fl_Widget::deimage() instead.

Reimplemented in Fl_Bitmap, Fl_RGB_Image, and Fl_Pixmap.

13.66.3.20 ld() [1/2]

```
int Fl_Image::ld ( ) const  [inline]
```

Returns the current line data size in bytes.

See also

ld(int)

13.66.3.21 ld() [2/2]

```
void Fl_Image::ld (
            int LD )  [inline], [protected]
```

Sets the current line data size in bytes.
Color images may contain extra data (padding) that is included after every line of color image data and is normally not present.
If `LD` is zero, then line data size is assumed to be data_w() * d() bytes.
If `LD` is non-zero, then it must be positive and larger than data_w() * d() to account for the extra data per line.

13.66.3.22 release()

```
virtual void Fl_Image::release ( )  [inline], [virtual]
```

Releases an Fl_Image - the same as '`delete this`'.
This virtual method is for almost all image classes the same as calling
`delete image;`
where image is an `Fl_Image *` pointer.
However, for subclass Fl_Shared_Image and its subclasses this virtual method is reimplemented and maintains shared images.
This virtual method makes it possible to `destroy` all image types in the same way by calling
`image->release();`
Reasoning: If you have an 'Fl_Image *' base class pointer and don't know if the object is one of the class Fl_Shared_Image or any other subclass of Fl_Image (for instance Fl_RGB_Image) then you can't just use operator delete since this is not appropriate for Fl_Shared_Image objects.
The virtual method release() handles this properly.

Since

1.4.0 in the base class Fl_Image and virtual in Fl_Shared_Image

Reimplemented in Fl_Shared_Image.

13.66.3.23 RGB_scaling()

```
void Fl_Image::RGB_scaling (
            Fl_RGB_Scaling method )  [static]
```

Sets the RGB image scaling method used for copy(int, int).
Applies to all RGB images, defaults to FL_RGB_SCALING_NEAREST.

13.66.3.24 scale()

```
void Fl_Image::scale (
            int width,
            int height,
            int proportional = 1,
            int can_expand = 0 )  [virtual]
```

Sets the drawing size of the image.

This function controls the values returned by member functions w() and h() which in turn control how the image is drawn: the full image data (whose size is given by data_w() and data_h()) are drawn scaled to an area of the drawing surface sized at w() x h() FLTK units. This can make a difference if the drawing surface has more than 1 pixel per FLTK unit because the image can be drawn at the full resolution of the drawing surface. Examples of such drawing surfaces: HiDPI displays, laser printers, PostScript files, PDF printers.

Parameters

width,height	maximum values, in FLTK units, that w() and h() should return
proportional	if not null, keep the values returned by w() and h() proportional to data_w() and data_h()
can_expand	if null, the values returned by w() and h() will not be larger than data_w() and data_h(), respectively

Note

This function generally changes the values returned by the w() and h() member functions. In contrast, the values returned by data_w() and data_h() remain unchanged.

If the processed image is an Fl_SVG_Image, setting Fl_SVG_Image::proportional to `false` is required if `proportional` here is null.

Version

1.4 (1.3.4 and FL_ABI_VERSION for Fl_Shared_Image only)

Example code: scale an image to fit in a box

```
Fl_Box *b = ...  // a box
Fl_Image *img = new Fl_PNG_Image("/path/to/picture.png"); // read a picture file
// set the drawing size of the image to the size of the box keeping its aspect ratio
img->scale(b->w(), b->h());
b->image(img); // use the image as the box image
```

Reimplemented in Fl_SVG_Image.

13.66.3.25 scaling_algorithm()

```
static void Fl_Image::scaling_algorithm (
            Fl_RGB_Scaling algorithm )  [inline], [static]
```

Sets what algorithm is used when resizing a source image to draw it.

The default algorithm is FL_RGB_SCALING_BILINEAR. Drawing an Fl_Image is sometimes performed by first resizing the source image and then drawing the resized copy. This occurs, e.g., when drawing to screen under X11 without Xrender support after having called scale(). This function controls what method is used when the image to be resized is an Fl_RGB_Image.

Version

1.4

13.66.3.26 uncache()

```
void Fl_Image::uncache ( )  [virtual]
```

If the image has been cached for display, delete the cache data.

This allows you to change the data used for the image and then redraw it without recreating an image object.

Reimplemented in Fl_Anim_GIF_Image, Fl_Bitmap, Fl_RGB_Image, Fl_Pixmap, and Fl_Shared_Image.

13.66.3.27 w() [1/2]

```
int Fl_Image::w ( ) const  [inline]
```

Returns the current image drawing width in FLTK units.
The values of w() and data_w() are identical unless scale() has been called after which they may differ.

13.66.3.28 w() [2/2]

```
void Fl_Image::w (
            int W )  [inline], [protected]
```

Sets the width of the image data.
This protected function sets both image widths: the width of the image data returned by data_w() and the image drawing width in FLTK units returned by w().
The documentation for this class was generated from the following files:

- Fl_Image.H
- Fl_Image.cxx

13.67 Fl_Image_Reader Class Reference

Public Member Functions

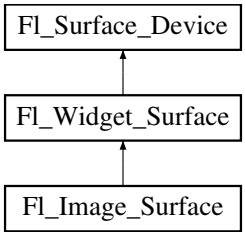
- size_t **data_size** () const
- const unsigned char ∗ **data_start** () const
- int **error** () const
- char **is_data** () const
- char **is_file** () const
- const char ∗ **name** () const
- int **open** (const char ∗filename)
- int **open** (const char ∗imagename, const unsigned char ∗data)
- int **open** (const char ∗imagename, const unsigned char ∗data, const size_t datasize)
- unsigned char **read_byte** ()
- unsigned int **read_dword** ()
- int **read_long** ()
- unsigned short **read_word** ()
- void **seek** (unsigned int n)
- void **skip** (unsigned int n)
- long **tell** () const

The documentation for this class was generated from the following files:

- Fl_Image_Reader.h
- Fl_Image_Reader.cxx

13.68 Fl_Image_Surface Class Reference

Directs all graphics requests to an Fl_Image.
`#include <Fl_Image_Surface.H>`
Inheritance diagram for Fl_Image_Surface:



Public Member Functions

- Fl_Image_Surface (int w, int h, int high_res=0, Fl_Offscreen off=0)

  Constructor with optional high resolution.
- Fl_Shared_Image ∗ highres_image ()

  Returns a possibly high resolution image made of all drawings sent to the Fl_Image_Surface object.
- Fl_RGB_Image ∗ image ()

  Returns a depth-3 image made of all drawings sent to the Fl_Image_Surface object.
- bool is_current () override

  Is this surface the current drawing surface?
- void mask (const Fl_RGB_Image ∗)

  Defines a mask applied to drawings made after use of this function.
- Fl_Offscreen offscreen ()

  Returns the Fl_Offscreen object associated to the image surface.
- void origin (int ∗x, int ∗y) override

  Computes the coordinates of the current origin of graphics functions.
- void origin (int x, int y) override

  Sets the position of the origin of graphics in the drawable part of the drawing surface.
- int printable_rect (int ∗w, int ∗h) override

  Computes the width and height of the drawable area of the drawing surface.
- void rescale ()

  Adapts the Fl_Image_Surface object to the new value of the GUI scale factor.
- void set_current () override

  Make this surface the current drawing surface.
- ∼**Fl_Image_Surface** ()

  The destructor.

Public Member Functions inherited from Fl_Widget_Surface

- void draw (Fl_Widget ∗widget, int delta_x=0, int delta_y=0)

  Draws the widget on the drawing surface.
- void draw_decorated_window (Fl_Window ∗win, int x_offset=0, int y_offset=0)

  Draws a window with its title bar and frame if any.
- void print_window_part (Fl_Window ∗win, int x, int y, int w, int h, int delta_x=0, int delta_y=0)

  Draws a rectangular part of an on-screen window.

Public Member Functions inherited from Fl_Surface_Device

- Fl_Graphics_Driver ∗ **driver** ()

  Returns the graphics driver of this drawing surface.
- virtual ∼**Fl_Surface_Device** ()

  The destructor.

Protected Member Functions

- void translate (int x, int y) override

  Translates the current graphics origin accounting for the current rotation.
- void untranslate () override

  Undoes the effect of a previous translate() call.

Protected Member Functions inherited from Fl_Widget_Surface

- Fl_Widget_Surface (Fl_Graphics_Driver ∗d)

  The constructor.

Protected Member Functions inherited from Fl_Surface_Device

- void **driver** (Fl_Graphics_Driver *graphics_driver)

  Sets the graphics driver of this drawing surface.
- virtual void end_current ()

  FLTK calls this each time a surface ceases to be the current drawing surface.
- **Fl_Surface_Device** (Fl_Graphics_Driver *graphics_driver)

  Constructor that sets the graphics driver to use for the created surface.

Friends

- class **Fl_Graphics_Driver**

Additional Inherited Members

Static Public Member Functions inherited from Fl_Surface_Device

- static Fl_Surface_Device * pop_current ()

  Removes the top element from the current drawing surface stack, and makes the new top element current.
- static void push_current (Fl_Surface_Device *new_current)

  Pushes `new_current` on top of the stack of current drawing surfaces, and makes it current.
- static Fl_Surface_Device * surface ()

  The current drawing surface.

Protected Attributes inherited from Fl_Widget_Surface

- int **x_offset**

  horizontal offset to the origin of graphics coordinates
- int **y_offset**

  vertical offset to the origin of graphics coordinates

13.68.1 Detailed Description

Directs all graphics requests to an Fl_Image.
After creation of an Fl_Image_Surface object, make it the current drawing surface calling Fl_Surface_Device::push_current(), and all subsequent graphics requests will be recorded in the image. It's possible to draw widgets (using Fl_Image_Surface::draw()) or to use any of the Drawing functions or the Color & Font functions. Finally, call image() on the object to obtain a newly allocated Fl_RGB_Image object. Fl_Gl_Window objects can be drawn in the image as well.
Usage example:

```
// this is the widget that you want to draw into an image
Fl_Widget *g = ...;

// create an Fl_Image_Surface object
Fl_Image_Surface *image_surface = new Fl_Image_Surface(g->w(), g->h());

// direct all further graphics requests to the image
Fl_Surface_Device::push_current(image_surface);

// draw a white background
fl_color(FL_WHITE);
fl_rectf(0, 0, g->w(), g->h());

// draw the g widget in the image
image_surface->draw(g);

// get the resulting image
Fl_RGB_Image* image = image_surface->image();

// direct graphics requests back to their previous destination
Fl_Surface_Device::pop_current();

// delete the image_surface object, but not the image itself
delete image_surface;
```

13.68.2 Constructor & Destructor Documentation

13.68.2.1 Fl_Image_Surface()

```
Fl_Image_Surface::Fl_Image_Surface (
            int w,
            int h,
            int high_res = 0,
            Fl_Offscreen off = 0 )
```

Constructor with optional high resolution.

Parameters

w,h	Width and height of the resulting image. The value of the `high_res` parameter controls whether `w` and `h` are interpreted as pixels or FLTK units.
high_res	If zero, the created image surface is sized at `w` x `h` pixels. If non-zero, the pixel size of the created image surface depends on the value of the display scale factor (see Fl::screen_scale(int)): the resulting image has the same number of pixels as an area of the display of size `w` x `h` expressed in FLTK units.
off	If not null, the image surface is constructed around a pre-existing Fl_Offscreen. The caller is responsible for both construction and destruction of this Fl_Offscreen object. Is mostly intended for internal use by FLTK.

Version

1.3.4 (1.3.3 without the `highres` parameter)

13.68.3 Member Function Documentation

13.68.3.1 highres_image()

```
Fl_Shared_Image * Fl_Image_Surface::highres_image ( )
```

Returns a possibly high resolution image made of all drawings sent to the Fl_Image_Surface object.
The Fl_Image_Surface object should have been constructed with Fl_Image_Surface(W, H, 1). The returned Fl_Shared_Image object is scaled to a size of WxH FLTK units and may have a pixel size larger than these values. The returned object should be deallocated with Fl_Shared_Image::release() after use.

Deprecated Use image() instead.

Version

1.4 (1.3.4 for MacOS platform only)

13.68.3.2 image()

```
Fl_RGB_Image * Fl_Image_Surface::image ( )
```

Returns a depth-3 image made of all drawings sent to the Fl_Image_Surface object.
The returned object contains its own copy of the RGB data; the caller is responsible for deleting it.

See also

Fl_Image_Surface::mask(Fl_RGB_Image∗)

13.68.3.3 is_current()

```
bool Fl_Image_Surface::is_current ( ) [override], [virtual]
```

Is this surface the current drawing surface?
Reimplemented from Fl_Surface_Device.

13.68.3.4 mask()

```
void Fl_Image_Surface::mask (
            const Fl_RGB_Image * mask )
```

Defines a mask applied to drawings made after use of this function.

The mask is an Fl_RGB_Image made of a white scene drawn on a solid black background; the drawable part of the image surface is reduced to the white areas of the mask after this member function gets called. If necessary, the `mask` image is internally replaced by a copy resized to the surface's pixel size. Overall, the image returned by Fl_Image_Surface::image() contains all drawings made until the mask() method assigned a mask, at which point subsequent drawing operations to the image surface were passed through the white areas of the mask. On some platforms, shades of gray in the mask image control the blending of foreground and background pixels; mask pixels closer in color to white produce image pixels closer to the image surface pixel, those closer to black produce image pixels closer to what the image surface pixel was before the call to mask().

The mask is easily constructed using an Fl_Image_Surface object, drawing white areas on a black background there, and calling Fl_Image_Surface::image().

Parameters

mask	A depth-3 image determining the drawable areas of the image surface. The `mask` object is not used after return from this member function.

Note

- The image surface must not be the current drawing surface when this function gets called.
- The mask can have any size but is best when it has the size of the image surface.
- It's possible to use several masks in succession on the same image surface provided member function Fl_Image_Surface::image() is called between successive calls to Fl_Image_Surface::mask(const Fl_RGB_Image*).

Example of procedure to construct a masked image:

```
int W = ..., H = ...; // width and height of the image under construction
Fl_Image_Surface *surf = new Fl_Image_Surface(W, H, 1);
// first, construct the mask
Fl_Surface_Device::push_current(surf);
fl_color(FL_BLACK); // draw a black background
fl_rectf(0, 0, W, H);
fl_color(FL_WHITE); // next, draw in white what the mask should not filter out
fl_pie(0, 0, W, H, 0, 360); // here, an ellipse with axes lengths WxH
Fl_RGB_Image *mask = surf->image(); // get the mask
// second, draw the image background
fl_color(FL_YELLOW); // here, draw a yellow background
fl_rectf(0, 0, W, H);
// third, apply the mask
Fl_Surface_Device::pop_current();
surf->mask(mask);
delete mask; // the mask image can be safely deleted at this point
Fl_Surface_Device::push_current(surf);
// fourth, draw the image foreground, part of which will be filtered out by the mask
surf->draw(widget, 0, 0); // here the foreground is a drawn widget
// fifth, get the final result, masked_image, as a depth-3 Fl_RGB_Image
Fl_RGB_Image *masked_image = surf->image();
// Only the part of the foreground, here a drawn widget, that has not been
// filtered out by the mask, here the white ellipse, is in masked_image;
// the background, here solid yellow, shows up in the remaining areas of masked_image.
Fl_Surface_Device::pop_current();
delete surf;
```

Since

1.4.0

13.68.3.5 offscreen()

```
Fl_Offscreen Fl_Image_Surface::offscreen ( )
```

Returns the Fl_Offscreen object associated to the image surface.

The returned Fl_Offscreen object is deleted when the Fl_Image_Surface object is deleted, unless the Fl_Image_Surface was constructed with non-null Fl_Offscreen argument.

13.68.3.6 origin() [1/2]

```
void Fl_Image_Surface::origin (
            int * x,
            int * y )  [override], [virtual]
```

Computes the coordinates of the current origin of graphics functions.

Parameters

out	*x,y*	If non-null, ∗x and ∗y are set to the horizontal and vertical coordinates of the graphics origin.

Reimplemented from Fl_Widget_Surface.

13.68.3.7 origin() [2/2]

```
void Fl_Image_Surface::origin (
            int x,
            int y )  [override], [virtual]
```

Sets the position of the origin of graphics in the drawable part of the drawing surface.
Arguments should be expressed relatively to the result of a previous printable_rect() call. That is, `printable_rect(&w, &h); origin(w/2, 0);` sets the graphics origin at the top center of the drawable area. Successive origin() calls don't combine their effects. Origin() calls are not affected by rotate() calls (for classes derived from Fl_Paged_Device).

Parameters

in	*x,y*	Horizontal and vertical positions in the drawing surface of the desired origin of graphics.

Reimplemented from Fl_Widget_Surface.

13.68.3.8 printable_rect()

```
int Fl_Image_Surface::printable_rect (
            int * w,
            int * h )  [override], [virtual]
```

Computes the width and height of the drawable area of the drawing surface.
Values are in the same unit as that used by FLTK drawing functions and are unchanged by calls to origin(). If the object is derived from class Fl_Paged_Device, values account for the user-selected paper type and print orientation and are changed by scale() calls.

Returns

0 if OK, non-zero if any error

Reimplemented from Fl_Widget_Surface.

13.68.3.9 rescale()

```
void Fl_Image_Surface::rescale ( )
```

Adapts the Fl_Image_Surface object to the new value of the GUI scale factor.
The Fl_Image_Surface object must not be the current drawing surface. This function is useful only for an object constructed with non-zero `high_res` parameter.

Version

1.4

13.68.3.10 set_current()

```
void Fl_Image_Surface::set_current (
            void  )  [override], [virtual]
```

Make this surface the current drawing surface.
This surface will receive all future graphics requests.
Since FLTK 1.4.0 the preferred API to change the current drawing surface is Fl_Surface_Device::push_current( ) / Fl_Surface_Device::pop_current().

Note

It is recommended to use this function only as follows :

- The current drawing surface is the display;
- make current another surface, e.g., an Fl_Printer or an Fl_Image_Surface object, calling set_current() on this object;
- draw to that surface;
- make the display current again with Fl_Display_Device::display_device()->set_current();
  don't do any other call to set_current() before this one.

Other scenarios of drawing surface changes should be performed via Fl_Surface_Device::push_current() and Fl_Surface_Device::pop_current().
Reimplemented from Fl_Surface_Device.

13.68.3.11 translate()

```
void Fl_Image_Surface::translate (
            int x,
            int y )  [override], [protected], [virtual]
```

Translates the current graphics origin accounting for the current rotation.
Each translate() call must be matched by an untranslate() call. Successive translate() calls add up their effects.
Reimplemented from Fl_Widget_Surface.

13.68.3.12 untranslate()

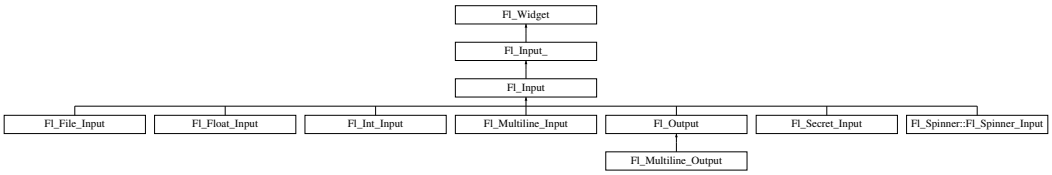
```
void Fl_Image_Surface::untranslate (
            void  )  [override], [protected], [virtual]
```

Undoes the effect of a previous translate() call.
Reimplemented from Fl_Widget_Surface.
The documentation for this class was generated from the following files:

- Fl_Image_Surface.H
- Fl_Image_Surface.cxx

13.69 Fl_Input Class Reference

This is the FLTK text input widget.
`#include <Fl_Input.H>`
Inheritance diagram for Fl_Input:



Public Member Functions

- Fl_Input (int, int, int, int, const char *=0)

  Creates a new Fl_Input widget using the given position, size, and label string.
- int handle (int) override

  Handles the specified event.

Public Member Functions inherited from Fl_Input_

- int append (const char *t, int l=0, char keep_selection=0)

  Append text at the end.
- bool can_redo () const

  Check if there is a redo action available.
- bool can_undo () const

  Check if the last operation can be undone.
- int copy (int clipboard)

  Put the current selection into the clipboard.
- int copy_cuts ()

  Copies the yank buffer to the clipboard.
- Fl_Color cursor_color () const

  Gets the color of the cursor.
- void cursor_color (Fl_Color n)

  Sets the color of the cursor.
- int cut ()

  Deletes the current selection.
- int cut (int a, int b)

  Deletes all characters between index `a` and `b`.
- int cut (int n)

  Deletes the next `n` bytes rounded to characters before or after the cursor.
- double dvalue () const

  Returns the widget text interpreted as a floating point number.
- Fl_Input_ (int, int, int, int, const char *=0)

  Creates a new Fl_Input_ widget.
- unsigned int index (int i) const

  Returns the character at index `i`.
- int input_type () const

  Gets the input field type.
- void input_type (int t)

  Sets the input field type.
- int insert (const char *t, int l=0)

  Inserts text at the cursor position.
- int insert_position () const

  Gets the position of the text cursor.
- int insert_position (int p)

  Sets the cursor position and mark.
- int insert_position (int p, int m)

  Sets the index for the cursor and mark.
- int ivalue () const

  Returns the widget text interpreted as a signed integer.
- int mark () const

  Gets the current selection mark.
- int mark (int m)

  Sets the current selection mark.
- int maximum_size () const

  Gets the maximum length of the input field in characters.
- void maximum_size (int m)

  Sets the maximum length of the input field in characters.
- const char * placeholder () const

Get the placeholder text.

- void placeholder (const char *text)

  Set the placeholder text.
- int position () const
- int position (int p)
- int position (int p, int m)
- int readonly () const

  Gets the read-only state of the input field.
- void readonly (int b)

  Sets the read-only state of the input field.
- int redo ()

  Redo previous undo operation.
- int replace (int b, int e, const char *text, int ilen=0)

  Deletes text from `b` to `e` and inserts the new string `text`.
- void resize (int, int, int, int) override

  Changes the size of the widget.
- int shortcut () const

  Return the shortcut key associated with this widget.
- void shortcut (int s)

  Sets the shortcut key associated with this widget.
- int size () const

  Returns the number of bytes in value().
- void size (int W, int H)

  Sets the width and height of this widget.
- int static_value (const char *)

  Changes the widget text.
- int static_value (const char *, int)

  Changes the widget text.
- int tab_nav () const

  Gets whether the Tab key causes focus navigation in multiline input fields or not.
- void tab_nav (int val)

  Sets whether the Tab key does focus navigation, or inserts tab characters into Fl_Multiline_Input.
- Fl_Color textcolor () const

  Gets the color of the text in the input field.
- void textcolor (Fl_Color n)

  Sets the color of the text in the input field.
- Fl_Font textfont () const

  Gets the font of the text in the input field.
- void textfont (Fl_Font s)

  Sets the font of the text in the input field.
- Fl_Fontsize textsize () const

  Gets the size of the text in the input field.
- void textsize (Fl_Fontsize s)

  Sets the size of the text in the input field.
- int undo ()

  Undoes previous changes to the text buffer.
- const char * value () const

  Returns the text displayed in the widget.
- int value (const char *)

  Changes the widget text.
- int value (const char *, int)

Changes the widget text.

- int value (double value)

  Changes the widget text to a floating point number ("%g").
- int value (int value)

  Changes the widget text to a signed integer number.
- int wrap () const

  Gets the word wrapping state of the input field.
- void wrap (int b)

  Sets the word wrapping state of the input field.
- ~Fl_Input_ ()

  Destroys the widget.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.

- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

Gets the image that is used as part of the widget label when in the inactive state.

- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.

- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

Makes a widget visible.

- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.

- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Static Public Attributes

- static const char * **copy_menu_text** = "Copy"

  [this text may be customized at run-time]
- static const char * **cut_menu_text** = "Cut"

  [this text may be customized at run-time]
- static const char * **paste_menu_text** = "Paste"

  [this text may be customized at run-time]

Protected Member Functions

- void draw () override

  Draws the widget.
- int handle_key ()

  Handles a keystroke.
- int handle_rmb ()

  Handle right mouse button down events.

Protected Member Functions inherited from Fl_Input_

- int apply_undo ()

  Apply the current undo/redo operation.
- void drawtext (int, int, int, int)

  Draws the text in the passed bounding box.
- void drawtext (int, int, int, int, bool draw_active)

  Draws the text in the passed bounding box.
- void handle_mouse (int, int, int, int, int keepmark=0)

  Handles mouse clicks and mouse moves.
- int handletext (int e, int, int, int, int)

  Handles all kinds of text field related events.
- int line_end (int i) const

  Finds the end of a line.
- int line_start (int i) const

  Finds the start of a line.
- int **linesPerPage** ()
- void **maybe_do_callback** (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
- int up_down_position (int, int keepmark=0)

  Moves the cursor to the column given by `up_down_pos`.
- int word_end (int i) const

  Finds the end of a word.
- int word_start (int i) const

  Finds the start of a word.
- int **xscroll** () const
- int **yscroll** () const
- void **yscroll** (int yOffset)

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Friends

- class **Fl_Cocoa_Screen_Driver**
- class **Fl_Screen_Driver**

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 , AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.69.1 Detailed Description

This is the FLTK text input widget.
It displays a single line of text and lets the user edit it. Normally it is drawn with an inset box and a white background. The text may contain any characters, and will correctly display any UTF text, using ^X notation for unprintable control characters. It assumes the font can draw any characters of the used scripts, which is true for standard fonts under Windows and Mac OS X. Characters can be input using the keyboard or the character palette/map. Character composition is done using dead keys and/or a compose key as defined by the operating system.

Table 13.286 Keyboard and mouse bindings.

Mouse button 1	Moves the cursor to this point. Drag selects characters. Double click selects words. Triple click selects all line. Shift+click extends the selection. When you select text it is automatically copied to the selection buffer.
Mouse button 2	Insert the selection buffer at the point clicked. You can also select a region and replace it with the selection buffer by selecting the region with mouse button 2.
Mouse button 3	Currently acts like button 1.
Backspace	Deletes one character to the left, or deletes the selected region.
Delete	Deletes one character to the right, or deletes the selected region. Combine with Shift for equivalent of ^X (copy+cut).
Enter	May cause the callback, see when().

Table 13.287 Platform specific keyboard bindings.

Windows/Linux	Mac	Function
^A	**Command-A**	**Selects all text in the widget.**
^C	**Command-C**	**Copy the current selection to the clipboard.**
^I	**^I**	**Insert a tab.**
^J	**^J**	**Insert a Line Feed.** (Similar to literal 'Enter' character)
^L	**^L**	**Insert a Form Feed.**
^M	**^M**	**Insert a Carriage Return.**

^V, Shift-Insert	**Command-V**	**Paste the clipboard.** (Macs keyboards don't have "Insert" keys, but if they did, Shift-Insert would work)
^X, Shift-Delete	**Command-X, Shift-Delete**	**Cut.** Copy the selection to the clipboard and delete it. (If there's no selection, Shift-Delete acts like Delete)
^Z	**Command-Z**	**Undo.** This is a single-level undo mechanism, but all adjacent deletions and insertions are concatenated into a single "undo". Often this will undo a lot more than you expected.
Shift-^Z	**Shift-Command-Z**	**Redo.** Currently same behavior as ^Z. Reserved for future multilevel undo/redo.
Arrow Keys	**Arrow Keys**	**Standard cursor movement.** Can be combined with Shift to extend selection.
Home	**Command-Up, Command-Left**	**Move to start of line.** Can be combined with Shift to extend selection.
End	**Command-Down, Command-Right**	**Move to end of line.** Can be combined with Shift to extend selection.
Ctrl-Home	**Command-Up, Command-PgUp, Ctrl-Left**	**Move to top of document/field.** In single line input, moves to start of line. In multiline input, moves to start of top line. Can be combined with Shift to extend selection.
Ctrl-End	**Command-End, Command-PgDn, Ctrl-Right**	**Move to bottom of document/field.** In single line input, moves to end of line. In multiline input, moves to end of last line. Can be combined with Shift to extend selection.
Ctrl-Left	**Alt-Left**	**Word left.** Can be combined with Shift to extend selection.
Ctrl-Right	**Alt-Right**	**Word right.** Can be combined with Shift to extend selection.
Ctrl-Backspace	**Alt-Backspace**	**Delete word left.**
Ctrl-Delete	**Alt-Delete**	**Delete word right.**

13.69.2 Constructor & Destructor Documentation

13.69.2.1 Fl_Input()

```
Fl_Input::Fl_Input (
            int X,
            int Y,
            int W,
            int H,
            const char * l = 0 )
```

Creates a new Fl_Input widget using the given position, size, and label string.
The default boxtype is FL_DOWN_BOX.

13.69.3 Member Function Documentation

13.69.3.1 draw()

```
void Fl_Input::draw ( ) [override], [protected], [virtual]
```

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();                 // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

13.69.3.2 handle()

```
int Fl_Input::handle (
            int event ) [override], [virtual]
```

Handles the specified event.

You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.

When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.

Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.

One exception to the rule in the previous paragraph is if you really want to *override* the behavior of the base class. This requires knowledge of the details of the inherited class.

In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to *filter* event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you *consumed* the event by returning 1 even if you didn't *do* anything with it.

Parameters

in	*event*	the kind of event received

Return values

0	if the event was not used or understood
1	if the event was used and can be deleted

See also

Fl_Event

Reimplemented from Fl_Widget.

Reimplemented in Fl_Spinner::Fl_Spinner_Input, and Fl_Secret_Input.

13.69.3.3 handle_key()

```
int Fl_Input::handle_key ( ) [protected]
```

Handles a keystroke.

This `protected` method handles a keystroke in an Fl_Input or derived class. It handles compose key sequences and can also be used e.g. in Fl_Multiline_Input, Fl_Float_Input and several more derived classes.

The method first checks in Fl::compose if the keystroke is a text entry or a control key. If it is text, the method inserts the composed characters into the input field, taking into account the input type (e.g., numeric fields).

If the keystroke is a control key as determined by Fl::compose, the method handles key combinations for Insert, Enter, and Tab depending on the widget's input_type().

The method then checks for Ctrl key combinations, such as Ctrl-A, Ctrl-C, Ctrl-V, Ctrl-X, and Ctrl-Z, which are commonly used for select all, copy, paste, cut, and undo operations.

Finally, the method checks for ASCII control characters, such as Ctrl-H, Ctrl-I, Ctrl-J, Ctrl-L, and Ctrl-M, which can be used to insert literal control characters into the input field.
If none of the above cases match, the method returns 0, indicating that the keystroke was not handled.

Returns

1 if the keystroke is handled by us, 0 if not.

13.69.3.4 handle_rmb()

`int Fl_Input::handle_rmb ( ) [protected]`
Handle right mouse button down events.

Returns

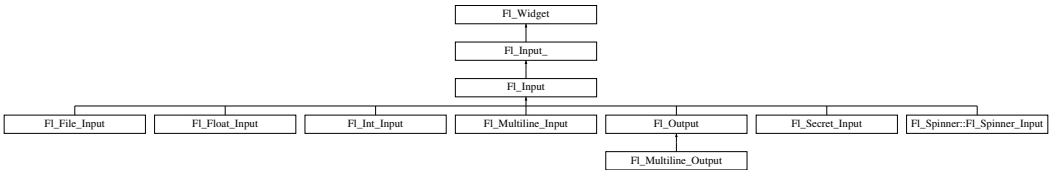
1

The documentation for this class was generated from the following files:

- Fl_Input.H
- Fl_Input.cxx

13.70 Fl_Input_ Class Reference

This class provides a low-overhead text input field.
`#include <Fl_Input_.H>`
Inheritance diagram for Fl_Input_:



Public Member Functions

- int append (const char *t, int l=0, char keep_selection=0)

  Append text at the end.
- bool can_redo () const

  Check if there is a redo action available.
- bool can_undo () const

  Check if the last operation can be undone.
- int copy (int clipboard)

  Put the current selection into the clipboard.
- int copy_cuts ()

  Copies the yank buffer to the clipboard.
- Fl_Color cursor_color () const

  Gets the color of the cursor.
- void cursor_color (Fl_Color n)

  Sets the color of the cursor.
- int cut ()

  Deletes the current selection.
- int cut (int a, int b)

  Deletes all characters between index `a` and `b`.
- int cut (int n)

Deletes the next `n` bytes rounded to characters before or after the cursor.

- double dvalue () const

  Returns the widget text interpreted as a floating point number.
- Fl_Input_ (int, int, int, int, const char *=0)

  Creates a new Fl_Input_ widget.
- unsigned int index (int i) const

  Returns the character at index `i`.
- int input_type () const

  Gets the input field type.
- void input_type (int t)

  Sets the input field type.
- int insert (const char *t, int l=0)

  Inserts text at the cursor position.
- int insert_position () const

  Gets the position of the text cursor.
- int insert_position (int p)

  Sets the cursor position and mark.
- int insert_position (int p, int m)

  Sets the index for the cursor and mark.
- int ivalue () const

  Returns the widget text interpreted as a signed integer.
- int mark () const

  Gets the current selection mark.
- int mark (int m)

  Sets the current selection mark.
- int maximum_size () const

  Gets the maximum length of the input field in characters.
- void maximum_size (int m)

  Sets the maximum length of the input field in characters.
- const char * placeholder () const

  Get the placeholder text.
- void placeholder (const char *text)

  Set the placeholder text.
- int position () const
- int position (int p)
- int position (int p, int m)
- int readonly () const

  Gets the read-only state of the input field.
- void readonly (int b)

  Sets the read-only state of the input field.
- int redo ()

  Redo previous undo operation.
- int replace (int b, int e, const char *text, int ilen=0)

  Deletes text from `b` to `e` and inserts the new string `text`.
- void resize (int, int, int, int) override

  Changes the size of the widget.
- int shortcut () const

  Return the shortcut key associated with this widget.
- void shortcut (int s)

  Sets the shortcut key associated with this widget.
- int size () const

Returns the number of bytes in value().

- void size (int W, int H)

  Sets the width and height of this widget.
- int static_value (const char ∗)

  Changes the widget text.
- int static_value (const char ∗, int)

  Changes the widget text.
- int tab_nav () const

  Gets whether the Tab key causes focus navigation in multiline input fields or not.
- void tab_nav (int val)

  Sets whether the Tab key does focus navigation, or inserts tab characters into Fl_Multiline_Input.
- Fl_Color textcolor () const

  Gets the color of the text in the input field.
- void textcolor (Fl_Color n)

  Sets the color of the text in the input field.
- Fl_Font textfont () const

  Gets the font of the text in the input field.
- void textfont (Fl_Font s)

  Sets the font of the text in the input field.
- Fl_Fontsize textsize () const

  Gets the size of the text in the input field.
- void textsize (Fl_Fontsize s)

  Sets the size of the text in the input field.
- int undo ()

  Undoes previous changes to the text buffer.
- const char ∗ value () const

  Returns the text displayed in the widget.
- int value (const char ∗)

  Changes the widget text.
- int value (const char ∗, int)

  Changes the widget text.
- int value (double value)

  Changes the widget text to a floating point number ("%g").
- int value (int value)

  Changes the widget text to a signed integer number.
- int wrap () const

  Gets the word wrapping state of the input field.
- void wrap (int b)

  Sets the word wrapping state of the input field.
- ∼Fl_Input_ ()

  Destroys the widget.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

Returns whether the widget and all of its parents are active.

- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ **as_gl_window** () const
- virtual Fl_Group ∗ as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const ∗ **as_group** () const
- virtual Fl_Window ∗ as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ **as_window** () const
- void bind_deimage (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗cb, void ∗p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

Sets a widget to accept input.

- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- virtual void draw ()=0

  Draws the widget.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual int handle (int event)

  Handles the specified event.

- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

Sets whether this widget needs a keyboard.

- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.

- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- int apply_undo ()

  Apply the current undo/redo operation.
- void drawtext (int, int, int, int)

  Draws the text in the passed bounding box.
- void drawtext (int, int, int, int, bool draw_active)

  Draws the text in the passed bounding box.
- void handle_mouse (int, int, int, int, int keepmark=0)

  Handles mouse clicks and mouse moves.
- int handletext (int e, int, int, int, int)

  Handles all kinds of text field related events.

- int line_end (int i) const

  Finds the end of a line.
- int line_start (int i) const

  Finds the start of a line.
- int **linesPerPage** ()
- void **maybe_do_callback** (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
- int up_down_position (int, int keepmark=0)

  Moves the cursor to the column given by `up_down_pos`.
- int word_end (int i) const

  Finds the end of a word.
- int word_start (int i) const

  Finds the start of a word.
- int **xscroll** () const
- int **yscroll** () const
- void **yscroll** (int yOffset)

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.70.1 Detailed Description

This class provides a low-overhead text input field.
This is a virtual base class below Fl_Input. It has all the same interfaces, but lacks the handle() and draw() method. You may want to subclass it if you are one of those people who likes to change how the editing keys work. It may also be useful for adding scrollbars to the input field.
This can act like any of the subclasses of Fl_Input, by setting type() to one of the following values:

```
#define FL_NORMAL_INPUT           0
#define FL_FLOAT_INPUT            1
#define FL_INT_INPUT              2
#define FL_MULTILINE_INPUT        4
#define FL_SECRET_INPUT           5
#define FL_INPUT_TYPE             7
#define FL_INPUT_READONLY         8
#define FL_NORMAL_OUTPUT          (FL_NORMAL_INPUT | FL_INPUT_READONLY)
#define FL_MULTILINE_OUTPUT       (FL_MULTILINE_INPUT | FL_INPUT_READONLY)
#define FL_INPUT_WRAP             16
#define FL_MULTILINE_INPUT_WRAP   (FL_MULTILINE_INPUT | FL_INPUT_WRAP)
#define FL_MULTILINE_OUTPUT_WRAP  (FL_MULTILINE_INPUT | FL_INPUT_READONLY | FL_INPUT_WRAP)
```

All variables that represent an index into a text buffer are byte-oriented, not character oriented, counting from 0 (at or before the first character) to size() (at the end of the buffer, after the last byte). Since UTF-8 characters can be up to six bytes long, simply incrementing such an index will not reliably advance to the next character in the text buffer. Indices and pointers into the text buffer should always point at a 7 bit ASCII character or the beginning of a UTF-8 character sequence. Behavior for false UTF-8 sequences and pointers into the middle of a sequence are undefined.

See also

Fl_Text_Display, Fl_Text_Editor for more powerful text handling widgets

Fl_Widget::shortcut_label(int)

13.70.2 Constructor & Destructor Documentation

13.70.2.1 Fl_Input_()

```
Fl_Input_::Fl_Input_ (
```

```
            int X,
            int Y,
            int W,
            int H,
            const char * l = 0 )
```

Creates a new Fl_Input_ widget.
This function creates a new Fl_Input_ widget and adds it to the current Fl_Group. The value() is set to `NULL`. The default boxtype is `FL_DOWN_BOX`.

Parameters

X,Y,W,H	the dimensions of the new widget
l	an optional label text

13.70.2.2 ~Fl_Input_()

```
Fl_Input_::~Fl_Input_ ( )
```

Destroys the widget.
The destructor clears all allocated buffers and removes the widget from the parent Fl_Group.

13.70.3 Member Function Documentation

13.70.3.1 append()

```
int Fl_Input_::append (
            const char * t,
            int l = 0,
            char keep_selection = 0 )
```

Append text at the end.
This function appends the string in `t` to the end of the text. It does not moves the new position or mark.

Parameters

in	*t*	text that will be appended
in	*l*	length of text, or 0 if the string is terminated by `nul`.
in	*keep_selection*	if this is 1, the current text selection will remain, if 0, the cursor will move to the end of the inserted text.

Returns

0 if no text was appended

13.70.3.2 apply_undo()

```
int Fl_Input_::apply_undo ( )  [protected]
```

Apply the current undo/redo operation.
It's up to undo() and redo() to push and pop actions to and from the lists.

Returns

1 if the current action changed any text.

See also

undo(), redo()

13.70.3.3 can_redo()

```
bool Fl_Input_::can_redo ( ) const
```

Check if there is a redo action available.

Returns

true if the widget can redo the last undo action

13.70.3.4 can_undo()

```
bool Fl_Input_::can_undo ( ) const
```

Check if the last operation can be undone.

Returns

true if the widget can undo the last change

13.70.3.5 copy()

```
int Fl_Input_::copy (
            int clipboard )
```

Put the current selection into the clipboard.

This function copies the current selection between mark() and position() into the specified `clipboard`. This does not replace the old clipboard contents if position() and mark() are equal. Clipboard 0 maps to the current text selection and clipboard 1 maps to the cut/paste clipboard.

Parameters

clipboard	the clipboard destination 0 or 1

Returns

0 if no text is selected, 1 if the selection was copied

See also

Fl::copy(const char *, int, int)

13.70.3.6 copy_cuts()

```
int Fl_Input_::copy_cuts ( )
```

Copies the *yank* buffer to the clipboard.

This method copies all the previous contiguous cuts from the undo information to the clipboard. This function implements the ^K shortcut key.

Returns

0 if the operation did not change the clipboard

See also

copy(int), cut()

13.70.3.7 cursor_color() [1/2]

```
Fl_Color Fl_Input_::cursor_color ( ) const  [inline]
```

Gets the color of the cursor.

Returns

the current cursor color

13.70.3.8 cursor_color() [2/2]

```
void Fl_Input_::cursor_color (
            Fl_Color n )  [inline]
```

Sets the color of the cursor.
The default color for the cursor is FL_BLACK.

Parameters

in	*n*	the new cursor color

13.70.3.9 cut() [1/3]

```
int Fl_Input_::cut ( )  [inline]
```

Deletes the current selection.
This function deletes the currently selected text *without* storing it in the clipboard. To use the clipboard, you may call copy() first or copy_cuts() after this call.

Returns

0 if no data was copied

13.70.3.10 cut() [2/3]

```
int Fl_Input_::cut (
            int a,
            int b )  [inline]
```

Deletes all characters between index a and b.
This function deletes the currently selected text *without* storing it in the clipboard. To use the clipboard, you may call copy() first or copy_cuts() after this call.

Parameters

a,b	range of bytes rounded to full characters and clamped to the buffer

Returns

0 if no data was copied

13.70.3.11 cut() [3/3]

```
int Fl_Input_::cut (
            int n )  [inline]
```

Deletes the next n bytes rounded to characters before or after the cursor.
This function deletes the currently selected text *without* storing it in the clipboard. To use the clipboard, you may call copy() first or copy_cuts() after this call.

Parameters

n	number of bytes rounded to full characters and clamped to the buffer. A negative number will cut characters to the left of the cursor.

Returns

0 if no data was copied

13.70.3.12 drawtext() [1/2]

```
void Fl_Input_::drawtext (
            int X,
            int Y,
            int W,
            int H )  [protected]
```

Draws the text in the passed bounding box.
If `damage()` & `FL_DAMAGE_ALL` is true, this assumes the area has already been erased to color(). Otherwise it does minimal update and erases the area itself.

Parameters

X,Y,W,H	area that must be redrawn

13.70.3.13 drawtext() [2/2]

```
void Fl_Input_::drawtext (
            int X,
            int Y,
            int W,
            int H,
            bool draw_active )  [protected]
```

Draws the text in the passed bounding box.
This version of `drawtext` allows the user to control whether the widget is drawn as acitive, i.e. with the text cursor, or inactive. This is useful for compound widgets where the input should be shown as active when actually the container widget is the active one.
A caller should not draw the widget with `active` set if another text widget may indeed be the active widget.

Parameters

X,Y,W,H	area that must be redrawn
draw_active	if set, the cursor will be drawn, even if the widget is not active

See also

Fl_Input_::drawtext(int X, int Y, int W, int H)

13.70.3.14 dvalue()

```
double Fl_Input_::dvalue ( ) const
```

Returns the widget text interpreted as a floating point number.

Returns

double precision floating point value

See also

Fl_Input_::ivalue()

Fl_Input_::value(double)

13.70.3.15 handle_mouse()

```
void Fl_Input_::handle_mouse (
            int X,
            int Y,
```

```
            int ,
            int ,
            int drag = 0 )  [protected]
```

Handles mouse clicks and mouse moves.

Todo Add comment and parameters

13.70.3.16 handletext()

```
int Fl_Input_::handletext (
            int event,
            int X,
            int Y,
            int W,
            int H )  [protected]
```

Handles all kinds of text field related events.
This is called by derived classes.

Todo Add comment and parameters

13.70.3.17 index()

```
unsigned int Fl_Input_::index (
            int i ) const
```

Returns the character at index `i`.
This function returns the UTF-8 character at `i` as a ucs4 character code.

Parameters

in	*i*	index into the value field

Returns

the character at index `i`

13.70.3.18 input_type() [1/2]

```
int Fl_Input_::input_type ( ) const  [inline]
```

Gets the input field type.

Returns

the current input type

13.70.3.19 input_type() [2/2]

```
void Fl_Input_::input_type (
            int t )  [inline]
```

Sets the input field type.
A redraw() is required to reformat the input field.

Parameters

in	*t*	new input type

13.70.3.20 insert()

```
int Fl_Input_::insert (
            const char * t,
            int l = 0 )  [inline]
```

Inserts text at the cursor position.
This function inserts the string in `t` at the cursor position() and moves the new position and mark to the end of the inserted text.

Parameters

in	*t*	text that will be inserted
in	*l*	length of text, or 0 if the string is terminated by `nul`.

Returns

0 if no text was inserted

13.70.3.21 insert_position() [1/3]

```
int Fl_Input_::insert_position ( ) const  [inline]
```

Gets the position of the text cursor.

Returns

the cursor position as an index in the range 0..size()

See also

insert_position(int, int)

13.70.3.22 insert_position() [2/3]

```
int Fl_Input_::insert_position (
            int p )  [inline]
```

Sets the cursor position and mark.
position(n) is the same as `position(n, n)`.

Parameters

p	new index for cursor and mark

Returns

0 if no positions changed

See also

insert_position(int, int), insert_position(), mark(int)

13.70.3.23 insert_position() [3/3]

```
int Fl_Input_::insert_position (
            int p,
            int m )
```

Sets the index for the cursor and mark.

The input widget maintains two pointers into the string. The *position* (p) is where the cursor is. The *mark* (m) is the other end of the selected text. If they are equal then there is no selection. Changing this does not affect the clipboard (use copy() to do that).
Changing these values causes a redraw(). The new values are bounds checked.

Parameters

p	index for the cursor position
m	index for the mark

Returns

0 if no positions changed

See also

position(int), position(), mark(int)

13.70.3.24 ivalue()

```
int Fl_Input_::ivalue ( ) const
```

Returns the widget text interpreted as a signed integer.

Returns

signed integer value

See also

Fl_Input_::dvalue()

Fl_Input_::value(int)

13.70.3.25 line_end()

```
int Fl_Input_::line_end (
            int i ) const  [protected]
```

Finds the end of a line.
This call calculates the end of a line based on the given index `i`.

Parameters

in	*i*	starting index for the search

Returns

end of the line

13.70.3.26 line_start()

```
int Fl_Input_::line_start (
            int i ) const  [protected]
```

Finds the start of a line.
This call calculates the start of a line based on the given index `i`.

Parameters

in	*i*	starting index for the search

Returns

start of the line

13.70.3.27 mark() [1/2]

```
int Fl_Input_::mark ( ) const  [inline]
```

Gets the current selection mark.

Returns

index into the text

13.70.3.28 mark() [2/2]

```
int Fl_Input_::mark (
            int m )  [inline]
```

Sets the current selection mark.
mark(n) is the same as `insert_position(insert_position(),n)`.

Parameters

m	new index of the mark

Returns

0 if the mark did not change

See also

insert_position(), insert_position(int, int)

13.70.3.29 maximum_size() [1/2]

```
int Fl_Input_::maximum_size ( ) const  [inline]
```

Gets the maximum length of the input field in characters.

See also

maximum_size(int).

13.70.3.30 maximum_size() [2/2]

```
void Fl_Input_::maximum_size (
            int m )  [inline]
```

Sets the maximum length of the input field in characters.
This limits the number of **characters** that can be inserted in the widget.
Since FLTK 1.3 this is different than the buffer size, since one character can be more than one byte in UTF-8 encoding. In FLTK 1.1 this was the same (one byte = one character).

13.70.3.31 placeholder() [1/2]

```
const char * Fl_Input_::placeholder ( ) const
```

Get the placeholder text.

Returns

pointer to an internal buffer to the placeholder text, or nullptr. The buffer will be freed when the widget is destroyed or a new placeholder is set.

13.70.3.32 placeholder() [2/2]

```
void Fl_Input_::placeholder (
            const char * text )
```

Set the placeholder text.

Parameters

in	*text*	the placeholder text

13.70.3.33 position() [1/3]

```
int Fl_Input_::position ( ) const  [inline]
```

Deprecated "since 1.4.0 - use insert_position() instead"

13.70.3.34 position() [2/3]

```
int Fl_Input_::position (
            int p )  [inline]
```

Deprecated "since 1.4.0 - use insert_position(p) instead"

13.70.3.35 position() [3/3]

```
int Fl_Input_::position (
            int p,
            int m )  [inline]
```

Deprecated "since 1.4.0 - use insert_position(p, m) or Fl_Widget::position(x, y) instead"

13.70.3.36 readonly() [1/2]

```
int Fl_Input_::readonly ( ) const  [inline]
```

Gets the read-only state of the input field.

Returns

non-zero if this widget is read-only

13.70.3.37 readonly() [2/2]

```
void Fl_Input_::readonly (
            int b )  [inline]
```

Sets the read-only state of the input field.

Parameters

in	*b*	if b is 0, the text in this widget can be edited by the user

13.70.3.38 redo()

```
int Fl_Input_::redo ( )
```

Redo previous undo operation.
This call reapplies previously executed undo operations.

Returns

non-zero if any change was made.

13.70.3.39 replace()

```
int Fl_Input_::replace (
            int b,
            int e,
            const char * text,
            int ilen = 0 )
```

Deletes text from `b` to `e` and inserts the new string `text`.
All changes to the text buffer go through this function. It deletes the region between `b` and `e` (either one may be less or equal to the other), and then inserts the string `text` at that point and moves the mark() and position() to the end of the insertion. Does the callback if `when() & FL_WHEN_CHANGED` and there is a change.
Set `b` and `e` equal to not delete anything. Set `text` to `NULL` to not insert anything.
`ilen` can be zero or `strlen(text)`, which saves a tiny bit of time if you happen to already know the length of the insertion, or can be used to insert a portion of a string. If `ilen` is zero, `strlen(text)` is used instead.
`b` and `e` are clamped to the `0..size()` range, so it is safe to pass any values. `b`, `e`, and `ilen` are used as numbers of bytes (not characters), where `b` and `e` count from 0 to size() (end of buffer).
If `b` and/or `e` don't point to a valid UTF-8 character boundary, they are adjusted to the previous (`b`) or the next (`e`) valid UTF-8 character boundary, resp..
If the current number of characters in the buffer minus deleted characters plus inserted characters in `text` would overflow the number of allowed characters (maximum_size()), then only the first characters of the string are inserted, so that maximum_size() is not exceeded.
cut() and insert() are just inline functions that call replace().

Parameters

in	*b*	beginning index of text to be deleted
in	*e*	ending index of text to be deleted and insertion position
in	*text*	string that will be inserted
in	*ilen*	length of `text` or 0 for `nul` terminated strings

Returns

0 if nothing changed

Note

If `text` does not point to a valid UTF-8 character or includes invalid UTF-8 sequences, the text is inserted nevertheless (counting invalid UTF-8 bytes as one character each).

13.70.3.40 resize()

```
void Fl_Input_::resize (
            int X,
            int Y,
            int W,
            int H )  [override], [virtual]
```

Changes the size of the widget.
This call updates the text layout so that the cursor is visible.

Parameters

in	*X,Y,W,H*	new size of the widget

See also

Fl_Widget::resize(int, int, int, int)

Reimplemented from Fl_Widget.

13.70.3.41 shortcut() [1/2]

```
int Fl_Input_::shortcut ( ) const  [inline]
```

Return the shortcut key associated with this widget.

Returns

shortcut keystroke

See also

Fl_Button::shortcut()

13.70.3.42 shortcut() [2/2]

```
void Fl_Input_::shortcut (
            int s )  [inline]
```

Sets the shortcut key associated with this widget.
Pressing the shortcut key gives text editing focus to this widget.

Parameters

in	*s*	new shortcut keystroke

See also

Fl_Button::shortcut()

13.70.3.43 size() [1/2]

```
int Fl_Input_::size ( ) const  [inline]
```

Returns the number of bytes in value().
This may be greater than `strlen(value())` if there are `nul` characters in the text.

Returns

number of bytes in the text

13.70.3.44 size() [2/2]

```
void Fl_Input_::size (
            int W,
            int H )  [inline]
```

Sets the width and height of this widget.

Parameters

in	*W,H*	new width and height

See also

Fl_Widget::size(int, int)

13.70.3.45 static_value() [1/2]

```
int Fl_Input_::static_value (
            const char * str )
```

Changes the widget text.
This function changes the text and sets the mark and the point to the end of it. The string is *not* copied. If the user edits the string it is copied to the internal buffer then. This can save a great deal of time and memory if your program is rapidly changing the values of text fields, but this will only work if the passed string remains unchanged until either the Fl_Input is destroyed or value() is called again.

Parameters

in	*str*	the new text

Returns

non-zero if the new value is different than the current one

13.70.3.46 static_value() [2/2]

```
int Fl_Input_::static_value (
            const char * str,
            int len )
```

Changes the widget text.
This function changes the text and sets the mark and the point to the end of it. The string is *not* copied. If the user edits the string it is copied to the internal buffer then. This can save a great deal of time and memory if your program is rapidly changing the values of text fields, but this will only work if the passed string remains unchanged until either the Fl_Input is destroyed or value() is called again.
You can use the `len` parameter to directly set the length if you know it already or want to put `nul` characters in the text.

Parameters

in	*str*	the new text
in	*len*	the length of the new text

Returns

non-zero if the new value is different than the current one

13.70.3.47 tab_nav() [1/2]

```
int Fl_Input_::tab_nav ( ) const  [inline]
```

Gets whether the Tab key causes focus navigation in multiline input fields or not.
If enabled (default), hitting Tab causes focus navigation to the next widget.
If disabled, hitting Tab inserts a tab character into the text field.

Returns

1 if Tab advances focus (default), 0 if Tab inserts tab characters.

See also

tab_nav(int), Fl::OPTION_ARROW_FOCUS.

13.70.3.48 tab_nav() [2/2]

```
void Fl_Input_::tab_nav (
            int val )  [inline]
```

Sets whether the Tab key does focus navigation, or inserts tab characters into Fl_Multiline_Input.
By default this flag is enabled to provide the 'normal' behavior most users expect; Tab navigates focus to the next widget. To inserting an actual Tab character, users can use Ctrl-I or copy/paste.
Disabling this flag gives the old FLTK behavior where Tab inserts a tab character into the text field, in which case only the mouse can be used to navigate to the next field.
History: This flag was provided for backwards support of FLTK's old 1.1.x behavior where Tab inserts a tab character instead of navigating focus to the next widget. This behavior was unique to Fl_Multiline_Input. With the advent of Fl_Text_Editor, this old behavior has been deprecated.

Parameters

in	*val*	If `val` is 1, Tab advances focus (default). If `val` is 0, Tab inserts a tab character (old FLTK behavior).

See also

tab_nav(), Fl::OPTION_ARROW_FOCUS.

13.70.3.49 textcolor() [1/2]

```
Fl_Color Fl_Input_::textcolor ( ) const  [inline]
```

Gets the color of the text in the input field.

Returns

the text color

See also

textcolor(Fl_Color)

13.70.3.50 textcolor() [2/2]

```
void Fl_Input_::textcolor (
            Fl_Color n )  [inline]
```

Sets the color of the text in the input field.
The text color defaults to `FL_FOREGROUND_COLOR`.

Parameters

in	*n*	new text color

See also

textcolor()

13.70.3.51 textfont() [1/2]

```
Fl_Font Fl_Input_::textfont ( ) const  [inline]
```

Gets the font of the text in the input field.

Returns

the current Fl_Font index

13.70.3.52 textfont() [2/2]

```
void Fl_Input_::textfont (
            Fl_Font s ) [inline]
```

Sets the font of the text in the input field.
The text font defaults to FL_HELVETICA.

Parameters

in	*s*	the new text font

13.70.3.53 textsize() [1/2]

```
Fl_Fontsize Fl_Input_::textsize ( ) const [inline]
```

Gets the size of the text in the input field.

Returns

the text height in pixels

13.70.3.54 textsize() [2/2]

```
void Fl_Input_::textsize (
            Fl_Fontsize s ) [inline]
```

Sets the size of the text in the input field.
The text height defaults to FL_NORMAL_SIZE.

Parameters

in	*s*	the new font height in pixel units

13.70.3.55 undo()

```
int Fl_Input_::undo ( )
```

Undoes previous changes to the text buffer.
This call undoes a number of previous calls to replace().

Returns

non-zero if any change was made.

13.70.3.56 up_down_position()

```
int Fl_Input_::up_down_position (
            int i,
            int keepmark = 0 ) [protected]
```

Moves the cursor to the column given by up_down_pos.
This function is helpful when implementing up and down cursor movement. It moves the cursor from the beginning of a line to the column indicated by the global variable up_down_pos in pixel units.

Parameters

in	*i*	index into the beginning of a line of text
in	*keepmark*	if set, move only the cursor, but not the mark

Returns

index to new cursor position

13.70.3.57 value() [1/5]

```
const char * Fl_Input_::value ( ) const  [inline]
```

Returns the text displayed in the widget.
This function returns the current value, which is a pointer to the internal buffer and is valid only until the next event is handled.

Returns

pointer to an internal buffer - do not free() this

See also

Fl_Input_::value(const char*)

13.70.3.58 value() [2/5]

```
int Fl_Input_::value (
            const char * str )
```

Changes the widget text.
This function changes the text and sets the mark and the point to the end of it. The string is copied to the internal buffer. Passing `NULL` is the same as `""`.

Parameters

in	*str*	the new text

Returns

non-zero if the new value is different than the current one

See also

Fl_Input_::value(const char* str, int len), Fl_Input_::value()

13.70.3.59 value() [3/5]

```
int Fl_Input_::value (
            const char * str,
            int len )
```

Changes the widget text.
This function changes the text and sets the mark and the point to the end of it. The string is copied to the internal buffer. Passing `NULL` is the same as "".
You can use the `length` parameter to directly set the length if you know it already or want to put `nul` characters in the text.

Parameters

in	*str*	the new text
in	*len*	the length of the new text

Returns

non-zero if the new value is different than the current one

See also

Fl_Input_::value(const char* str), Fl_Input_::value()

13.70.3.60 value() [4/5]

```
int Fl_Input_::value (
            double v )
```

Changes the widget text to a floating point number ("%g").

Parameters

in	*v*	the new value

Returns

non-zero if the new value is different than the current one

See also

Fl_Input_::value(const char* str), Fl_Input_::ivalue()

13.70.3.61 value() [5/5]

```
int Fl_Input_::value (
            int v )
```

Changes the widget text to a signed integer number.

Parameters

in	*v*	the new value

Returns

non-zero if the new value is different than the current one

See also

Fl_Input_::value(const char* str), Fl_Input_::ivalue()

13.70.3.62 word_end()

```
int Fl_Input_::word_end (
            int i ) const  [protected]
```

Finds the end of a word.
Returns the index after the last byte of a word. If the index is already at the end of a word, it will find the end of the following word, so if you call it repeatedly you will move forwards to the end of the text.
Note that this is inconsistent with line_end().

Parameters

in	*i*	starting index for the search

Returns

end of the word

13.70.3.63 word_start()

```
int Fl_Input_::word_start (
            int i ) const  [protected]
```

Finds the start of a word.
Returns the index of the first byte of a word. If the index is already at the beginning of a word, it will find the beginning of the previous word, so if you call it repeatedly you will move backwards to the beginning of the text.
Note that this is inconsistent with line_start().

Parameters

in	*i*	starting index for the search

Returns

start of the word, or previous word

13.70.3.64 wrap() [1/2]

```
int Fl_Input_::wrap ( ) const  [inline]
```

Gets the word wrapping state of the input field.
Word wrap is only functional with multi-line input fields.

13.70.3.65 wrap() [2/2]

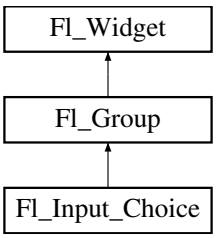
```
void Fl_Input_::wrap (
            int b )  [inline]
```

Sets the word wrapping state of the input field.
Word wrap is only functional with multi-line input fields.
The documentation for this class was generated from the following files:

- Fl_Input_.H
- Fl_Input_.cxx

13.71 Fl_Input_Choice Class Reference

A combination of the input widget and a menu button.
Inheritance diagram for Fl_Input_Choice:



Public Member Functions

- void add (const char *s)

  Adds an item to the menu.
- int **changed** () const

Returns the combined changed() state of the input and menu button widget.

- void **clear** ()

  Removes all items from the menu.

- void **clear_changed** ()

  Clears the changed() state of both input and menu button widgets.

- Fl_Boxtype **down_box** () const

  Gets the box type of the menu button.

- void **down_box** (Fl_Boxtype b)

  Sets the box type of the menu button.

- Fl_Input_Choice (int X, int Y, int W, int H, const char ∗L=0)

  Creates a new Fl_Input_Choice widget using the given position, size, and label string.

- Fl_Input ∗ input () const

  Returns a pointer to the internal Fl_Input widget.

- const Fl_Menu_Item ∗ **menu** ()

  Gets the Fl_Menu_Item array used for the menu.

- void **menu** (const Fl_Menu_Item ∗m)

  Sets the Fl_Menu_Item array used for the menu.

- Fl_Menu_Button ∗ menubutton () const

  Returns a pointer to the internal Fl_Menu_Button widget.

- void resize (int X, int Y, int W, int H) override

  Resizes the Fl_Input_Choice widget.

- void **set_changed** ()

  Sets the changed() state of both input and menu button widgets to the specified value.

- Fl_Color **textcolor** () const

  Gets the Fl_Input text field's text color.

- void **textcolor** (Fl_Color c)

  Sets the Fl_Input text field's text color to `c`.

- Fl_Font **textfont** () const

  Gets the Fl_Input text field's font style.

- void **textfont** (Fl_Font f)

  Sets the Fl_Input text field's font style to `f`.

- Fl_Fontsize **textsize** () const

  Gets the Fl_Input text field's font size.

- void **textsize** (Fl_Fontsize s)

  Sets the Fl_Input text field's font size to `s`.

- int update_menubutton ()

  Updates the menubutton with the string value in Fl_Input.

- const char ∗ **value** () const

  Returns the Fl_Input text field's current contents.

- void value (const char ∗val)

  Sets the Fl_Input text field's contents to `val`.

- void value (int val)

  Chooses item# `val` in the menu, and sets the Fl_Input text field to that value.

Public Member Functions inherited from Fl_Group

- Fl_Widget *& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget *o)

  See void Fl_Group::add(Fl_Widget &w)
- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget *const * array () const

  Returns a pointer to the internal array of children.
- Fl_Group const * as_group () const override
- Fl_Group * as_group () override

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const

  Returns the n'th child.
- int **children** () const

  Returns how many child widgets the group has.
- void clear ()

  Deletes all child widgets from memory recursively.
- unsigned int clip_children () const

  Returns the current clipping mode.
- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)

  Removes the widget at `index` from the group and deletes it.
- void end ()

  Exactly the same as current(this->parent()).
- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*
- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.
- void **forms_end** ()

  This is for forms compatibility only.
- int handle (int) override

  Handles the specified event.
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

  This does insert(w, find(before)).
- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.

- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.
- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.

- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

Sets the image to use as part of the widget label when in the inactive state.

- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.

- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

Gives the widget the keyboard focus.

- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void draw () override

  Draws the widget.
- virtual int **inp_h** () const

  See inp_x() for info.
- virtual int **inp_w** () const

  See inp_x() for info.
- virtual int inp_x () const

  The methods inp_x(), inp_y(), inp_w() and inp_h() return the desired position and size of the internal Fl_Input widget.
- virtual int **inp_y** () const

  See inp_x() for info.
- virtual int **menu_h** () const

  See menu_x() for info.
- virtual int **menu_w** () const

  See menu_x() for info.
- virtual int menu_x () const

  The methods menu_x(), menu_y(), menu_w() and menu_h() return the desired position and size of the internal Fl_Menu_Button widget.
- virtual int **menu_y** () const

  See menu_x() for info.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()

  Returns the internal array of widget sizes and positions.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget *, int)

  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)

  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)

  Allow derived groups to act when a child widget is removed from the group.
- int * sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

Draws a box of type t, of color c at the widget's position and size.

- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()

  Returns the currently active group.
- static void current (Fl_Group *g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.71.1 Detailed Description

A combination of the input widget and a menu button.

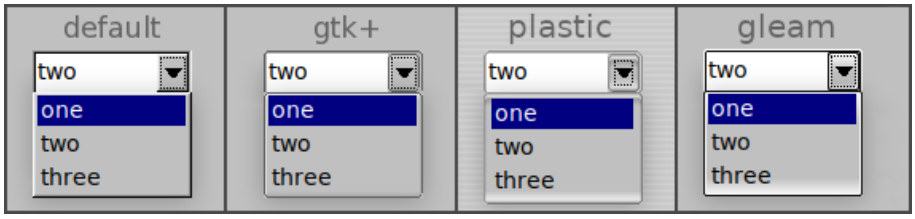


Figure 13.26 Fl_Input_Choice widget

The user can either type into the input area, or use the menu button chooser on the right to choose an item which loads the input area with the selected text.
The application can directly access both the internal Fl_Input and Fl_Menu_Button widgets respectively using the input() and menubutton() accessor methods.
The default behavior is to invoke the Fl_Input_Choice::callback() if the user changes the input field's contents, either by typing, pasting, or clicking a different item in the choice menu.
The callback can determine if an item was picked vs. typing into the input field by checking the value of menubutton()->changed(), which will be:

```
- 1: the user picked a different item in the choice menu
- 0: the user typed or pasted directly into the input field
```

FLTK triggers an `FL_BEFORE_MENU` event for this widget right before displaying the menu. This event provides an opportunity to update menu item states and activation.

Example Use of Fl_Input_Choice

```
#include <stdio.h>
#include <FL/Fl.H>
#include <FL/Fl_Double_Window.H>
#include <FL/Fl_Input_Choice.H>
// Fl_Input_Choice callback()
void choice_cb(Fl_Widget *w, void *userdata) {
  // Show info about the picked item
  Fl_Input_Choice *choice = (Fl_Input_Choice*)w;
  printf("*** Choice Callback:\n");
  printf("    widget's text value='%s'\n", choice->value());   // normally all you need
  // Access the menu via menubutton()..
  const Fl_Menu_Item *item = choice->menubutton()->mvalue();
  printf("    item label()='%s'\n", item ? item->label() : "(No item)");
  printf("    item value()=%d\n", choice->menubutton()->value());
  printf("    input value()='%s'\n", choice->input()->value());
  printf("    The user %s\n", choice->menubutton()->changed()
                                ? "picked a menu item"
                                : "typed text");
```

```
}
int main() {
  Fl_Double_Window win(200,100,"Input Choice");
  win.begin();
    Fl_Input_Choice choice(10,10,100,30);
    choice.callback(choice_cb, 0);
    choice.add("Red");
    choice.add("Orange");
    choice.add("Yellow");
    //choice.value("Red");    // uncomment to make "Red" default
  win.end();
  win.show();
  return Fl::run();
}
```

Subclassing Example

One can subclass Fl_Input_Choice to override the virtual methods inp_x/y/w/h() and menu_x/y/w/h() to take control of the internal Fl_Input and Fl_Menu_Button widget positioning. In this example, input and menubutton's positions are swapped:

```
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Input_Choice.H>

class MyInputChoice : public Fl_Input_Choice {
protected:
  virtual int inp_x()  const { return x() + Fl::box_dx(box()) + menu_w(); }  // override to reposition
  virtual int menu_x() const { return x() + Fl::box_dx(box()); }              // override to reposition
public:
  MyInputChoice(int X,int Y,int W,int H,const char*L=0) : Fl_Input_Choice(X,Y,W,H,L) {
    resize(X,Y,W,H);  // necessary for ctor to trigger our overrides
  }
};

int main(int argc, char **argv) {
  Fl_Window *win = new Fl_Window(400,300);
  MyInputChoice *mychoice = new MyInputChoice(150,40,150,25,"Right Align Input");
  mychoice->add("Aaa");
  mychoice->add("Bbb");
  mychoice->add("Ccc");
  win->end();
  win->resizable(win);
  win->show();
  return Fl::run();
}
```

13.71.2 Constructor & Destructor Documentation

13.71.2.1 Fl_Input_Choice()

```
Fl_Input_Choice::Fl_Input_Choice (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

Creates a new Fl_Input_Choice widget using the given position, size, and label string.
Inherited destructor destroys the widget and any values associated with it.

13.71.3 Member Function Documentation

13.71.3.1 add()

```
void Fl_Input_Choice::add (
            const char * s )  [inline]
```

Adds an item to the menu.
When any item is selected, the Fl_Input_Choice callback() is invoked, which can do something with the selected item.
You can access the more complex Fl_Menu_Button::add() methods (setting item-specific callbacks, userdata, etc), via menubutton(). Example:

```
Fl_Input_Choice *choice = new Fl_Input_Choice(100,10,120,25,"Fonts");
Fl_Menu_Button *mb = choice->menubutton();             // use Fl_Input_Choice's Fl_Menu_Button
mb->add("Helvetica", 0, MyFont_CB,     (void*)mydata); // use Fl_Menu_Button's add() methods
```

```
mb->add("Courier",    0, MyFont_CB,      (void*)mydata);
mb->add("More..",     0, FontDialog_CB, (void*)mydata);
```

13.71.3.2 draw()

```
void Fl_Input_Choice::draw ( )  [override], [protected], [virtual]
```

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();                 // calls Fl_Scrollbar::draw()
```

Reimplemented from Fl_Group.

13.71.3.3 inp_x()

```
virtual int Fl_Input_Choice::inp_x ( ) const  [inline], [protected], [virtual]
```

The methods inp_x(), inp_y(), inp_w() and inp_h() return the desired position and size of the internal Fl_Input widget. These can be overridden by a subclass to redefine positioning. See code example in the Description for subclassing details.

13.71.3.4 input()

```
Fl_Input * Fl_Input_Choice::input ( ) const  [inline]
```

Returns a pointer to the internal Fl_Input widget.

This can be used to directly access all of the Fl_Input widget's methods.

13.71.3.5 menu_x()

```
virtual int Fl_Input_Choice::menu_x ( ) const  [inline], [protected], [virtual]
```

The methods menu_x(), menu_y(), menu_w() and menu_h() return the desired position and size of the internal Fl_Menu_Button widget.

These can be overridden by a subclass to redefine positioning. See code example in the Description for subclassing details.

13.71.3.6 menubutton()

```
Fl_Menu_Button * Fl_Input_Choice::menubutton ( ) const  [inline]
```

Returns a pointer to the internal Fl_Menu_Button widget.

This can be used to access any of the methods of the menu button, e.g.

```
Fl_Input_Choice *choice = new Fl_Input_Choice(100,10,120,25,"Choice:");
[..]
// Print all the items in the choice menu
for ( int t=0; t<choice->menubutton()->size(); t++ ) {
   const Fl_Menu_Item &item = choice->menubutton()->menu()[t];
   printf("item %d -- label=%s\n", t, item.label() ? item.label() : "(Null)");
}
```

13.71.3.7 resize()

```
void Fl_Input_Choice::resize (
            int X,
            int Y,
            int W,
            int H )  [override], [virtual]
```

Resizes the Fl_Input_Choice widget.

Reimplemented from Fl_Group.

13.71.3.8 update_menubutton()

```
int Fl_Input_Choice::update_menubutton ( )
```

Updates the menubutton with the string value in Fl_Input.
If the string value currently in Fl_Input matches one of the menu items in menubutton(), that menu item will become the current item selected.
Call this method after setting value(const char*) if you need the menubutton() to be synchronized with the Fl_Input field.

```
// Add items
choice->add(".25");
choice->add(".50");
choice->add("1.0");
choice->add("2.0");
choice->add("4.0");

choice->value("1.0");            // sets Fl_Input to "1.0"
choice->update_menubutton();     // cause menubutton to reflect this value too
                                 // (returns 1 if match was found, 0 if not)
// Verify menubutton()'s value.
printf("menu button choice index=%d, value=%s\n",
                           choice->menubutton()->value(),    // would be -1 if update not done
                           choice->menubutton()->text());    // would be NULL if update not done
```

Returns

1 if a matching menuitem was found and value set, 0 if not.

Version

1.4.0

13.71.3.9 value() [1/2]

```
void Fl_Input_Choice::value (
            const char * val )  [inline]
```

Sets the Fl_Input text field's contents to `val`.
Note it is possible to set the value() to one that is not in the menubutton's list of choices.
Setting the value() does NOT affect the menubutton's selection. If that's needed, call update_menubutton() after setting value().

See also

void value(int val), update_menubutton()

13.71.3.10 value() [2/2]

```
void Fl_Input_Choice::value (
            int val )
```

Chooses item# `val` in the menu, and sets the Fl_Input text field to that value.
Any previous text is cleared.

See also

void value(const char *val)

The documentation for this class was generated from the following files:

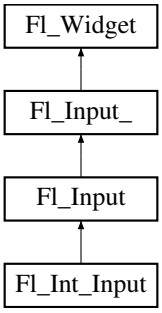
- Fl_Input_Choice.H
- Fl_Input_Choice.cxx

13.72 Fl_Int_Input Class Reference

The Fl_Int_Input class is a subclass of Fl_Input that only allows the user to type decimal digits (or hex numbers of the form 0xaef).

```
#include <Fl_Int_Input.H>
```

Inheritance diagram for Fl_Int_Input:



Public Member Functions

- Fl_Int_Input (int X, int Y, int W, int H, const char *l=0)

  Creates a new Fl_Int_Input widget using the given position, size, and label string.

Public Member Functions inherited from Fl_Input

- Fl_Input (int, int, int, int, const char *=0)

  Creates a new Fl_Input widget using the given position, size, and label string.
- int handle (int) override

  Handles the specified event.

Public Member Functions inherited from Fl_Input_

- int append (const char *t, int l=0, char keep_selection=0)

  Append text at the end.
- bool can_redo () const

  Check if there is a redo action available.
- bool can_undo () const

  Check if the last operation can be undone.
- int copy (int clipboard)

  Put the current selection into the clipboard.
- int copy_cuts ()

  Copies the yank buffer to the clipboard.
- Fl_Color cursor_color () const

  Gets the color of the cursor.
- void cursor_color (Fl_Color n)

  Sets the color of the cursor.
- int cut ()

  Deletes the current selection.
- int cut (int a, int b)

  Deletes all characters between index `a` and `b`.
- int cut (int n)

  Deletes the next `n` bytes rounded to characters before or after the cursor.
- double dvalue () const

  Returns the widget text interpreted as a floating point number.
- Fl_Input_ (int, int, int, int, const char *=0)

  Creates a new Fl_Input_ widget.
- unsigned int index (int i) const

  Returns the character at index `i`.

- int input_type () const

  Gets the input field type.
- void input_type (int t)

  Sets the input field type.
- int insert (const char *t, int l=0)

  Inserts text at the cursor position.
- int insert_position () const

  Gets the position of the text cursor.
- int insert_position (int p)

  Sets the cursor position and mark.
- int insert_position (int p, int m)

  Sets the index for the cursor and mark.
- int ivalue () const

  Returns the widget text interpreted as a signed integer.
- int mark () const

  Gets the current selection mark.
- int mark (int m)

  Sets the current selection mark.
- int maximum_size () const

  Gets the maximum length of the input field in characters.
- void maximum_size (int m)

  Sets the maximum length of the input field in characters.
- const char * placeholder () const

  Get the placeholder text.
- void placeholder (const char *text)

  Set the placeholder text.
- int position () const
- int position (int p)
- int position (int p, int m)
- int readonly () const

  Gets the read-only state of the input field.
- void readonly (int b)

  Sets the read-only state of the input field.
- int redo ()

  Redo previous undo operation.
- int replace (int b, int e, const char *text, int ilen=0)

  Deletes text from `b` *to* `e` *and inserts the new string* `text`.
- void resize (int, int, int, int) override

  Changes the size of the widget.
- int shortcut () const

  Return the shortcut key associated with this widget.
- void shortcut (int s)

  Sets the shortcut key associated with this widget.
- int size () const

  Returns the number of bytes in value().
- void size (int W, int H)

  Sets the width and height of this widget.
- int static_value (const char *)

  Changes the widget text.
- int static_value (const char *, int)

  Changes the widget text.

- int tab_nav () const

  Gets whether the Tab key causes focus navigation in multiline input fields or not.
- void tab_nav (int val)

  Sets whether the Tab key does focus navigation, or inserts tab characters into Fl_Multiline_Input.
- Fl_Color textcolor () const

  Gets the color of the text in the input field.
- void textcolor (Fl_Color n)

  Sets the color of the text in the input field.
- Fl_Font textfont () const

  Gets the font of the text in the input field.
- void textfont (Fl_Font s)

  Sets the font of the text in the input field.
- Fl_Fontsize textsize () const

  Gets the size of the text in the input field.
- void textsize (Fl_Fontsize s)

  Sets the size of the text in the input field.
- int undo ()

  Undoes previous changes to the text buffer.
- const char ∗ value () const

  Returns the text displayed in the widget.
- int value (const char ∗)

  Changes the widget text.
- int value (const char ∗, int)

  Changes the widget text.
- int value (double value)

  Changes the widget text to a floating point number ("%g").
- int value (int value)

  Changes the widget text to a signed integer number.
- int wrap () const

  Gets the word wrapping state of the input field.
- void wrap (int b)

  Sets the word wrapping state of the input field.
- ∼Fl_Input_ ()

  Destroys the widget.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.

- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.

- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

Sets the image to use as part of the widget label when in the active state.

- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.

- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

Sets the user data for this widget.

- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget ∗widget, void ∗data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char ∗t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char ∗, const bool require_alt=false)

  Returns true if the given text `t` *contains the entered '&x' shortcut.*

Static Public Attributes inherited from Fl_Input

- static const char ∗ **copy_menu_text** = "Copy"

  [this text may be customized at run-time]
- static const char ∗ **cut_menu_text** = "Cut"

  [this text may be customized at run-time]
- static const char ∗ **paste_menu_text** = "Paste"

  [this text may be customized at run-time]

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Input

- void draw () override

  Draws the widget.
- int handle_key ()

  Handles a keystroke.
- int handle_rmb ()

  Handle right mouse button down events.

Protected Member Functions inherited from Fl_Input_

- int apply_undo ()

  Apply the current undo/redo operation.
- void drawtext (int, int, int, int)

  Draws the text in the passed bounding box.
- void drawtext (int, int, int, int, bool draw_active)

  Draws the text in the passed bounding box.
- void handle_mouse (int, int, int, int, int keepmark=0)

  Handles mouse clicks and mouse moves.
- int handletext (int e, int, int, int, int)

  Handles all kinds of text field related events.
- int line_end (int i) const

  Finds the end of a line.
- int line_start (int i) const

  Finds the start of a line.
- int **linesPerPage** ()
- void **maybe_do_callback** (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
- int up_down_position (int, int keepmark=0)

  Moves the cursor to the column given by `up_down_pos`.
- int word_end (int i) const

  Finds the end of a word.
- int word_start (int i) const

  Finds the start of a word.
- int **xscroll** () const
- int **yscroll** () const
- void **yscroll** (int yOffset)

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

13.72.1 Detailed Description

The Fl_Int_Input class is a subclass of Fl_Input that only allows the user to type decimal digits (or hex numbers of the form 0xaef).

13.72.2 Constructor & Destructor Documentation

13.72.2.1 Fl_Int_Input()

```
Fl_Int_Input::Fl_Int_Input (
            int X,
            int Y,
            int W,
            int H,
            const char * l = 0 )
```

Creates a new Fl_Int_Input widget using the given position, size, and label string.
The default boxtype is FL_DOWN_BOX.
Inherited destructor destroys the widget and any value associated with it.
The documentation for this class was generated from the following files:

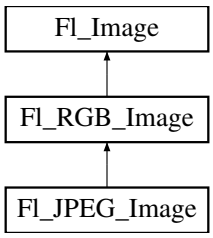
- Fl_Int_Input.H
- Fl_Input.cxx

13.73 Fl_JPEG_Image Class Reference

The Fl_JPEG_Image class supports loading, caching, and drawing of Joint Photographic Experts Group (JPEG) File Interchange Format (JFIF) images.

`#include <Fl_JPEG_Image.H>`

Inheritance diagram for Fl_JPEG_Image:



Public Member Functions

- Fl_JPEG_Image (const char *filename)

  The constructor loads the JPEG image from the given jpeg filename.
- Fl_JPEG_Image (const char *name, const unsigned char *data, int data_length=-1)

  The constructor loads the JPEG image from memory.

Public Member Functions inherited from Fl_RGB_Image

- virtual Fl_SVG_Image * as_svg_image ()

  Returns whether an RGB image is an Fl_SVG_Image or not.
- virtual const Fl_SVG_Image * as_svg_image () const

  Returns whether an RGB image is an Fl_SVG_Image or not.
- int **cache_h** () const
- int **cache_w** () const
- void color_average (Fl_Color c, float i) override

  The color_average() method averages the colors in the image with the provided FLTK color value.
- Fl_Image * **copy** () const
- Fl_Image * copy (int W, int H) const override

  Creates a resized copy of the image.
- void desaturate () override

  The desaturate() method converts an image to grayscale.
- void **draw** (int X, int Y)
- void draw (int X, int Y, int W, int H, int cx=0, int cy=0) override

  Draws the image to the current drawing surface with a bounding box.
- Fl_RGB_Image (const Fl_Pixmap *pxm, Fl_Color bg=FL_GRAY)

  The constructor creates a new RGBA image from the specified Fl_Pixmap.
- Fl_RGB_Image (const uchar *bits, int bits_length, int W, int H, int D, int LD)

  The constructor creates a new image from the specified data.
- Fl_RGB_Image (const uchar *bits, int W, int H, int D=3, int LD=0)

  The constructor creates a new image from the specified data.
- void label (Fl_Menu_Item *m) override

  This method is an obsolete way to set the image attribute of a menu item.
- void label (Fl_Widget *w) override

This method is an obsolete way to set the image attribute of a widget or menu item.

- virtual void normalize ()

  Makes sure the object is fully initialized.

- void uncache () override

  If the image has been cached for display, delete the cache data.

- ~**Fl_RGB_Image** () override

  The destructor frees all memory and server resources that are used by the image.

Public Member Functions inherited from Fl_Image

- virtual class Fl_Shared_Image * as_shared_image ()

  Returns whether an image is an Fl_Shared_Image or not.

- Fl_Image * copy () const

  Creates a copy of the image in the same size.

- int count () const

  Returns the number of data values associated with the image.

- int d () const

  Returns the image depth.

- const char *const * data () const

  Returns a pointer to the current image data array.

- int **data_h** () const

  Returns the height of the image data.

- int **data_w** () const

  Returns the width of the image data.

- void draw (int X, int Y)

  Draws the image to the current drawing surface.

- int fail () const

  Returns a value that is not 0 if there is currently no image available.

- Fl_Image (int W, int H, int D)

  The constructor creates an empty image with the specified width, height, and depth.

- int h () const

  Returns the current image drawing height in FLTK units.

- void inactive ()

  The inactive() method calls color_average(FL_BACKGROUND_COLOR, 0.33f) to produce an image that appears grayed out.

- int ld () const

  Returns the current line data size in bytes.

- virtual void release ()

  Releases an Fl_Image - the same as '`delete this`'.

- virtual void scale (int width, int height, int proportional=1, int can_expand=0)

  Sets the drawing size of the image.

- int w () const

  Returns the current image drawing width in FLTK units.

- virtual ~**Fl_Image** ()

  The destructor is a virtual method that frees all memory used by the image.

Protected Member Functions

- void **load_jpg_** (const char *filename, const char *sharename, const unsigned char *data, int data_length=-1)

Protected Member Functions inherited from Fl_Image

- void **d** (int D)

  Sets the current image depth.
- void data (const char ∗const ∗p, int c)

  Sets the current data pointer and count of pointers in the array.
- void draw_empty (int X, int Y)

  The protected method draw_empty() draws a box with an X in it.
- int draw_scaled (int X, int Y, int W, int H)

  Draw the image to the current drawing surface rescaled to a given width and height.
- void h (int H)

  Sets the height of the image data.
- void ld (int LD)

  Sets the current line data size in bytes.
- void w (int W)

  Sets the width of the image data.

Additional Inherited Members

Static Public Member Functions inherited from Fl_RGB_Image

- static size_t max_size ()

  Returns the maximum allowed image size in bytes when creating an Fl_RGB_Image object.
- static void max_size (size_t size)

  Sets the maximum allowed image size in bytes when creating an Fl_RGB_Image object.

Static Public Member Functions inherited from Fl_Image

- static Fl_Labeltype **define_FL_IMAGE_LABEL** ()
- static Fl_RGB_Scaling **RGB_scaling** ()

  Returns the currently used RGB image scaling method.
- static void RGB_scaling (Fl_RGB_Scaling)

  Sets the RGB image scaling method used for copy(int, int).
- static Fl_RGB_Scaling **scaling_algorithm** ()

  Gets what algorithm is used when resizing a source image to draw it.
- static void scaling_algorithm (Fl_RGB_Scaling algorithm)

  Sets what algorithm is used when resizing a source image to draw it.

Public Attributes inherited from Fl_RGB_Image

- int **alloc_array**

  If non-zero, the object's data array is delete[]'d when deleting the object.
- const uchar ∗ array

  Points to the start of the object's data array.

Static Public Attributes inherited from Fl_Image

- static const int **ERR_FILE_ACCESS** = -2
- static const int **ERR_FORMAT** = -3
- static const int **ERR_MEMORY_ACCESS** = -4
- static const int **ERR_NO_IMAGE** = -1
- static bool **register_images_done** = false

  True after fl_register_images() was called, false before.

Static Protected Member Functions inherited from Fl_Image

- static void **labeltype** (const Fl_Label *lo, int lx, int ly, int lw, int lh, Fl_Align la)
- static void **measure** (const Fl_Label *lo, int &lw, int &lh)

13.73.1 Detailed Description

The Fl_JPEG_Image class supports loading, caching, and drawing of Joint Photographic Experts Group (JPEG) File Interchange Format (JFIF) images.
The class supports grayscale and color (RGB) JPEG image files.

13.73.2 Constructor & Destructor Documentation

13.73.2.1 Fl_JPEG_Image() [1/2]

```
Fl_JPEG_Image::Fl_JPEG_Image (
            const char * filename )
```

The constructor loads the JPEG image from the given jpeg filename.
The inherited destructor frees all memory and server resources that are used by the image.
Use Fl_Image::fail() to check if Fl_JPEG_Image failed to load. fail() returns ERR_FILE_ACCESS if the file could not be opened or read, ERR_FORMAT if the JPEG format could not be decoded, and ERR_NO_IMAGE if the image could not be loaded for another reason. If the image has loaded correctly, w(), h(), and d() should return values greater than zero.

Parameters

in	*filename*	a full path and name pointing to a valid jpeg file.

See also

Fl_JPEG_Image::Fl_JPEG_Image(const char *imagename, const unsigned char *data)

13.73.2.2 Fl_JPEG_Image() [2/2]

```
Fl_JPEG_Image::Fl_JPEG_Image (
            const char * name,
            const unsigned char * data,
            int data_length = -1 )
```

The constructor loads the JPEG image from memory.
Construct an image from a block of memory inside the application. Fluid offers "binary Data" chunks as a great way to add image data into the C++ source code. name_png can be NULL. If a name is given, the image is added to the list of shared images (see: Fl_Shared_Image) and will be available by that name.
The inherited destructor frees all memory and server resources that are used by the image.
Use Fl_Image::fail() to check if Fl_JPEG_Image failed to load. fail() returns ERR_FILE_ACCESS if the file could not be opened or read, ERR_FORMAT if the JPEG format could not be decoded, and ERR_NO_IMAGE if the image could not be loaded for another reason. If the image has loaded correctly, w(), h(), and d() should return values greater than zero.

Parameters

name	A unique name or NULL
data	A pointer to the memory location of the JPEG image
data_length	optional length of `data`. This will protect memory outside of the `data` array from illegal read operations

See also

Fl_JPEG_Image::Fl_JPEG_Image(const char *filename)

Fl_Shared_Image

The documentation for this class was generated from the following files:

- Fl_JPEG_Image.H
- Fl_JPEG_Image.cxx

13.74 Fl_Label Struct Reference

This struct stores all information for a text or mixed graphics label.
`#include <Fl_Widget.H>`

Public Member Functions

- void draw (int, int, int, int, Fl_Align) const

  Draws the label aligned to the given box.
- void measure (int &w, int &h) const

  Measures the size of the label.

Public Attributes

- Fl_Align **align_**

  alignment of label
- Fl_Color **color**

  text color
- Fl_Image * **deimage**

  optional image for a deactivated label
- Fl_Font **font**

  label font used in text
- signed char **h_margin_**

  Spacing between label and the horizontally aligned side of the widget.
- Fl_Image * **image**

  optional image for an active label
- Fl_Fontsize **size**

  size of label font
- uchar **spacing**

  Spacing between an image and the label text.
- uchar type

  type of label.
- signed char **v_margin_**

  Spacing between label and the vertically aligned side of the widget.
- const char * **value**

  label text

13.74.1 Detailed Description

This struct stores all information for a text or mixed graphics label.

Todo There is an aspiration that the Fl_Label type will become a widget by itself. That way we will be avoiding a lot of code duplication by handling labels in a similar fashion to widgets containing text. We also provide an easy interface for very complex labels, containing html or vector graphics. However, this re-factoring is not in place in this release.

13.74.2 Member Function Documentation

13.74.2.1 draw()

```
void Fl_Label::draw (
            int X,
            int Y,
            int W,
            int H,
            Fl_Align align ) const
```

Draws the label aligned to the given box.
Draws a label with arbitrary alignment in an arbitrary box.

13.74.2.2 measure()

```
void Fl_Label::measure (
            int & W,
            int & H ) const
```

Measures the size of the label.

Parameters

in,out	*W,H*	: this is the requested size for the label text plus image; on return, this will contain the size needed to fit the label

13.74.3 Member Data Documentation

13.74.3.1 type

```
uchar Fl_Label::type
```

type of label.

See also

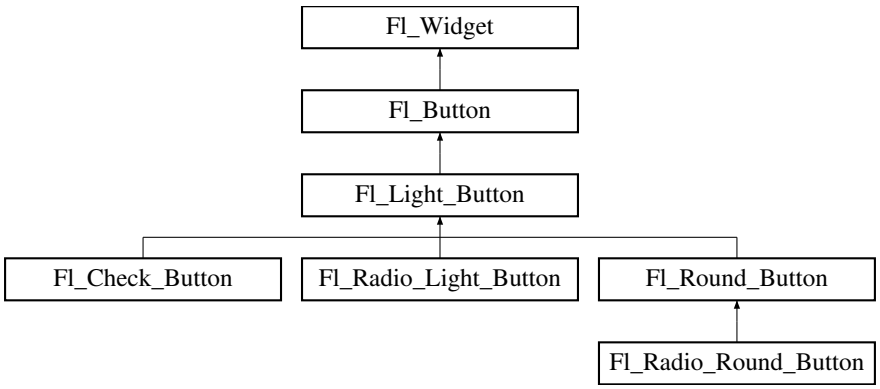
Fl_Labeltype

The documentation for this struct was generated from the following files:

- Fl_Widget.H
- fl_labeltype.cxx

13.75 Fl_Light_Button Class Reference

This subclass displays the "on" state by turning on a light, rather than drawing pushed in.
`#include <Fl_Light_Button.H>`
Inheritance diagram for Fl_Light_Button:



Public Member Functions

- Fl_Light_Button (int x, int y, int w, int h, const char *l=0)

  Creates a new Fl_Light_Button widget using the given position, size, and label string.
- int handle (int) override

  Handles the specified event.

Public Member Functions inherited from Fl_Button

- int clear ()

  Same as `value(0)`.
- uchar compact () const

  Return true if buttons are rendered as compact buttons.
- void compact (uchar v)

  Decide if buttons should be rendered in compact mode.
- Fl_Boxtype down_box () const

  Returns the current down box type, which is drawn when value() is non-zero.
- void down_box (Fl_Boxtype b)

  Sets the down box type.
- Fl_Color **down_color** () const

  (for backwards compatibility)
- void **down_color** (unsigned c)

  (for backwards compatibility)
- Fl_Button (int X, int Y, int W, int H, const char *L=0)

  The constructor creates the button using the given position, size, and label.
- int set ()

  Same as `value(1)`.
- void **setonly** ()

  Turns on this button and turns off all other radio buttons in the group (calling `value(1)` or `set()` does not do this).
- int shortcut () const

  Returns the current shortcut key for the button.
- void **shortcut** (const char *s)

  (for backwards compatibility)
- void shortcut (int s)

  Sets the shortcut key to `s`.
- char **value** () const

  Returns the current value of the button (0 or 1).
- int value (int v)

  Sets the current value of the button.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.

- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.

- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

Gets the image that is used as part of the widget label when in the active state.

- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".

- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void draw () override

  Draws the widget.

Protected Member Functions inherited from Fl_Button

- void **simulate_key_action** ()

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

Draws a box of type t, of color c at the widget's position and size.

- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Static Protected Member Functions inherited from Fl_Button

- static void **key_release_timeout** (void ∗)

Static Protected Attributes inherited from Fl_Button

- static Fl_Widget_Tracker ∗ **key_release_tracker** = 0

13.75.1 Detailed Description

This subclass displays the "on" state by turning on a light, rather than drawing pushed in.
The shape of the "light" is initially set to FL_DOWN_BOX. The color of the light when on is controlled with selection_color(), which defaults to FL_YELLOW.
Buttons generate callbacks when they are clicked by the user. You control exactly when and how by changing the values for type() and when().

Figure 13.27 Fl_Light_Button

13.75.2 Constructor & Destructor Documentation

13.75.2.1 Fl_Light_Button()

```
Fl_Light_Button::Fl_Light_Button (
            int X,
            int Y,
            int W,
            int H,
            const char * l = 0 )
```

Creates a new Fl_Light_Button widget using the given position, size, and label string.
The default box type is `FL_UP_BOX` and the default down box type down_box() is `FL_NO_BOX` (0).
The selection_color() sets the color of the "light". Default is FL_YELLOW.
The default label alignment is `'FL_ALIGN_LEFT|FL_ALIGN_INSIDE'` so the label is drawn inside the button area right of the "light".

Note

Do not change the default box types of Fl_Light_Button. The box types determine how the button is drawn. If you change the down_box() type the drawing behavior is undefined.

13.75.3 Member Function Documentation

13.75.3.1 draw()

```
void Fl_Light_Button::draw ( )  [override], [protected], [virtual]
```

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();                 // calls Fl_Scrollbar::draw()
```

Reimplemented from Fl_Button.

13.75.3.2 handle()

```
int Fl_Light_Button::handle (
            int event )  [override], [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to *override* the behavior of the base class. This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to *filter* event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you *consumed* the event by returning 1 even if you didn't *do* anything with it.

Parameters

in	*event*	the kind of event received

Return values

0	if the event was not used or understood
1	if the event was used and can be deleted

See also

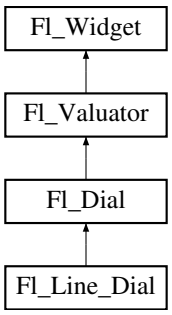
Fl_Event

Reimplemented from Fl_Button.
The documentation for this class was generated from the following files:

- Fl_Light_Button.H
- Fl_Light_Button.cxx

13.76 Fl_Line_Dial Class Reference

Inheritance diagram for Fl_Line_Dial:



Public Member Functions

- **Fl_Line_Dial** (int X, int Y, int W, int H, const char ∗L=0)

Public Member Functions inherited from Fl_Dial

- short angle1 () const

  Sets Or gets the angles used for the minimum and maximum values.
- void **angle1** (short a)

See short angle1() const.

- short **angle2** () const

  See short angle1() const.
- void **angle2** (short a)

  See short angle1() const.
- void **angles** (short a, short b)

  See short angle1() const.
- Fl_Dial (int x, int y, int w, int h, const char *l=0)

  Creates a new Fl_Dial widget using the given position, size, and label string.
- int handle (int) override

  Allow subclasses to handle event based on current position and size.

Public Member Functions inherited from Fl_Valuator

- void **bounds** (double a, double b)

  Sets the minimum (a) and maximum (b) values for the valuator widget.
- double **clamp** (double)

  Clamps the passed value to the valuator range.
- virtual int format (char *)

  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.
- virtual std::string format_str ()

  C++11 API for Fl_Valuator::format(char buffer).*
- double increment (double, int)

  Adds n times the step value to the passed value.
- double maximum () const

  Gets the maximum value for the valuator.
- void maximum (double a)

  Sets the maximum value for the valuator.
- double minimum () const

  Gets the minimum value for the valuator.
- void minimum (double a)

  Sets the minimum value for the valuator.
- void precision (int digits)

  Sets the step value to 1.0 / 10^{digits}.
- void range (double a, double b)

  Sets the minimum and maximum values for the valuator.
- double round (double)

  Round the passed value to the nearest step increment.
- double step () const

  Gets or sets the step value.
- void **step** (double a, int b)

  See double Fl_Valuator::step() const
- void **step** (double s)

  See double Fl_Valuator::step() const.
- void **step** (int a)

  See double Fl_Valuator::step() const
- double value () const

  Gets the floating point(double) value.
- int value (double)

  Sets the current value.
- ~**Fl_Valuator** () override

  Destructor is accessible despite protected constructor.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

Checks if the widget value changed since the last callback.

- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.

- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

Sets width ww and height hh accordingly with the label size.

- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.

- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Dial

- void draw () override

  Draws dial at current position and size.
- void draw (int X, int Y, int W, int H)

  Draws dial at given position and size.
- int handle (int event, int X, int Y, int W, int H)

  Allows subclasses to handle event based on given position and size.

Protected Member Functions inherited from Fl_Valuator

- Fl_Valuator (int X, int Y, int W, int H, const char *L)

  Creates a new Fl_Valuator widget using the given position, size, and label string.
- void **handle_drag** (double newvalue)

  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.
- void **handle_push** ()

  Stores the current value in the previous value.
- void **handle_release** ()

  Called after an FL_WHEN_RELEASE event is received and before the callback.
- int **horizontal** () const

  Tells if the valuator is an FL_HORIZONTAL one.
- double **previous_value** () const

  Gets the previous floating point value before an event changed it.
- void **set_value** (double v)

  Sets the current floating point value.
- double **softclamp** (double)

Clamps the value, but accepts v if the previous value is not already out of range.

- virtual void value_damage ()

  Asks for partial redraw.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

The documentation for this class was generated from the following files:

- Fl_Line_Dial.H
- Fl_Dial.cxx

13.77 Fl_Mac_App_Menu Class Reference

Static Public Member Functions

- static void custom_application_menu_items (const Fl_Menu_Item *m)

  Adds custom menu item(s) to the application menu of the system menu bar.

Static Public Attributes

- static const char * **about**

  Localizable text for the "About xxx" application menu item.
- static const char * **hide**

  Localizable text for the "Hide xxx" application menu item.
- static const char * **hide_others**

  Localizable text for the "Hide Others" application menu item.
- static const char * print

  Localizable text for the "Print Front Window" application menu item.
- static const char * **print_no_titlebar**

  Localizable text for the "Print Front Window" application menu item.
- static const char * **quit**

  Localizable text for the "Quit xxx" application menu item.
- static const char * **services**

  Localizable text for the "Services" application menu item.
- static const char * **show**

  Localizable text for the "Show All" application menu item.
- static const char * **toggle_print_titlebar**

  Localizable text for the "Toggle print titlebar" application menu item.

13.77.1 Member Function Documentation

13.77.1.1 custom_application_menu_items()

```
static void Fl_Mac_App_Menu::custom_application_menu_items (
            const Fl_Menu_Item * m )  [static]
```

Adds custom menu item(s) to the application menu of the system menu bar.
They are positioned after the "Print Front Window / Toggle printing of titlebar" items, or at their place if they were removed with `Fl_Mac_App_Menu::print = ""`.

Parameters

m	zero-ending array of Fl_Menu_Item 's.

13.77.2 Member Data Documentation

13.77.2.1 print

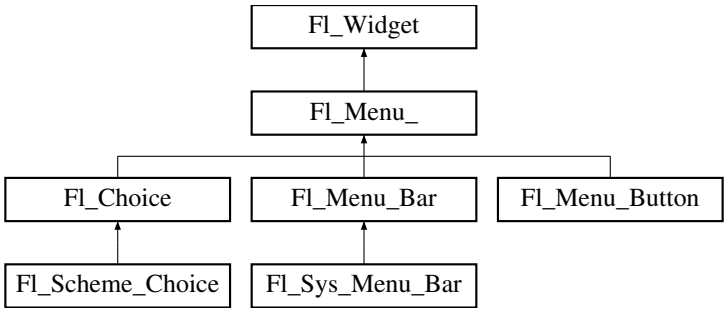
```
const char* Fl_Mac_App_Menu::print  [static]
```

Localizable text for the "Print Front Window" application menu item.
This menu item and next one won't be displayed if Fl_Mac_App_Menu::print is set to an empty string.
The documentation for this class was generated from the following file:

- mac.H

13.78 Fl_Menu_ Class Reference

Base class of all widgets that have a menu in FLTK.
`#include <Fl_Menu_.H>`
Inheritance diagram for Fl_Menu_:



Public Member Functions

- int add (const char ∗)

  This is a Forms (and SGI GL library) compatible add function, it adds many menu items, with '|' separating the menu items, and tab separating the menu item names from an optional shortcut string.
- int add (const char ∗, int shortcut, Fl_Callback ∗, void ∗=0, int=0)

  Adds a new menu item.
- int **add** (const char ∗a, const char ∗b, Fl_Callback ∗c, void ∗d=0, int e=0)

  See int Fl_Menu_::add(const char∗ label, int shortcut, Fl_Callback∗, void ∗user_data=0, int flags=0)
- void clear ()

  Same as menu(NULL), set the array pointer to null, indicating a zero-length menu.
- int clear_submenu (int index)

  Clears the specified submenu pointed to by `index` of all menu items.
- void copy (const Fl_Menu_Item ∗m, void ∗user_data=0)

  Sets the menu array pointer with a copy of m that will be automatically deleted.
- Fl_Boxtype **down_box** () const

  This box type is used to surround the currently-selected items in the menus.
- void down_box (Fl_Boxtype b)

  Sets the box type used to surround the currently-selected items in the menus.
- Fl_Color **down_color** () const

  For back compatibility, same as selection_color()
- void **down_color** (unsigned c)

  For back compatibility, same as selection_color()
- int find_index (const char ∗name) const

  Find the menu item index for a given menu `pathname`, such as "Edit/Copy".
- int find_index (const Fl_Menu_Item ∗item) const

  Find the index into the menu array for a given `item`.
- int find_index (Fl_Callback ∗cb) const

  Find the index into the menu array for a given callback `cb`.
- const Fl_Menu_Item ∗ find_item (const char ∗name)

  Find the menu item for a given menu `pathname`, such as "Edit/Copy".
- const Fl_Menu_Item ∗ find_item (Fl_Callback ∗)

  Find the menu item for the given callback `cb`.
- const Fl_Menu_Item ∗ find_item_with_argument (long)

  Find the menu item for the given user argument `v`.
- const Fl_Menu_Item ∗ find_item_with_user_data (void ∗)

  Find the menu item for the given user data `v`.
- Fl_Menu_ (int, int, int, int, const char ∗=0)

  Creates a new Fl_Menu_ widget using the given position, size, and label string.
- void global ()

  Make the shortcuts for this menu work no matter what window has the focus when you type it.

- int insert (int index, const char *, int shortcut, Fl_Callback *, void *=0, int=0)

  Inserts a new menu item at the specified `index` position.
- int **insert** (int index, const char *a, const char *b, Fl_Callback *c, void *d=0, int e=0)

  See int Fl_Menu_::insert(const char label, int shortcut, Fl_Callback*, void *user_data=0, int flags=0)*
- int item_pathname (char *name, int namelen, const Fl_Menu_Item *finditem=0) const

  Get the menu 'pathname' for the specified menuitem.
- const Fl_Menu_Item * menu () const

  Returns a pointer to the array of Fl_Menu_Items.
- void menu (const Fl_Menu_Item *m)

  Sets the menu array pointer directly.
- Fl_Boxtype menu_box () const

  Get the box type for the menu popup windows.
- void menu_box (Fl_Boxtype b)

  Set the box type for the menu popup windows.
- const Fl_Menu_Item * menu_end ()

  Finishes menu modifications and returns menu().
- int mode (int i) const

  Get the flags of item i.
- void mode (int i, int fl)

  Set the flags of item i.
- const Fl_Menu_Item * mvalue () const

  Return a pointer to the last menu item that was picked.
- const Fl_Menu_Item * picked (const Fl_Menu_Item *)

  When user picks a menu item, call this.
- const Fl_Menu_Item * prev_mvalue () const

  Return a pointer to the menu item that was picked before the current one was picked.
- void remove (int)

  Deletes item `i` from the menu.
- void replace (int, const char *)

  Changes the text of item `i`.
- void **setonly** (Fl_Menu_Item *item)

  Turns the radio item "on" for the menu item and turns "off" adjacent radio items of the same group.
- void **shortcut** (int i, int s)

  Change the shortcut of item `i` to `s`.
- int size () const

  This returns the number of Fl_Menu_Item structures that make up the menu, correctly counting submenus.
- void **size** (int W, int H)
- const Fl_Menu_Item * test_shortcut ()

  Returns the menu item with the entered shortcut (key value).
- const char * text () const

  Returns the title of the last item chosen.
- const char * text (int i) const

  Returns the title of item i.
- Fl_Color textcolor () const

  Get the current color of menu item labels.
- void **textcolor** (Fl_Color c)

  Sets the current color of menu item labels.
- Fl_Font textfont () const

  Gets the current font of menu item labels.
- void textfont (Fl_Font c)

  Sets the current font of menu item labels.

- Fl_Fontsize textsize () const

  Gets the font size of menu item labels.
- void textsize (Fl_Fontsize c)

  Sets the font size of menu item labels.
- int value () const

  Return the index into the menu() of the last item chosen by the user.
- int value (const Fl_Menu_Item ∗)

  Set the value of a menu to the menu item `m`.
- int value (int i)

  Set the value of the menu to index `i`.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ **as_gl_window** () const
- virtual Fl_Group ∗ as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const ∗ **as_group** () const
- virtual Fl_Window ∗ as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ **as_window** () const
- void bind_deimage (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

Gets the current callback function for the widget.

- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.

- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- virtual void draw ()=0

  Draws the widget.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual int handle (int event)

  Handles the specified event.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

Gets the label color.

- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.

- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

Sets the flags used to decide when a callback is called.

- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- int **item_pathname_** (char *name, int namelen, const Fl_Menu_Item *finditem, const Fl_Menu_Item *menu=0) const

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Protected Attributes

- uchar **alloc**
- uchar **down_box_**
- Fl_Boxtype **menu_box_**
- Fl_Color **textcolor_**
- Fl_Font **textfont_**
- Fl_Fontsize **textsize_**

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.78.1 Detailed Description

Base class of all widgets that have a menu in FLTK.
Currently FLTK provides you with Fl_Menu_Button, Fl_Menu_Bar, and Fl_Choice.
The class contains a pointer to an array of structures of type Fl_Menu_Item. The array may either be supplied directly by the user program, or it may be "private": a dynamically allocated array managed by the Fl_Menu_.
When the user clicks a menu item, value() is set to that item and then:

- If the Fl_Menu_Item has a callback set, that callback is invoked with any userdata configured for it. (The Fl_Menu_ widget's callback is NOT invoked.)
- For any Fl_Menu_Items that **don't** have a callback set, the Fl_Menu_ widget's callback is invoked with any userdata configured for it. The callback can determine which item was picked using value(), mvalue(), item_pathname(), etc.

The line spacing between menu items can be controlled with the global setting Fl::menu_linespacing().

See also

Fl_Widget::shortcut_label(int)

13.78.2 Constructor & Destructor Documentation

13.78.2.1 Fl_Menu_()

```
Fl_Menu_::Fl_Menu_ (
            int X,
            int Y,
            int W,
            int H,
            const char * l = 0 )
```

Creates a new Fl_Menu_ widget using the given position, size, and label string.
menu() is initialized to null.

13.78.3 Member Function Documentation

13.78.3.1 add() [1/2]

```
int Fl_Menu_::add (
            const char * str )
```

This is a Forms (and SGI GL library) compatible add function, it adds many menu items, with '|' separating the menu items, and tab separating the menu item names from an optional shortcut string.
The passed string is split at any '|' characters and then add(s,0,0,0,0) is done with each section. This is often useful if you are just using the value, and is compatible with Forms and other GL programs. The section strings use the same special characters as described for the long version of add().
No items must be added to a menu during a callback to the same menu.

Parameters

str	string containing multiple menu labels as described above

Returns

the index into the menu() array, where the entry was added

13.78.3.2 add() [2/2]

```
int Fl_Menu_::add (
            const char * label,
            int shortcut,
            Fl_Callback * callback,
            void * userdata = 0,
            int flags = 0 )
```

Adds a new menu item.

Parameters

in	*label*	The text label for the menu item.
in	*shortcut*	Optional keyboard shortcut that can be an int or string: (FL_CTRL+'a') or "^a". Default 0 if none.
in	*callback*	Optional callback invoked when user clicks the item. Default 0 if none.
in	*userdata*	Optional user data passed as an argument to the callback. Default 0 if none.
in	*flags*	Optional flags that control the type of menu item; see below. Default is 0 for none.

Returns

The index into the menu() array, where the entry was added.

Description

If the menu array was directly set with menu(x), then copy() is done to make a private array.

Since this method can change the internal menu array, any menu item pointers or indices the application may have cached can become stale, and should be recalculated/refreshed.

A menu item's callback must not add() items to its parent menu during the callback.

Due to backwards compatibility and historical restrictions we recommend to use either

- static menu arrays that are not extended during runtime or
- dynamic, extendable menu item arrays that are entirely created by using add() or insert().

This ensures that all menu arrays and strings are copied to internal storage and released when required.

Note

If you create menus from static Fl_Menu_Item arrays and add() or insert() more menu items later, then the menu array is copied to local storage but some local (static) strings may appear to "leak memory". This is a known issue and discouraged usage (see description above) but the impact on memory usage should typically be small.

Detailed Description of Parameters

label

The menu item's label. This argument is required and must not be NULL.

The characters "&", "/", "\", and "_" are treated as special characters in the label string. The "&" character specifies that the following character is an accelerator and will be underlined. The "" character is used to escape the next character in the string. Labels starting with the "_" character cause a divider to be placed after that menu item.

A label of the form "File/Quit" will create the submenu "File" with a menu item called "Quit".

The label string is copied to new memory and can be freed. The other arguments (including the shortcut) are copied into the menu item unchanged.

If an item exists already with that name then it is replaced with this new one. Otherwise this new one is added to the end of the correct menu or submenu. The return value is the offset into the array that the new entry was placed at.

shortcut

The keyboard shortcut for this menu item.

This parameter is optional, and defaults to 0 to indicate no shortcut.

The shortcut can either be a raw integer value (eg. FL_CTRL+'A') or a string (eg. "^c" or "^97").

Raw integer shortcuts can be a combination of keyboard chars (eg. 'A') and optional keyboard modifiers (see Fl::event_state(), e.g. FL_SHIFT, etc). In addition, FL_COMMAND can be used to denote FL_META under Mac OS X and FL_CTRL under other platforms.

String shortcuts can be specified in one of two ways:

```
[#+^]<ascii_value>    e.g. "97", "^97", "+97", "#97"
[#+^]<ascii_char>     e.g. "a", "^a", "+a", "#a"
```

..where <ascii_value> is a decimal value representing an ASCII character (eg. 97 is the ascii code for 'a'), and the optional prefixes enhance the value that follows. Multiple prefixes must appear in the order below.

```
# - Alt
+ - Shift
^ - Control
```

Internally, the text shortcuts are converted to integer values using fl_old_shortcut(const char*).

callback

The callback to invoke when this menu item is selected.

This parameter is optional, and defaults to 0 for no callback.

userdata

The callback's 'user data' that is passed to the callback.

This parameter is optional, and defaults to 0.

flags

These are bit flags to define what kind of menu item this is.

This parameter is optional, and defaults to 0 to define a 'regular' menu item.

These flags can be 'OR'ed together:

```
FL_MENU_INACTIVE     // Deactivate menu item (gray out)
FL_MENU_TOGGLE       // Item is a checkbox toggle (shows checkbox for on/off state)
FL_MENU_VALUE        // The on/off state for checkbox/radio buttons (if set, state is 'on')
FL_MENU_RADIO        // Item is a radio button (one checkbox of many can be on)
FL_MENU_INVISIBLE    // Item will not show up (shortcut will work)
FL_SUBMENU_POINTER   // Indicates user_data() is a pointer to another menu array
FL_SUBMENU           // This item is a submenu to other items
FL_MENU_DIVIDER      // Creates divider line below this item. Also ends a group of radio buttons.
```

All other bits in `'flags'` are reserved and must not be used.

If FL_SUBMENU is set in an item's flags, then actually two items are added:

- the first item is the menu item (submenu title), as expected, and
- the second item is the submenu terminating item with the label and all other members set to 0.

If you add submenus with the 'path' technique, then the corresponding submenu terminators (maybe more than one) are added as well.

Todo Raw integer shortcut needs examples. Dependent on responses to https://www.fltk.org/newsgroups.php?gfltk.coredev+v:10086 and results of STR#2344

13.78.3.3 clear()

```
void Fl_Menu_::clear ( )
```

Same as menu(NULL), set the array pointer to null, indicating a zero-length menu.
Menus must not be cleared during a callback to the same menu.

13.78.3.4 clear_submenu()

```
int Fl_Menu_::clear_submenu (
            int index )
```

Clears the specified submenu pointed to by `index` of all menu items.
This method is useful for clearing a submenu so that it can be re-populated with new items. Example: a "File/Recent Files/..." submenu that shows the last few files that have been opened.
The specified `index` must point to a submenu.
The submenu is cleared with remove(). If the menu array was directly set with menu(x), then copy() is done to make a private array.

Warning

Since this method can change the internal menu array, any menu item pointers or indices the application may have cached can become stale, and should be recalculated/refreshed.

Example:

```
int index = menubar->find_index("File/Recent");    // get index of "File/Recent" submenu
if ( index != -1 ) menubar->clear_submenu(index);  // clear the submenu
menubar->add("File/Recent/Aaa");
menubar->add("File/Recent/Bbb");
[..]
```

Parameters

index	The index of the submenu to be cleared

Returns

0 on success, -1 if the index is out of range or not a submenu

See also

remove(int)

13.78.3.5 copy()

```
void Fl_Menu_::copy (
            const Fl_Menu_Item * m,
            void * ud = 0 )
```

Sets the menu array pointer with a copy of m that will be automatically deleted.
If userdata `ud` is not NULL, then all user data pointers are changed in the menus as well. See void Fl_Menu_::menu(const Fl_Menu_Item* m).

13.78.3.6 down_box()

```
void Fl_Menu_::down_box (
            Fl_Boxtype b )  [inline]
```

Sets the box type used to surround the currently-selected items in the menus.

13.78.3.7 find_index() [1/3]

```
int Fl_Menu_::find_index (
            const char * pathname ) const
```

Find the menu item index for a given menu `pathname`, such as "Edit/Copy".
This method finds a menu item's index position for the given menu pathname, also traversing submenus, but **not** submenu pointers (FL_SUBMENU_POINTER).
To get the menu item pointer for a pathname, use find_item()

Parameters

in	*pathname*	The path and name of the menu item to find

Returns

The index of the matching item, or -1 if not found.

See also

item_pathname()

13.78.3.8 find_index() [2/3]

```
int Fl_Menu_::find_index (
            const Fl_Menu_Item * item ) const
```

Find the index into the menu array for a given `item`.
A way to convert a menu item pointer into an index.
Does **not** handle items that are in submenu pointers (FL_SUBMENU_POINTER).
-1 is returned if the item is not in this menu or is part of an FL_SUBMENU_POINTER submenu.
Current implementation is fast and not expensive.

```
// Convert an index-to-item
int index = 12;
const Fl_Menu_Item *item = mymenu->menu() + index;

// Convert an item-to-index
int index = mymenu->find_index(item);
if ( index == -1 ) { ..error.. }
```

Parameters

in	*item*	The item to be found

Returns

The index of the item, or -1 if not found.

See also

menu()

13.78.3.9 find_index() [3/3]

```
int Fl_Menu_::find_index (
            Fl_Callback * cb ) const
```

Find the index into the menu array for a given callback `cb`.
This method finds a menu item's index position, also traversing submenus, but **not** submenu pointers (FL_↩ SUBMENU_POINTER). This is useful if an application uses internationalisation and a menu item can not be found using its label. This search is also much faster.

Parameters

cb	Find the first item with this callback

Returns

The index of the item with the specific callback, or -1 if not found

See also

find_index(const char*)

13.78.3.10 find_item() [1/2]

```
const Fl_Menu_Item * Fl_Menu_::find_item (
            const char * pathname )
```

Find the menu item for a given menu `pathname`, such as "Edit/Copy".
This method finds a menu item in the menu array, also traversing submenus, but not submenu pointers (FL_SUBMENU_POINTER).
To get the menu item's index, use find_index(const char*)

Example:

```
Fl_Menu_Bar *menubar = new Fl_Menu_Bar(..);
menubar->add("File/&Open");
menubar->add("File/&Save");
menubar->add("Edit/&Copy");
// [..]
Fl_Menu_Item *item;
if ( ( item = (Fl_Menu_Item*)menubar->find_item("File/&Open") ) != NULL ) {
    item->labelcolor(FL_RED);
}
if ( ( item = (Fl_Menu_Item*)menubar->find_item("Edit/&Copy") ) != NULL ) {
    item->labelcolor(FL_GREEN);
}
```

Parameters

pathname	The path and name of the menu item

Returns

The item found, or NULL if not found

See also

find_index(const char*), find_item(Fl_Callback*), item_pathname()

13.78.3.11 find_item() [2/2]

```
const Fl_Menu_Item * Fl_Menu_::find_item (
            Fl_Callback * cb )
```

Find the menu item for the given callback `cb`.
This method finds a menu item in a menu array, also traversing submenus, but not submenu pointers. This is useful if an application uses internationalisation and a menu item can not be found using its label. This search is also much faster.

Parameters

in	*cb*	find the first item with this callback

Returns

The item found, or NULL if not found

See also

find_item(const char*)

13.78.3.12 find_item_with_argument()

```
const Fl_Menu_Item * Fl_Menu_::find_item_with_argument (
            long v )
```

Find the menu item for the given user argument `v`.

Parameters

in	*v*	find the first item with this user argument

Returns

The item found, or NULL if not found

See also

find_item(const char*)

13.78.3.13 find_item_with_user_data()

```
const Fl_Menu_Item * Fl_Menu_::find_item_with_user_data (
            void * v )
```

Find the menu item for the given user data `v`.

Parameters

in	*v*	find the first item with this user data

Returns

The item found, or NULL if not found

See also

find_item(const char*)

13.78.3.14 global()

```
void Fl_Menu_::global ( )
```

Make the shortcuts for this menu work no matter what window has the focus when you type it.
This is done by using Fl::add_handler(). This Fl_Menu_ widget does not have to be visible (ie the window it is in can be hidden, or it does not have to be put in a window at all).
Currently there can be only one global()menu. Setting a new one will replace the old one. There is no way to remove the global() setting (so don't destroy the widget!)

13.78.3.15 insert()

```
int Fl_Menu_::insert (
            int index,
            const char * label,
            int shortcut,
            Fl_Callback * callback,
            void * userdata = 0,
            int flags = 0 )
```

Inserts a new menu item at the specified `index` position.
If `index` is -1, the menu item is appended; same behavior as add().

To properly insert a menu item, `label` must be the name of the item (eg. "Quit"), and not a 'menu pathname' (eg. "File/Quit"). If a menu pathname is specified, the value of `index` is *ignored*, the new item's position defined by the pathname.
For more details, see add(). Except for the `index` parameter, add() has more detailed information on parameters and behavior, and is functionally equivalent.

Parameters

in	*index*	The menu array's index position where the new item is inserted. If -1, behavior is the same as add().
in	*label*	The text label for the menu item. If the label is a menu pathname, `index` is ignored, and the pathname indicates the position of the new item.
in	*shortcut*	Optional keyboard shortcut. Can be an int (FL_CTRL+'a') or a string ("^a"). Default is 0.
in	*callback*	Optional callback invoked when user clicks the item. Default 0 if none.
in	*userdata*	Optional user data passed as an argument to the callback. Default 0 if none.
in	*flags*	Optional flags that control the type of menu item; see add() for more info. Default is 0 for none.

Returns

The index into the menu() array, where the entry was added.

See also

add()

13.78.3.16 item_pathname()

```
int Fl_Menu_::item_pathname (
            char * name,
            int namelen,
            const Fl_Menu_Item * finditem = 0 ) const
```

Get the menu 'pathname' for the specified menuitem.
If finditem==NULL, mvalue() is used (the most recently picked menuitem).
Example:

```
Fl_Menu_Bar *menubar = 0;
void my_menu_callback(Fl_Widget*,void*) {
  char name[80];
  if ( menubar->item_pathname(name, sizeof(name)-1) == 0 ) {   // recently picked item
    if ( strcmp(name, "File/&Open") == 0 ) { .. }              // open invoked
    if ( strcmp(name, "File/&Save") == 0 ) { .. }              // save invoked
    if ( strcmp(name, "Edit/&Copy") == 0 ) { .. }              // copy invoked
  }
}
int main() {
  [..]
  menubar = new Fl_Menu_Bar(..);
  menubar->add("File/&Open",  0, my_menu_callback);
  menubar->add("File/&Save",  0, my_menu_callback);
  menubar->add("Edit/&Copy",  0, my_menu_callback);
  [..]
}
```

Returns

- 0 : OK (name has menuitem's pathname)
- -1 : item not found (name="")
- -2 : 'name' not large enough (name="")

See also

find_item()

13.78.3.17 menu() [1/2]

```
const Fl_Menu_Item * Fl_Menu_::menu ( ) const  [inline]
```

Returns a pointer to the array of Fl_Menu_Items.
This will either be the value passed to menu(value) or the private copy or an internal (temporary) location (see note below).

Note

> **Implementation details - may be changed in the future.** All modifications of the menu array are done by copying the entire menu array to an internal storage for optimization of memory allocations, for instance when using add() or insert(). While this is done, menu() returns the pointer to this internal location. The entire menu will be copied back to private storage when needed, i.e. when **another** Fl_Menu_ is modified. You can force this reallocation after you're done with all menu modifications by calling Fl_Menu_::menu_end() to make sure menu() returns a permanent pointer to private storage (until the menu is modified again). Note also that some menu methods (e.g. Fl_Menu_Button::popup()) call menu_end() internally to ensure a consistent menu array while the menu is open.

See also

> size() – returns the size of the Fl_Menu_Item array.
>
> menu_end() – finish menu modifications (optional)

Example: How to walk the array:

```
for ( int t=0; t<menubar->size(); t++ ) {                // walk array of items
    const Fl_Menu_Item &item = menubar->menu()[t];       // get each item
    fprintf(stderr, "item #%d -- label=%s, value=%s type=%s\n",
        t,
        item.label() ? item.label() : "(Null)",          // menu terminators have NULL labels
        (item.flags & FL_MENU_VALUE) ? "set" : "clear",  // value of toggle or radio items
        (item.flags & FL_SUBMENU) ? "Submenu" : "Item"); // see if item is a submenu or actual item
}
```

13.78.3.18 menu() [2/2]

```
void Fl_Menu_::menu (
            const Fl_Menu_Item * m )
```

Sets the menu array pointer directly.
If the old menu is private it is deleted. NULL is allowed and acts the same as a zero-length menu. If you try to modify the array (with add(), replace(), or remove()) a private copy is automatically done.

13.78.3.19 menu_box() [1/2]

```
Fl_Boxtype Fl_Menu_::menu_box ( ) const  [inline]
```

Get the box type for the menu popup windows.

Returns

> the box type, or FL_NO_BOX if Fl_Menu_::box() is to be used instead

13.78.3.20 menu_box() [2/2]

```
void Fl_Menu_::menu_box (
            Fl_Boxtype b )  [inline]
```

Set the box type for the menu popup windows.
If menu_box set to FL_NO_BOX, the menu window will use Fl_Menu_::box() instead.

Parameters

in	*b*	new box type or FL_NO_BOX

13.78.3.21 menu_end()

```
const Fl_Menu_Item * Fl_Menu_::menu_end ( )
```

Finishes menu modifications and returns menu().
Call menu_end() after using add(), insert(), remove(), or any other methods that may change the menu array if you want to access the menu array anytime later with menu(). This should be called only once after the **last** menu modification for performance reasons.
Does nothing if the menu array is already in a private location.
Some methods like Fl_Menu_Button::popup() call this method before their menu is opened.

Note

After menu changes like add(), insert(), etc. menu() would return a pointer to a temporary internal menu array that may be relocated at unexpected times. This is due to performance considerations and may be changed w/o further notice.

Since

1.4.0

Returns

New Fl_Menu_Item array pointer.

See also

Fl_Menu_::menu()

13.78.3.22 mode() [1/2]

```
int Fl_Menu_::mode (
            int i ) const  [inline]
```

Get the flags of item i.
For a list of the flags, see Fl_Menu_Item.

13.78.3.23 mode() [2/2]

```
void Fl_Menu_::mode (
            int i,
            int fl )  [inline]
```

Set the flags of item i.
For a list of the flags, see Fl_Menu_Item.

13.78.3.24 mvalue()

```
const Fl_Menu_Item * Fl_Menu_::mvalue ( ) const  [inline]
```

Return a pointer to the last menu item that was picked.

13.78.3.25 picked()

```
const Fl_Menu_Item * Fl_Menu_::picked (
            const Fl_Menu_Item * v )
```

When user picks a menu item, call this.
It will do the callback.
Unfortunately this also casts away const for the checkboxes, but this was necessary so non-checkbox menus can really be declared 'const'.

Parameters

in	*v*	The menu item that was picked by the user.

Returns

The same Fl_Menu_Item* that was set (v).

13.78.3.26 prev_mvalue()

```
const Fl_Menu_Item * Fl_Menu_::prev_mvalue ( ) const  [inline]
```

Return a pointer to the menu item that was picked before the current one was picked.
This call gives developers additional details how a user changed a choice in the Fl_Choice widget.

13.78.3.27 remove()

```
void Fl_Menu_::remove (
            int i )
```

Deletes item i from the menu.
If the menu array was directly set with menu(x) then copy() is done to make a private array.
No items must be removed from a menu during a callback to the same menu.

Parameters

i	index into menu array

13.78.3.28 replace()

```
void Fl_Menu_::replace (
            int i,
            const char * str )
```

Changes the text of item i.
This is the only way to get slash into an add()'ed menu item. If the menu array was directly set with menu(x) then copy() is done to make a private array.

Parameters

i	index into menu array
str	new label for menu item at index i

13.78.3.29 size()

```
int Fl_Menu_::size ( ) const
```

This returns the number of Fl_Menu_Item structures that make up the menu, correctly counting submenus.
This includes the "terminator" item at the end. To copy a menu array you need to copy size()*sizeof(Fl_Menu_Item) bytes. If the menu is NULL this returns zero (an empty menu will return 1).

13.78.3.30 test_shortcut()

```
const Fl_Menu_Item * Fl_Menu_::test_shortcut ( )  [inline]
```

Returns the menu item with the entered shortcut (key value).
This searches the complete menu() for a shortcut that matches the entered key value. It must be called for a FL_KEYBOARD or FL_SHORTCUT event.
If a match is found, the menu's callback will be called.

Returns

matched Fl_Menu_Item or NULL.

13.78.3.31 text() [1/2]

```
const char * Fl_Menu_::text ( ) const  [inline]
```

Returns the title of the last item chosen.

13.78.3.32 text() [2/2]

```
const char * Fl_Menu_::text (
            int i ) const  [inline]
```

Returns the title of item i.

13.78.3.33 textcolor()

```
Fl_Color Fl_Menu_::textcolor ( ) const  [inline]
```

Get the current color of menu item labels.

13.78.3.34 textfont() [1/2]

```
Fl_Font Fl_Menu_::textfont ( ) const  [inline]
```

Gets the current font of menu item labels.

13.78.3.35 textfont() [2/2]

```
void Fl_Menu_::textfont (
            Fl_Font c )  [inline]
```

Sets the current font of menu item labels.

13.78.3.36 textsize() [1/2]

```
Fl_Fontsize Fl_Menu_::textsize ( ) const  [inline]
```

Gets the font size of menu item labels.

13.78.3.37 textsize() [2/2]

```
void Fl_Menu_::textsize (
            Fl_Fontsize c )  [inline]
```

Sets the font size of menu item labels.

13.78.3.38 value() [1/3]

```
int Fl_Menu_::value ( ) const
```

Return the index into the menu() of the last item chosen by the user.
The *value* of the menu is the index into the menu() of the last item chosen by the user or -1.
It is `-1` initially (if no item has been chosen) or if the chosen menu item is part of a submenu addressed by an FL_SUBMENU_POINTER.

Note

All menu items are located in a contiguous array of Fl_Menu_Item's unless an item has the FL_SUBMENU↩_POINTER flag which redirects the submenu to an independent submenu array. This submenu array is not counted in the size() of the menu, and menu items in this submenu can't return a valid index into the **main** menu. Therefore menu items that are located in such a submenu return -1 when value() is called. This may be changed in a future version.

You can use mvalue() instead to retrieve the last picked menu item directly.

Returns

Index of the last chosen menu item or -1 (see description).

See also

const Fl_Menu_Item *mvalue()

13.78.3.39 value() [2/3]

```
int Fl_Menu_::value (
            const Fl_Menu_Item * m )
```

Set the value of a menu to the menu item `m`.
The *value* of the menu is the index into the menu() of the last item chosen by the user or -1.
It is `-1` initially (if no item has been chosen) or if the chosen menu item is part of a submenu addressed by an FL_SUBMENU_POINTER.

Note

All menu items are located in a contiguous array of Fl_Menu_Item's unless an item has the FL_SUBMENU↩_POINTER flag which redirects the submenu to an independent submenu array. This submenu array is not counted in the size() of the menu, and menu items in this submenu can't return a valid index into the **main** menu. Therefore menu items that are located in such a submenu return -1 when value() is called. This may be changed in a future version.

The menu item can be any menu item, even one in a detached submenu (see note about FL_SUBMENU_POINTER above).

Parameters

in	*m*	Pointer to any menu item.

Returns

Whether the new value is different than the old one.

Return values

0	The value didn't change.
1	The value was changed.

See also

int value(int)

int value()

const Fl_Menu_Item *mvalue()

13.78.3.40 value() [3/3]

```
int Fl_Menu_::value (
            int i ) [inline]
```

Set the value of the menu to index `i`.
The *value* of the menu is the index into the menu() of the last item chosen by the user or -1.
It is `-1` initially (if no item has been chosen) or if the chosen menu item is part of a submenu addressed by an FL_SUBMENU_POINTER.

Note

All menu items are located in a contiguous array of Fl_Menu_Item's unless an item has the FL_SUBMENU↩_POINTER flag which redirects the submenu to an independent submenu array. This submenu array is not counted in the size() of the menu, and menu items in this submenu can't return a valid index into the **main** menu. Therefore menu items that are located in such a submenu return -1 when value() is called. This may be changed in a future version.

You can set the value as an integer or with a pointer to a menu item. The integer value is restricted to the main menu array (0..size()-1) whereas the menu item can be any menu item, even one in a detached submenu (see note about FL_SUBMENU_POINTER above).

Parameters

in	*i*	Index of the menu item in the main menu array. Values outside 0..size()-1 are ignored (return 0).

Returns

Whether the new value is different than the old one.

Return values

0	The value didn't change.
1	The value was changed.

See also

int value(const Fl_Menu_Item*)

int value()

const Fl_Menu_Item *mvalue()

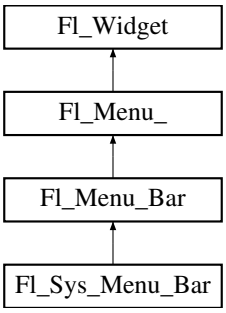
The documentation for this class was generated from the following files:

- Fl_Menu_.H
- Fl_Menu_.cxx
- Fl_Menu_add.cxx
- Fl_Menu_global.cxx

13.79 Fl_Menu_Bar Class Reference

This widget provides a standard menubar interface.

`#include <Fl_Menu_Bar.H>`
Inheritance diagram for Fl_Menu_Bar:



Public Member Functions

- Fl_Menu_Bar (int X, int Y, int W, int H, const char *l=0)

  Creates a new Fl_Menu_Bar widget using the given position, size, and label string.
- int handle (int) override

  Handles the specified event.
- virtual void play_menu (const Fl_Menu_Item *item)

  Opens the 1st level submenu of the menubar corresponding to `item`.
- virtual void update ()

  Updates the menu bar after any change to its items.

Public Member Functions inherited from Fl_Menu_

- int add (const char *)

  This is a Forms (and SGI GL library) compatible add function, it adds many menu items, with '|' separating the menu items, and tab separating the menu item names from an optional shortcut string.
- int add (const char *, int shortcut, Fl_Callback *, void *=0, int=0)

  Adds a new menu item.
- int **add** (const char *a, const char *b, Fl_Callback *c, void *d=0, int e=0)

  See int Fl_Menu_::add(const char label, int shortcut, Fl_Callback*, void *user_data=0, int flags=0)*
- void clear ()

  Same as menu(NULL), set the array pointer to null, indicating a zero-length menu.
- int clear_submenu (int index)

  Clears the specified submenu pointed to by `index` of all menu items.
- void copy (const Fl_Menu_Item *m, void *user_data=0)

  Sets the menu array pointer with a copy of m that will be automatically deleted.
- Fl_Boxtype **down_box** () const

  This box type is used to surround the currently-selected items in the menus.
- void down_box (Fl_Boxtype b)

  Sets the box type used to surround the currently-selected items in the menus.
- Fl_Color **down_color** () const

  For back compatibility, same as selection_color()
- void **down_color** (unsigned c)

  For back compatibility, same as selection_color()
- int find_index (const char *name) const

  Find the menu item index for a given menu `pathname`, such as "Edit/Copy".
- int find_index (const Fl_Menu_Item *item) const

  Find the index into the menu array for a given `item`.
- int find_index (Fl_Callback *cb) const

Find the index into the menu array for a given callback `cb`.

- const Fl_Menu_Item * find_item (const char *name)

  Find the menu item for a given menu `pathname`, such as "Edit/Copy".
- const Fl_Menu_Item * find_item (Fl_Callback *)

  Find the menu item for the given callback `cb`.
- const Fl_Menu_Item * find_item_with_argument (long)

  Find the menu item for the given user argument `v`.
- const Fl_Menu_Item * find_item_with_user_data (void *)

  Find the menu item for the given user data `v`.
- Fl_Menu_ (int, int, int, int, const char *=0)

  Creates a new Fl_Menu_ widget using the given position, size, and label string.
- void global ()

  Make the shortcuts for this menu work no matter what window has the focus when you type it.
- int insert (int index, const char *, int shortcut, Fl_Callback *, void *=0, int=0)

  Inserts a new menu item at the specified `index` position.
- int **insert** (int index, const char *a, const char *b, Fl_Callback *c, void *d=0, int e=0)

  See int Fl_Menu_::insert(const char label, int shortcut, Fl_Callback*, void *user_data=0, int flags=0)*
- int item_pathname (char *name, int namelen, const Fl_Menu_Item *finditem=0) const

  Get the menu 'pathname' for the specified menuitem.
- const Fl_Menu_Item * menu () const

  Returns a pointer to the array of Fl_Menu_Items.
- void menu (const Fl_Menu_Item *m)

  Sets the menu array pointer directly.
- Fl_Boxtype menu_box () const

  Get the box type for the menu popup windows.
- void menu_box (Fl_Boxtype b)

  Set the box type for the menu popup windows.
- const Fl_Menu_Item * menu_end ()

  Finishes menu modifications and returns menu().
- int mode (int i) const

  Get the flags of item i.
- void mode (int i, int fl)

  Set the flags of item i.
- const Fl_Menu_Item * mvalue () const

  Return a pointer to the last menu item that was picked.
- const Fl_Menu_Item * picked (const Fl_Menu_Item *)

  When user picks a menu item, call this.
- const Fl_Menu_Item * prev_mvalue () const

  Return a pointer to the menu item that was picked before the current one was picked.
- void remove (int)

  Deletes item `i` from the menu.
- void replace (int, const char *)

  Changes the text of item `i`.
- void **setonly** (Fl_Menu_Item *item)

  Turns the radio item "on" for the menu item and turns "off" adjacent radio items of the same group.
- void **shortcut** (int i, int s)

  Change the shortcut of item `i` to `s`.
- int size () const

  This returns the number of Fl_Menu_Item structures that make up the menu, correctly counting submenus.
- void **size** (int W, int H)
- const Fl_Menu_Item * test_shortcut ()

Returns the menu item with the entered shortcut (key value).

- const char * text () const

  Returns the title of the last item chosen.
- const char * text (int i) const

  Returns the title of item i.
- Fl_Color textcolor () const

  Get the current color of menu item labels.
- void **textcolor** (Fl_Color c)

  Sets the current color of menu item labels.
- Fl_Font textfont () const

  Gets the current font of menu item labels.
- void textfont (Fl_Font c)

  Sets the current font of menu item labels.
- Fl_Fontsize textsize () const

  Gets the font size of menu item labels.
- void textsize (Fl_Fontsize c)

  Sets the font size of menu item labels.
- int value () const

  Return the index into the menu() of the last item chosen by the user.
- int value (const Fl_Menu_Item *)

  Set the value of a menu to the menu item `m`.
- int value (int i)

  Set the value of the menu to index `i`.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const

- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

Returns non-zero if draw() needs to be called.

- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.

- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

Marks the widget as active without sending events or changing focus.

- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.

- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void draw () override

  Draws the widget.

Protected Member Functions inherited from Fl_Menu_

- int **item_pathname_** (char ∗name, int namelen, const Fl_Menu_Item ∗finditem, const Fl_Menu_Item ∗menu=0) const

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)

Creates a widget at the given position and size.

- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Friends

- class **Fl_Sys_Menu_Bar_Driver**

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Attributes inherited from Fl_Menu_

- uchar **alloc**
- uchar **down_box_**
- Fl_Boxtype **menu_box_**
- Fl_Color **textcolor_**
- Fl_Font **textfont_**
- Fl_Fontsize **textsize_**

13.79.1 Detailed Description

This widget provides a standard menubar interface.
Usually you will put this widget along the top edge of your window. The height of the widget should be 30 for the menu titles to draw correctly with the default font.
The items on the bar and the menus they bring up are defined by a single Fl_Menu_Item array. Because a Fl_Menu_Item array defines a hierarchy, the top level menu defines the items in the menubar, while the submenus define the pull-down menus. Sub-sub menus and lower pop up to the right of the submenus.

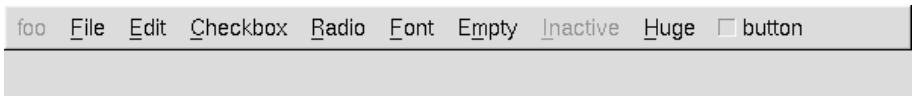


Figure 13.28 menubar

If there is an item in the top menu that is not a title of a submenu, then it acts like a "button" in the menubar. Clicking on it will pick it.
When the user clicks a menu item, value() is set to that item and then:

- The item's callback is done if one has been set; the Fl_Menu_Bar is passed as the Fl_Widget* argument, along with any userdata configured for the callback.
- If the item does not have a callback, the Fl_Menu_Bar's callback is done instead, along with any userdata configured for the callback. The callback can determine which item was picked using value(), mvalue(), item_pathname(), etc.

Submenus will also pop up in response to shortcuts indicated by putting a '&' character in the name field of the menu item. If you put a '&' character in a top-level "button" then the shortcut picks it. The '&' character in submenus is ignored until the menu is popped up.
Typing the shortcut() of any of the menu items will cause callbacks exactly the same as when you pick the item with the mouse.
FLTK triggers an `FL_BEFORE_MENU` event for this widget right before displaying the menu. This event provides an opportunity to update menu item states and activation.

13.79.2 Constructor & Destructor Documentation

13.79.2.1 Fl_Menu_Bar()

```
Fl_Menu_Bar::Fl_Menu_Bar (
            int X,
            int Y,
            int W,
            int H,
            const char * l = 0 )
```

Creates a new Fl_Menu_Bar widget using the given position, size, and label string.
The default boxtype is FL_UP_BOX.
The constructor sets menu() to NULL. See Fl_Menu_ for the methods to set or change the menu.
labelsize(), labelfont(), and labelcolor() are used to control how the menubar items are drawn. They are initialized from the Fl_Menu static variables, but you can change them if desired.
label() is ignored unless you change align() to put it outside the menubar.
The destructor removes the Fl_Menu_Bar widget and all of its menu items.

13.79.3 Member Function Documentation

13.79.3.1 draw()

```
void Fl_Menu_Bar::draw ( )  [override], [protected], [virtual]
```

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.
Override this function to draw your own widgets.

If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();                 // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.
Reimplemented in Fl_Sys_Menu_Bar.

13.79.3.2 handle()

```
int Fl_Menu_Bar::handle (
            int event )  [override], [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to *override* the behavior of the base class. This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to *filter* event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you *consumed* the event by returning 1 even if you didn't *do* anything with it.

Parameters

in	*event*	the kind of event received

Return values

0	if the event was not used or understood
1	if the event was used and can be deleted

See also

Fl_Event

Reimplemented from Fl_Widget.

13.79.3.3 play_menu()

```
void Fl_Menu_Bar::play_menu (
            const Fl_Menu_Item * item )  [virtual]
```

Opens the 1st level submenu of the menubar corresponding to `item`.

Since

1.4.0

Reimplemented in Fl_Sys_Menu_Bar.

13.79.3.4 update()

```
virtual void Fl_Menu_Bar::update ( )  [inline], [virtual]
```

Updates the menu bar after any change to its items.
This is useful when the menu bar can be an Fl_Sys_Menu_Bar object.
Reimplemented in Fl_Sys_Menu_Bar.
The documentation for this class was generated from the following files:

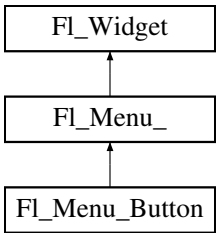
- Fl_Menu_Bar.H
- Fl_Menu_Bar.cxx

13.80 Fl_Menu_Button Class Reference

This is a button that when pushed pops up a menu (or hierarchy of menus) defined by an array of Fl_Menu_Item objects.

`#include <Fl_Menu_Button.H>`

Inheritance diagram for Fl_Menu_Button:



Public Types

- enum popup_buttons {
  POPUP1 = 1 , POPUP2 , POPUP12 , POPUP3 ,
  POPUP13 , POPUP23 , POPUP123 }

  indicate what mouse buttons pop up the menu.

Public Member Functions

- Fl_Menu_Button (int, int, int, int, const char ∗=0)

  Creates a new Fl_Menu_Button widget using the given position, size, and label string.
- int handle (int) override

  Handles the specified event.
- const Fl_Menu_Item ∗ popup ()

  Act exactly as though the user clicked the button or typed the shortcut key.

Public Member Functions inherited from Fl_Menu_

- int add (const char ∗)

  This is a Forms (and SGI GL library) compatible add function, it adds many menu items, with '|' separating the menu items, and tab separating the menu item names from an optional shortcut string.
- int add (const char ∗, int shortcut, Fl_Callback ∗, void ∗=0, int=0)

  Adds a new menu item.
- int **add** (const char ∗a, const char ∗b, Fl_Callback ∗c, void ∗d=0, int e=0)

  See int Fl_Menu_::add(const char∗ label, int shortcut, Fl_Callback∗, void ∗user_data=0, int flags=0)
- void clear ()

  Same as menu(NULL), set the array pointer to null, indicating a zero-length menu.
- int clear_submenu (int index)

  Clears the specified submenu pointed to by `index` of all menu items.
- void copy (const Fl_Menu_Item ∗m, void ∗user_data=0)

  Sets the menu array pointer with a copy of m that will be automatically deleted.
- Fl_Boxtype **down_box** () const

  This box type is used to surround the currently-selected items in the menus.
- void down_box (Fl_Boxtype b)

  Sets the box type used to surround the currently-selected items in the menus.
- Fl_Color **down_color** () const

  For back compatibility, same as selection_color()
- void **down_color** (unsigned c)

For back compatibility, same as selection_color()

- int find_index (const char *name) const

  Find the menu item index for a given menu `pathname`, such as "Edit/Copy".
- int find_index (const Fl_Menu_Item *item) const

  Find the index into the menu array for a given `item`.
- int find_index (Fl_Callback *cb) const

  Find the index into the menu array for a given callback `cb`.
- const Fl_Menu_Item * find_item (const char *name)

  Find the menu item for a given menu `pathname`, such as "Edit/Copy".
- const Fl_Menu_Item * find_item (Fl_Callback *)

  Find the menu item for the given callback `cb`.
- const Fl_Menu_Item * find_item_with_argument (long)

  Find the menu item for the given user argument `v`.
- const Fl_Menu_Item * find_item_with_user_data (void *)

  Find the menu item for the given user data `v`.
- Fl_Menu_ (int, int, int, int, const char *=0)

  Creates a new Fl_Menu_ widget using the given position, size, and label string.
- void global ()

  Make the shortcuts for this menu work no matter what window has the focus when you type it.
- int insert (int index, const char *, int shortcut, Fl_Callback *, void *=0, int=0)

  Inserts a new menu item at the specified `index` position.
- int **insert** (int index, const char *a, const char *b, Fl_Callback *c, void *d=0, int e=0)

  See int Fl_Menu_::insert(const char label, int shortcut, Fl_Callback*, void *user_data=0, int flags=0)*
- int item_pathname (char *name, int namelen, const Fl_Menu_Item *finditem=0) const

  Get the menu 'pathname' for the specified menuitem.
- const Fl_Menu_Item * menu () const

  Returns a pointer to the array of Fl_Menu_Items.
- void menu (const Fl_Menu_Item *m)

  Sets the menu array pointer directly.
- Fl_Boxtype menu_box () const

  Get the box type for the menu popup windows.
- void menu_box (Fl_Boxtype b)

  Set the box type for the menu popup windows.
- const Fl_Menu_Item * menu_end ()

  Finishes menu modifications and returns menu().
- int mode (int i) const

  Get the flags of item i.
- void mode (int i, int fl)

  Set the flags of item i.
- const Fl_Menu_Item * mvalue () const

  Return a pointer to the last menu item that was picked.
- const Fl_Menu_Item * picked (const Fl_Menu_Item *)

  When user picks a menu item, call this.
- const Fl_Menu_Item * prev_mvalue () const

  Return a pointer to the menu item that was picked before the current one was picked.
- void remove (int)

  Deletes item `i` from the menu.
- void replace (int, const char *)

  Changes the text of item `i`.
- void **setonly** (Fl_Menu_Item *item)

  Turns the radio item "on" for the menu item and turns "off" adjacent radio items of the same group.

- void **shortcut** (int i, int s)

  Change the shortcut of item `i` to `s`.
- int size () const

  This returns the number of Fl_Menu_Item structures that make up the menu, correctly counting submenus.
- void **size** (int W, int H)
- const Fl_Menu_Item ∗ test_shortcut ()

  Returns the menu item with the entered shortcut (key value).
- const char ∗ text () const

  Returns the title of the last item chosen.
- const char ∗ text (int i) const

  Returns the title of item i.
- Fl_Color textcolor () const

  Get the current color of menu item labels.
- void **textcolor** (Fl_Color c)

  Sets the current color of menu item labels.
- Fl_Font textfont () const

  Gets the current font of menu item labels.
- void textfont (Fl_Font c)

  Sets the current font of menu item labels.
- Fl_Fontsize textsize () const

  Gets the font size of menu item labels.
- void textsize (Fl_Fontsize c)

  Sets the font size of menu item labels.
- int value () const

  Return the index into the menu() of the last item chosen by the user.
- int value (const Fl_Menu_Item ∗)

  Set the value of a menu to the menu item `m`.
- int value (int i)

  Set the value of the menu to index `i`.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ **as_gl_window** () const

- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

Checks if w is a child of this widget.

- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.

- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

Gets the selection color.

- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.

- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void draw () override

  Draws the widget.

Protected Member Functions inherited from Fl_Menu_

- int **item_pathname_** (char ∗name, int namelen, const Fl_Menu_Item ∗finditem, const Fl_Menu_Item ∗menu=0) const

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

Draws a focus box for the widget at the given position and size.

- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Static Protected Attributes

- static Fl_Menu_Button * **pressed_menu_button_** = NULL

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Attributes inherited from Fl_Menu_

- uchar **alloc**
- uchar **down_box_**
- Fl_Boxtype **menu_box_**
- Fl_Color **textcolor_**
- Fl_Font **textfont_**
- Fl_Fontsize **textsize_**

13.80.1 Detailed Description

This is a button that when pushed pops up a menu (or hierarchy of menus) defined by an array of Fl_Menu_Item objects.

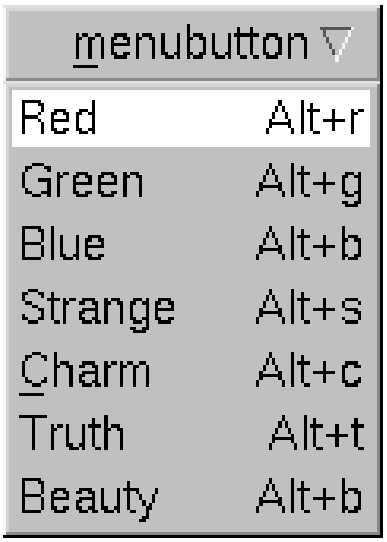


Figure 13.29 menu_button

Normally any mouse button will pop up a menu and it is lined up below the button as shown in the picture. However an Fl_Menu_Button may also control a pop-up menu. This is done by setting the type(). If type() is zero a normal menu button is produced. If it is nonzero then this is a pop-up menu. The bits in type() indicate what mouse buttons pop up the menu (see Fl_Menu_Button::popup_buttons).
The menu will also pop up in response to shortcuts indicated by putting a '&' character in the label().
Typing the shortcut() of any of the menu items will cause callbacks exactly the same as when you pick the item with the mouse. The '&' character in menu item names are only looked at when the menu is popped up, however.
When the user clicks a menu item, value() is set to that item and then:

- The item's callback is done if one has been set; the Fl_Menu_Button is passed as the Fl_Widget* argument, along with any userdata configured for the callback.
- If the item does not have a callback, the Fl_Menu_Button's callback is done instead, along with any userdata configured for it. The callback can determine which item was picked using value(), mvalue(), item_pathname(), etc.

FLTK triggers an `FL_BEFORE_MENU` event for this widget right before displaying the menu. This event provides an opportunity to update menu item states and activation.

13.80.2 Member Enumeration Documentation

13.80.2.1 popup_buttons

`enum Fl_Menu_Button::popup_buttons`
indicate what mouse buttons pop up the menu.
Values for type() used to indicate what mouse buttons pop up the menu. Fl_Menu_Button::POPUP3 is usually what you want.

Enumerator

POPUP1	pops up with the mouse 1st button.
POPUP2	pops up with the mouse 2nd button.
POPUP12	pops up with the mouse 1st or 2nd buttons.
POPUP3	pops up with the mouse 3rd button.
POPUP13	pops up with the mouse 1st or 3rd buttons.
POPUP23	pops up with the mouse 2nd or 3rd buttons.
POPUP123	pops up with any mouse button.

13.80.3 Constructor & Destructor Documentation

13.80.3.1 Fl_Menu_Button()

```
Fl_Menu_Button::Fl_Menu_Button (
            int X,
            int Y,
            int W,
            int H,
            const char * l = 0 )
```

Creates a new Fl_Menu_Button widget using the given position, size, and label string.
The default boxtype is FL_UP_BOX.
The constructor sets menu() to NULL. See Fl_Menu_ for the methods to set or change the menu.

13.80.4 Member Function Documentation

13.80.4.1 draw()

```
void Fl_Menu_Button::draw ( )  [override], [protected], [virtual]
```

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();                 // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

13.80.4.2 handle()

```
int Fl_Menu_Button::handle (
            int event )  [override], [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to *override* the behavior of the base class. This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to *filter* event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you *consumed* the event by returning 1 even if you didn't *do* anything with it.

Parameters

in	*event*	the kind of event received

Return values

0	if the event was not used or understood
1	if the event was used and can be deleted

See also

Fl_Event

Reimplemented from Fl_Widget.

13.80.4.3 popup()

```
const Fl_Menu_Item * Fl_Menu_Button::popup ( )
```

Act exactly as though the user clicked the button or typed the shortcut key.
The menu appears, it waits for the user to pick an item, and if they pick one it sets value() and does the callback or sets changed() as described above. The menu item is returned or NULL if the user dismisses the menu.

Note

Since FLTK 1.4.0 Fl_Menu_::menu_end() is called before the menu pops up to make sure the menu array is located in private storage.

See also

Fl_Menu_::menu_end()

The documentation for this class was generated from the following files:

- Fl_Menu_Button.H
- Fl_Menu_Button.cxx

13.81 Fl_Menu_Item Struct Reference

The Fl_Menu_Item structure defines a single menu item that is used by the Fl_Menu_ class.
`#include <Fl_Menu_Item.H>`

Public Member Functions

- void **activate** ()

  Allows a menu item to be picked.
- int **active** () const

  Gets whether or not the item can be picked.
- int **activevisible** () const

  Returns non 0 if FL_INACTIVE and FL_INVISIBLE are cleared, 0 otherwise.
- int add (const char ∗, int shortcut, Fl_Callback ∗, void ∗=0, int=0)

  Adds a menu item.
- int **add** (const char ∗a, const char ∗b, Fl_Callback ∗c, void ∗d=0, int e=0)

  See int add(const char∗, int shortcut, Fl_Callback∗, void∗, int)
- long argument () const

  Gets the user_data() argument that is sent to the callback function.
- void argument (long v)

Sets the user_data() argument that is sent to the callback function.

- Fl_Callback_p callback () const

  Returns the callback function that is set for the menu item.
- void callback (Fl_Callback *c)

  Sets the menu item's callback function.
- void callback (Fl_Callback *c, void *p)

  Sets the menu item's callback function and userdata() argument.
- void callback (Fl_Callback0 *c)

  Sets the menu item's callback function.
- void callback (Fl_Callback1 *c, long p=0)

  Sets the menu item's callback function and userdata() argument.
- int checkbox () const

  Returns true if a checkbox will be drawn next to this item.
- void **clear** ()

  Turns the check or radio item "off" for the menu item.
- void deactivate ()

  Prevents a menu item from being picked.
- void do_callback (Fl_Widget *o, Fl_Callback_Reason reason=FL_REASON_UNKNOWN) const

  Calls the Fl_Menu_Item item's callback, and provides the Fl_Widget argument.
- void do_callback (Fl_Widget *o, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN) const

  Calls the Fl_Menu_Item item's callback, and provides the Fl_Widget argument.
- void do_callback (Fl_Widget *o, void *arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN) const

  Calls the Fl_Menu_Item item's callback, and provides the Fl_Widget argument.
- void draw (int x, int y, int w, int h, const Fl_Menu_ *, int t=0) const

  Draws the menu item in the bounding box selected or unselected.
- const Fl_Menu_Item * find_shortcut (int *ip=0, const bool require_alt=false) const

  Search only the top level menu for a shortcut.
- Fl_Menu_Item * **first** ()

  Returns the first menu item, same as next(0).
- const Fl_Menu_Item * **first** () const

  Returns the first menu item, same as next(0).
- **FL_DEPRECATED** ("since 1.3 - use setonly(Fl_Menu_Item const* first) instead", void setonly())
- int **headline** () const

  Return non 0 if this menu item is used as a headline.
- void headline (bool yes)

  Set menu item as a headline.
- void **hide** ()

  Hides an item in the menu.
- void image (Fl_Image &image)

  Compatibility API for FLUID, same as image.label(this).
- void image (Fl_Image *image)

  Compatibility API for FLUID, same as image->label(this).
- void image_label (const Fl_Image *image)

  Sets the title (label()) to an icon or image.
- int insert (int, const char *, int, Fl_Callback *, void *=0, int=0)

  Inserts an item at position `index`.
- const char * label () const

  Returns the title (label) of the menu item.
- void label (const char *a)

  Sets the title (label) of the menu item.
- void label (Fl_Labeltype a, const char *b)

Sets the title (label) and the label type of the menu item.

- Fl_Color labelcolor () const

  Gets the menu item's label color.
- void labelcolor (Fl_Color a)

  Sets the menu item's label color.
- Fl_Font labelfont () const

  Gets the menu item's label font.
- void labelfont (Fl_Font a)

  Sets the menu item's label font.
- Fl_Fontsize **labelsize** () const

  Gets the label font pixel size/height.
- void **labelsize** (Fl_Fontsize a)

  Sets the label font pixel size/height.
- Fl_Labeltype labeltype () const

  Returns the menu item's labeltype.
- void labeltype (Fl_Labeltype a)

  Sets the menu item's labeltype.
- int measure (int *h, const Fl_Menu_ *) const

  Measures width of label, including effect of & characters.
- void multi_label (const Fl_Multi_Label *ml)

  Sets the title (label()) and labeltype() to an Fl_Multi_Label.
- Fl_Menu_Item * next (int i=1)

  Advances a pointer by n items through a menu array, skipping the contents of submenus and invisible items.
- const Fl_Menu_Item * next (int=1) const

  Advance a pointer by n items through a menu array.
- const Fl_Menu_Item * popup (int X, int Y, const char *title=0, const Fl_Menu_Item *picked=0, const Fl_Menu_ *=0) const

  This method is called by widgets that want to display menus.
- const Fl_Menu_Item * pulldown (int X, int Y, int W, int H, const Fl_Menu_Item *picked=0, const Fl_Menu_ *=0, const Fl_Menu_Item *title=0, int menubar=0) const

  Pulldown() is similar to popup(), but a rectangle is provided to position the menu.
- int radio () const

  Returns true if this item is a radio item.
- int selectable () const

  Returns non-0 if the menu item can be selected.
- void set ()

  Turns the check or radio item "on" for the menu item.
- void setonly (Fl_Menu_Item const *first)

  Turns the radio item "on" for the menu item and turns "off" adjacent radio items set.
- int **shortcut** () const

  Gets what key combination shortcut will trigger the menu item.
- void shortcut (int s)

  Sets exactly what key combination will trigger the menu item.
- void **show** ()

  Makes an item visible in the menu.
- int size () const

  Size of the menu starting from this menu item.
- int submenu () const

  Returns true if either FL_SUBMENU or FL_SUBMENU_POINTER is on in the flags.
- const Fl_Menu_Item * test_shortcut () const

  This is designed to be called by a widgets handle() method in response to a FL_SHORTCUT event.

- void ∗ **user_data** () const

  Gets the user_data() argument that is sent to the callback function.
- void **user_data** (void ∗v)

  Sets the user_data() argument that is sent to the callback function.
- int value () const

  Returns the current value of the check or radio item.
- void **value** (int v)

  Sets the current value of the check or radio item.
- int **visible** () const

  Gets the visibility of an item.

Public Attributes

- Fl_Callback ∗ **callback_**

  menu item callback
- int **flags**

  menu item flags like FL_MENU_TOGGLE, FL_MENU_RADIO
- Fl_Color **labelcolor_**

  menu item text color
- Fl_Font **labelfont_**

  which font for this menu item text
- Fl_Fontsize **labelsize_**

  size of menu item text
- uchar **labeltype_**

  how the menu item text looks like
- int **shortcut_**

  menu item shortcut
- const char ∗ **text**

  menu item text, returned by label()
- void ∗ **user_data_**

  menu item user_data for the menu's callback

13.81.1 Detailed Description

The Fl_Menu_Item structure defines a single menu item that is used by the Fl_Menu_ class.

```
struct Fl_Menu_Item {
 const char*   text;     // label()
 int           shortcut_;
 Fl_Callback*  callback_;
 void*         user_data_;
 int           flags;
 uchar         labeltype_;
 uchar         labelfont_;
 uchar         labelsize_;
 uchar         labelcolor_;
};

enum { // values for flags:
 FL_MENU_INACTIVE   = 1,          // Deactivate menu item (gray out)
 FL_MENU_TOGGLE     = 2,          // Item is a checkbox toggle (shows checkbox for on/off state)
 FL_MENU_VALUE      = 4,          // The on/off state for checkbox/radio buttons (if set, state is 'on')
 FL_MENU_RADIO      = 8,          // Item is a radio button (one checkbox of many can be on)
 FL_MENU_INVISIBLE  = 0x10,       // Item will not show up (shortcut will work)
 FL_SUBMENU_POINTER = 0x20,       // Indicates user_data() is a pointer to another menu array
 FL_SUBMENU         = 0x40,       // This item is a submenu to other items
 FL_MENU_DIVIDER    = 0x80,       // Creates divider line below this item. Also ends a group of radio
     buttons.
 FL_MENU_HORIZONTAL = 0x100       // reserved, do not use
 FL_MENU_CHATTY     = 0x200
};
```

Note

> **All** other bits in `flags` are reserved for FLTK usage, do not use any bits of the `flags` variable for your own purposes. Even **undocumented** bits can be used for internal purposes in this or any future FLTK version.

Some `flags` bits may be changed during runtime by user code, particularly if you need to change the value of a menu item (ON/OFF) or make it active or inactive. Such changes must be done with caution so they don't affect other (maybe undocumented) bits, i.e. you need to make proper bit operations to set or clear only these particular bits.

Typically menu items are statically defined; for example:

```
Fl_Menu_Item popup[] = {
 {"&alpha",     FL_ALT+'a', the_cb, (void*)1},
 {"&beta",      FL_ALT+'b', the_cb, (void*)2},
 {"gamma",      FL_ALT+'c', the_cb, (void*)3, FL_MENU_DIVIDER},
 {"&strange",   0,          strange_cb},
 {"&charm",     0,          charm_cb},
 {"&truth",     0,          truth_cb},
 {"b&eauty",    0,          beauty_cb},
 {"sub&menu",   0,          0, 0, FL_SUBMENU},
 {"one"},
 {"two"},
 {"three"},
 {0},
 {"inactive", FL_ALT+'i', 0, 0, FL_MENU_INACTIVE|FL_MENU_DIVIDER},
 {"invisible",FL_ALT+'i', 0, 0, FL_MENU_INVISIBLE},
 {"check",    FL_ALT+'i', 0, 0, FL_MENU_TOGGLE|FL_MENU_VALUE},
 {"box",      FL_ALT+'i', 0, 0, FL_MENU_TOGGLE},
 {0}};
```

produces:

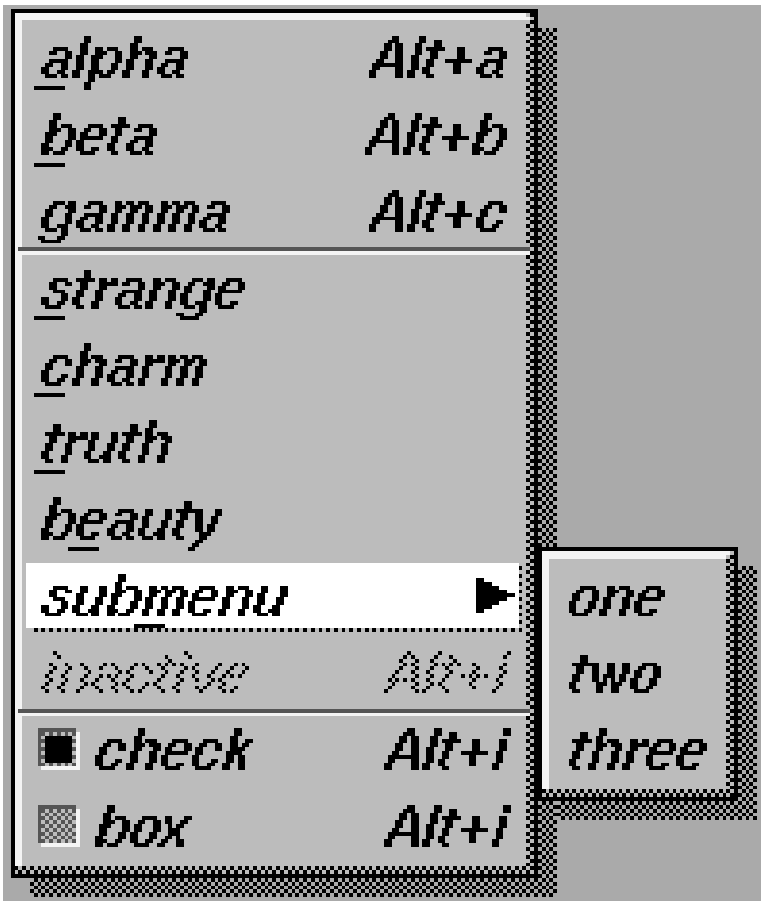


Figure 13.30 menu

A submenu title is identified by the bit FL_SUBMENU in the flags field, and ends with a label() that is NULL. You can nest menus to any depth. A pointer to the first item in the submenu can be treated as an Fl_Menu array itself. It is also possible to make separate submenu arrays with FL_SUBMENU_POINTER flags.

You should use the method functions to access structure members and not access them directly to avoid compatibility problems with future releases of FLTK.

Note

Adding menu items with insert(), add(), or any of its overloaded variants copies the entire menu to internal storage. Using the memory of a static menu array after that would access unused (but not released) memory and thus have no effect.

When a menu item is selected by the user and a callback is set, the callback will be called with FL_REASON_↩ SELECTED. The Fl_Widget pointer is set to the widget that opened the menu, a copy of the menu item's user data. If no menu item callback is set, the callback of the managing widget is called instead.
If the FL_MENU_CHATTY flag is set, the menu item callback may be called for additional reasons. When a menu item is highlighted, the callback is called with FL_REASON_GOT_FOCUS. If it is no longer highlighted, FL_↩ REASON_LOST_FOCUS is sent.

13.81.2 Member Function Documentation

13.81.2.1 add()

```
int Fl_Menu_Item::add (
            const char * mytext,
            int sc,
            Fl_Callback * cb,
            void * data = 0,
            int myflags = 0 )
```

Adds a menu item.
The text is split at '/' characters to automatically produce submenus (actually a totally unnecessary feature as you can now add submenu titles directly by setting FL_SUBMENU in the flags).

Returns

the index into the menu() array, where the entry was added

See also

Fl_Menu_Item::insert(int, const char∗, int, Fl_Callback∗, void∗, int)

13.81.2.2 argument() [1/2]

```
long Fl_Menu_Item::argument ( ) const  [inline]
```

Gets the user_data() argument that is sent to the callback function.
For convenience you can also define the callback as taking a long argument. This method casts the stored user-data() argument to long and returns it as a *long* value.

13.81.2.3 argument() [2/2]

```
void Fl_Menu_Item::argument (
            long v )  [inline]
```

Sets the user_data() argument that is sent to the callback function.
For convenience you can also define the callback as taking a long argument. This method casts the given argument v to void∗ and stores it in the menu item's userdata() member. This may not be portable to some machines.

13.81.2.4 callback() [1/5]

```
Fl_Callback_p Fl_Menu_Item::callback ( ) const  [inline]
```

Returns the callback function that is set for the menu item.
Each item has space for a callback function and an argument for that function. Due to back compatibility, the Fl_Menu_Item itself is not passed to the callback, instead you have to get it by calling ((Fl_Menu_∗)w)->mvalue() where w is the widget argument.

13.81.2.5 callback() [2/5]

```
void Fl_Menu_Item::callback (
            Fl_Callback * c )  [inline]
```

Sets the menu item's callback function.
This method does not set the userdata() argument.

See also

Fl_Callback_p Fl_MenuItem::callback() const

13.81.2.6 callback() [3/5]

```
void Fl_Menu_Item::callback (
            Fl_Callback * c,
            void * p )  [inline]
```

Sets the menu item's callback function and userdata() argument.

See also

Fl_Callback_p Fl_MenuItem::callback() const

13.81.2.7 callback() [4/5]

```
void Fl_Menu_Item::callback (
            Fl_Callback0 * c )  [inline]
```

Sets the menu item's callback function.
This method does not set the userdata() argument.

See also

Fl_Callback_p Fl_MenuItem::callback() const

13.81.2.8 callback() [5/5]

```
void Fl_Menu_Item::callback (
            Fl_Callback1 * c,
            long p = 0 )  [inline]
```

Sets the menu item's callback function and userdata() argument.
The argument `is` cast to void* and stored as the userdata() for the menu item's callback function.

See also

Fl_Callback_p Fl_MenuItem::callback() const

13.81.2.9 checkbox()

```
int Fl_Menu_Item::checkbox ( ) const  [inline]
```

Returns true if a checkbox will be drawn next to this item.
This is true if FL_MENU_TOGGLE or FL_MENU_RADIO is set in the flags.

13.81.2.10 deactivate()

```
void Fl_Menu_Item::deactivate ( )  [inline]
```

Prevents a menu item from being picked.
Note that this will also cause the menu item to appear grayed-out.

13.81.2.11 do_callback() [1/3]

```
void Fl_Menu_Item::do_callback (
            Fl_Widget * o,
            Fl_Callback_Reason reason = FL_REASON_UNKNOWN ) const  [inline]
```

Calls the Fl_Menu_Item item's callback, and provides the Fl_Widget argument.
The callback is called with the stored user_data() as its second argument. You must first check that callback() is non-zero before calling this.

13.81.2.12 do_callback() [2/3]

```
void Fl_Menu_Item::do_callback (
            Fl_Widget * o,
            long arg,
            Fl_Callback_Reason reason = FL_REASON_UNKNOWN ) const  [inline]
```

Calls the Fl_Menu_Item item's callback, and provides the Fl_Widget argument.
This call overrides the callback's second argument with the given value `arg`. long `arg` is cast to void∗ when calling the callback. You must first check that callback() is non-zero before calling this.

13.81.2.13 do_callback() [3/3]

```
void Fl_Menu_Item::do_callback (
            Fl_Widget * o,
            void * arg,
            Fl_Callback_Reason reason = FL_REASON_UNKNOWN ) const  [inline]
```

Calls the Fl_Menu_Item item's callback, and provides the Fl_Widget argument.
This call overrides the callback's second argument with the given value `arg`. You must first check that callback() is non-zero before calling this.

13.81.2.14 draw()

```
void Fl_Menu_Item::draw (
            int x,
            int y,
            int w,
            int h,
            const Fl_Menu_ * m,
            int draw_mode = 0 ) const
```

Draws the menu item in the bounding box selected or unselected.
This does not draw the shortcut: see menuwindow::drawentry().

Parameters

in	*x,y,w,h*	bounding box for the menu item
in	*m*	draw the background, label, and checkbox of this item
in	*draw_mode*	0 = draw unselected, 1 = draw selected, 2 = draw menu title

13.81.2.15 find_shortcut()

```
const Fl_Menu_Item * Fl_Menu_Item::find_shortcut (
            int * ip = 0,
            const bool require_alt = false ) const
```

Search only the top level menu for a shortcut.
Either &x in the label or the shortcut fields are used.
This tests the current event, which must be an FL_KEYBOARD or FL_SHORTCUT, against a shortcut value.

Parameters

ip	returns the index of the item, if `ip` is not NULL.
require_alt	if true: match only if Alt key is pressed.

Returns

found Fl_Menu_Item or NULL

13.81.2.16 headline()

```
void Fl_Menu_Item::headline (
            bool yes )  [inline]
```

Set menu item as a headline.
A menu item can be designated as a headline. Headline items are not selectable, but they are also not grayed out. To visually distinguish them, add indentation and apply bold or italic styling.

13.81.2.17 image() [1/2]

```
void Fl_Menu_Item::image (
            Fl_Image & image )  [inline]
```

Compatibility API for FLUID, same as image.label(this).

Note

This method is intended for internal use by fluid and may not do what you expect.

13.81.2.18 image() [2/2]

```
void Fl_Menu_Item::image (
            Fl_Image * image )  [inline]
```

Compatibility API for FLUID, same as image->label(this).

Note

This method is intended for internal use by fluid and may not do what you expect.

13.81.2.19 image_label()

```
void Fl_Menu_Item::image_label (
            const Fl_Image * image )  [inline]
```

Sets the title (label()) to an icon or image.
This sets the labeltype() to _FL_IMAGE_LABEL (note the leading underscore).

See also

const char* Fl_Menu_Item::label() const

Since

1.4.0

13.81.2.20 insert()

```
int Fl_Menu_Item::insert (
            int index,
            const char * mytext,
            int sc,
            Fl_Callback * cb,
```

```
void * data = 0,
int myflags = 0 )
```

Inserts an item at position `index`.
If `index` is -1, the item is added the same way as Fl_Menu_Item::add().
If 'mytext' contains any un-escaped front slashes (/), it's assumed a menu pathname is being specified, and the value of `index` will be ignored.
In all other aspects, the behavior of insert() is the same as add().

Parameters

in	*index*	insert new items here
in	*mytext*	new label string, details see above
in	*sc*	keyboard shortcut for new item
in	*cb*	callback function for new item
in	*data*	user data for new item
in	*myflags*	menu flags as described in Fl_Menu_Item

Returns

the index into the menu() array, where the entry was added

13.81.2.21 label() [1/3]

```
const char * Fl_Menu_Item::label ( ) const [inline]
```

Returns the title (label) of the menu item.
A NULL here indicates the end of the menu (or of a submenu). A '&' in the item will print an underscore under the next letter, and if the menu is popped up that letter will be a "shortcut" to pick that item. To get a real '&' put two in a row.
The returned value depends on the labeltype() that has been used to store the label.

Returns

A pointer to the label cast to (const char ∗)

Return values

(a)	A pointer to text (const char ∗)
(b)	A pointer to an image (Fl_Image ∗)
(c)	A pointer to a "multi label" (Fl_Multi_Label ∗)
NULL	End of menu or submenu

See also

Fl_Menu_Item::label(const char ∗)

Fl_Menu_Item::label(Fl_Labeltype, const char ∗)

Fl_Menu_Item::image_label(const Fl_Image ∗)

Fl_Menu_Item::multi_label(const Fl_Multi_Label ∗)

Fl_Multi_Label::label(Fl_Menu_Item ∗)

13.81.2.22 label() [2/3]

```
void Fl_Menu_Item::label (
            const char * a ) [inline]
```

Sets the title (label) of the menu item.

Note

This does **not** change the labeltype() for backwards compatibility. Take care to assign the correct labeltype() if you assign different label types to the same menu item sequentially.

The default labeltype() is FL_NORMAL_LABEL.

See also

label(Fl_Labeltype, const char∗)

const char∗ Fl_Menu_Item::label() const

13.81.2.23 label() [3/3]

```
void Fl_Menu_Item::label (
            Fl_Labeltype a,
            const char * b )  [inline]
```

Sets the title (label) and the label type of the menu item.
The default Fl_Labeltype when using Fl_Menu_Item::label(const char∗ a) is FL_NORMAL_LABEL but you can set any other label type, for instance FL_EMBOSSED_LABEL.
Special labeltypes are:

- FL_ICON_LABEL: draws the icon (Fl_Image) associated with the text
- FL_IMAGE_LABEL: draws the image (Fl_Image) associated with the text
- FL_MULTI_LABEL: draws multiple parts side by side, see Fl_Multi_Label

See also

const char∗ Fl_Menu_Item::label() const

13.81.2.24 labelcolor() [1/2]

```
Fl_Color Fl_Menu_Item::labelcolor ( ) const  [inline]
```

Gets the menu item's label color.
This color is passed to the labeltype routine, and is typically the color of the label text. This defaults to FL_BLACK. If this color is not black fltk will **not** use overlay bitplanes to draw the menu - this is so that images put in the menu draw correctly.

13.81.2.25 labelcolor() [2/2]

```
void Fl_Menu_Item::labelcolor (
            Fl_Color a )  [inline]
```

Sets the menu item's label color.

See also

Fl_Color Fl_Menu_Item::labelcolor() const

13.81.2.26 labelfont() [1/2]

```
Fl_Font Fl_Menu_Item::labelfont ( ) const  [inline]
```

Gets the menu item's label font.
Fonts are identified by small 8-bit indexes into a table. See the enumeration list for predefined fonts. The default value is a Helvetica font. The function Fl::set_font() can define new fonts.

13.81.2.27 labelfont() [2/2]

```
void Fl_Menu_Item::labelfont (
            Fl_Font a )  [inline]
```

Sets the menu item's label font.
Fonts are identified by small 8-bit indexes into a table. See the enumeration list for predefined fonts. The default value is a Helvetica font. The function Fl::set_font() can define new fonts.

13.81.2.28 labeltype() [1/2]

```
Fl_Labeltype Fl_Menu_Item::labeltype ( ) const  [inline]
```

Returns the menu item's labeltype.
A labeltype identifies a routine that draws the label of the widget. This can be used for special effects such as emboss, or to use the label() pointer as another form of data such as a bitmap. The value FL_NORMAL_LABEL prints the label as text.

13.81.2.29 labeltype() [2/2]

```
void Fl_Menu_Item::labeltype (
            Fl_Labeltype a )  [inline]
```

Sets the menu item's labeltype.
A labeltype identifies a routine that draws the label of the widget. This can be used for special effects such as emboss, or to use the label() pointer as another form of data such as a bitmap. The value FL_NORMAL_LABEL prints the label as text.

13.81.2.30 measure()

```
int Fl_Menu_Item::measure (
            int * hp,
            const Fl_Menu_ * m ) const
```

Measures width of label, including effect of & characters.
Optionally, can get height if hp is not NULL.

Parameters

out	*hp*	return the height of the label
in	*m*	for this menu item

Returns

width of the label without shortcut text, but including checkbox for radio and check items.

13.81.2.31 multi_label()

```
void Fl_Menu_Item::multi_label (
            const Fl_Multi_Label * ml )  [inline]
```

Sets the title (label()) and labeltype() to an Fl_Multi_Label.
This sets the labeltype() to _FL_MULTI_LABEL (note the leading underscore).

See also

const char* Fl_Menu_Item::label() const

Since

1.4.0

13.81.2.32 next() [1/2]

```
Fl_Menu_Item * Fl_Menu_Item::next (
            int i = 1 )  [inline]
```

Advances a pointer by n items through a menu array, skipping the contents of submenus and invisible items.
There are two calls so that you can advance through const and non-const data.

13.81.2.33 next() [2/2]

```
const Fl_Menu_Item * Fl_Menu_Item::next (
            int n = 1 ) const
```

Advance a pointer by n items through a menu array.
This function skips the contents of submenus, and also invisible items. There are two calls so that you can advance through const and non-const data.

Parameters

in	*n*	advance by n items

Returns

a pointer to the next menu item. If the label() of the returned item is nullptr, the function reached the end of the array and no next item was found.

13.81.2.34 popup()

```
const Fl_Menu_Item * Fl_Menu_Item::popup (
            int X,
            int Y,
            const char * title = 0,
            const Fl_Menu_Item * picked = 0,
            const Fl_Menu_ * menu_button = 0 ) const
```

This method is called by widgets that want to display menus.
The menu stays up until the user picks an item or dismisses it. The selected item (or NULL if none) is returned. *This does not do the callbacks or change the state of check or radio items.*
The menu is positioned so the cursor is centered over the item picked. This will work even if `picked` is in a submenu. If `picked` is zero or not in the menu item table the menu is positioned with the cursor in the top-left corner.

Parameters

in	*X,Y*	the position of the mouse cursor, relative to the window that got the most recent event (usually you can pass Fl::event_x() and Fl::event_y() unchanged here).
in	*title*	a character string title for the menu. If non-zero a small box appears above the menu with the title in it.
in	*picked*	if this pointer is not NULL, the popup menu will appear so that the picked menu is under the mouse pointer.
in	*menu_button*	is a pointer to an Fl_Menu_ from which the color and boxtypes for the menu are pulled. If NULL then defaults are used.

Returns

a pointer to the menu item selected by the user, or NULL

13.81.2.35 pulldown()

```
const Fl_Menu_Item * Fl_Menu_Item::pulldown (
            int X,
            int Y,
            int W,
            int H,
            const Fl_Menu_Item * initial_item = 0,
            const Fl_Menu_ * pbutton = 0,
```

```
const Fl_Menu_Item * title = 0,
int menubar = 0 ) const
```

Pulldown() is similar to popup(), but a rectangle is provided to position the menu.
The menu is made at least `W` wide, and the picked item `initial_item` is centered over the rectangle (like Fl_Choice uses).
If `initial_item` is `NULL` or not found, the menu is aligned just below the rectangle (like a pulldown menu).
The `title` and `menubar` arguments are used internally by the Fl_Menu_Bar widget.

13.81.2.36 radio()

```
int Fl_Menu_Item::radio ( ) const [inline]
```

Returns true if this item is a radio item.
When a radio button is selected all "adjacent" radio buttons are turned off. A set of radio items is delimited by an item that has radio() false, or by an item with FL_MENU_DIVIDER turned on.

13.81.2.37 selectable()

```
int Fl_Menu_Item::selectable ( ) const [inline]
```

Returns non-0 if the menu item can be selected.
Return 0 if the menu is hidden, inactive, or a headline.

13.81.2.38 set()

```
void Fl_Menu_Item::set ( ) [inline]
```

Turns the check or radio item "on" for the menu item.
Note that this does not turn off any adjacent radio items like setonly() does.

13.81.2.39 setonly()

```
void Fl_Menu_Item::setonly (
            Fl_Menu_Item const * first )
```

Turns the radio item "on" for the menu item and turns "off" adjacent radio items set.

Deprecated Please use Fl_Menu_Item::setonly(Fl_Menu_Item const∗ first) instead.

Note

This method is dangerous if radio items are first in the menu. Make sure that `first` is set ciorrectly or use Fl_Menu_::setonly(Fl_Menu_Item∗) instead.

Parameters

in	*first*	start of menu array or NULL (default) if the radio group is not the first item

13.81.2.40 shortcut()

```
void Fl_Menu_Item::shortcut (
            int s ) [inline]
```

Sets exactly what key combination will trigger the menu item.
The value is a logical 'or' of a key and a set of shift flags, for instance FL_ALT+'a' or FL_ALT+FL_F+10 or just 'a'. A value of zero disables the shortcut.
The key can be any value returned by Fl::event_key(), but will usually be an ASCII letter. Use a lower-case letter unless you require the shift key to be held down.
The shift flags can be any set of values accepted by Fl::event_state(). If the bit is on that shift key must be pushed. Meta, Alt, Ctrl, and Shift must be off if they are not in the shift flags (zero for the other bits indicates a "don't care" setting).

13.81.2.41 size()

`int Fl_Menu_Item::size ( ) const`

Size of the menu starting from this menu item.

This method counts all menu items starting with `this` menu item, including all menu items in the same (sub)menu level, all nested submenus, **and** the terminating empty (0) menu item.

It does **not** count menu items referred to by FL_SUBMENU_POINTER menu items (except the single menu item with FL_SUBMENU_POINTER).

All menu items counted are consecutive in memory (one array).

Example:

```
schemechoice = new Fl_Choice(X+125,Y,140,25,"FLTK Scheme");
schemechoice->add("none");
schemechoice->add("plastic");
schemechoice->add("gtk+");
schemechoice->add("gleam");
printf("schemechoice->menu()->size() = %d\n", schemechoice->menu()->size());
```

Output:

schemechoice->menu()->size() = 5

13.81.2.42 submenu()

`int Fl_Menu_Item::submenu ( ) const [inline]`

Returns true if either FL_SUBMENU or FL_SUBMENU_POINTER is on in the flags.

FL_SUBMENU indicates an embedded submenu that goes from the next item through the next one with a NULL label(). FL_SUBMENU_POINTER indicates that user_data() is a pointer to another menu array.

13.81.2.43 test_shortcut()

`const Fl_Menu_Item * Fl_Menu_Item::test_shortcut ( ) const`

This is designed to be called by a widgets handle() method in response to a FL_SHORTCUT event.

If the current event matches one of the items shortcut, that item is returned. If the keystroke does not match any shortcuts then NULL is returned. This only matches the shortcut() fields, not the letters in the title preceeded by '

13.81.2.44 value()

`int Fl_Menu_Item::value ( ) const [inline]`

Returns the current value of the check or radio item.

This is zero (0) if the menu item is not checked and non-zero otherwise.

Since

1.4.0 this method returns 1 if the item is checked but you should not rely on a particular value, only zero or non-zero.

Note

The returned value for a checked menu item was FL_MENU_VALUE (4) before FLTK 1.4.0.

The documentation for this struct was generated from the following files:

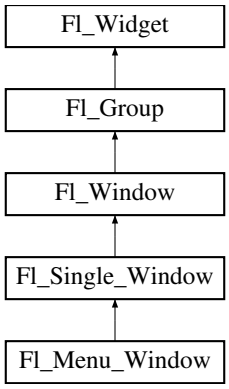
- Fl_Menu_Item.H
- Fl_Menu.cxx
- Fl_Menu_.cxx
- Fl_Menu_add.cxx

13.82 Fl_Menu_Window Class Reference

The Fl_Menu_Window widget is a window type originally used for menus.

`#include <Fl_Menu_Window.H>`

Inheritance diagram for Fl_Menu_Window:



Public Member Functions

- **Fl_Menu_Window** (int W, int H, const char *l=0)

  Creates a new Fl_Menu_Window widget using the given size, and label string.
- **Fl_Menu_Window** (int X, int Y, int W, int H, const char *l=0)

  Creates a new Fl_Menu_Window widget using the given position, size, and label string.
- ~**Fl_Menu_Window** ()

  Destroys the window and all of its children.

Public Member Functions inherited from Fl_Single_Window

- **Fl_Single_Window** (int W, int H, const char *l=0)

  Creates a new Fl_Single_Window widget using the given size, and label (title) string.
- **Fl_Single_Window** (int X, int Y, int W, int H, const char *l=0)

  Creates a new Fl_Single_Window widget using the given position, size, and label (title) string.
- void flush () override

  Same as Fl_Window::flush()
- void **make_current** ()

  Same as Fl_Window::make_current()
- void show () override

  Makes a widget visible.
- void **show** (int argc, char **argv)

  *Same as Fl_Window::show(int argc, char **argv)*

Public Member Functions inherited from Fl_Window

- void allow_expand_outside_parent ()

  Allow this subwindow to expand outside the area of its parent window.
- virtual class Fl_Double_Window * as_double_window ()

  Return non-null if this is an Fl_Double_Window object.
- virtual class Fl_Overlay_Window * as_overlay_window ()

  Return non-null if this is an Fl_Overlay_Window object.
- Fl_Window const * as_window () const override
- Fl_Window * as_window () override

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- unsigned int **border** () const

  Returns whether the window possesses a border.
- void border (int b)

  Sets whether or not the window manager border is around the window.
- void clear_border ()

Fast inline function to turn the window manager border off.

- void clear_modal_states ()

  Clears the "modal" flags and converts a "modal" or "non-modal" window back into a "normal" window.
- void **copy_label** (const char *a)

  Sets the window titlebar label to a copy of a character string.
- void cursor (const Fl_RGB_Image *, int, int)

  Changes the cursor for this window using the provided image as cursor's shape.
- void cursor (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)

  For back compatibility only.
- void cursor (Fl_Cursor)

  Changes the cursor for this window.
- int decorated_h () const

  Returns the window height including any window title bar and any frame added by the window manager.
- int decorated_w () const

  Returns the window width including any frame added by the window manager.
- void default_cursor (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)

  For back compatibility only.
- void default_cursor (Fl_Cursor)

  Sets the default window cursor.
- void **draw_backdrop** ()

  Draw the background image if one is set and is aligned inside.
- Fl_Window (int w, int h, const char *title=0)

  Creates a window from the given width `w`, height `h`, and `title`.
- Fl_Window (int x, int y, int w, int h, const char *title=0)

  Creates a window from the given position (x, y), size (w, h) and title.
- void free_position ()

  Undoes the effect of a previous resize() or show() so that the next time show() is called the window manager is free to position the window.
- void fullscreen ()

  Makes the window completely fill one or more screens, without any window manager border visible.
- unsigned int **fullscreen_active** () const

  Returns non zero if FULLSCREEN flag is set, 0 otherwise.
- void **fullscreen_off** ()

  Turns off any side effects of fullscreen()
- void **fullscreen_off** (int X, int Y, int W, int H)

  Turns off any side effects of fullscreen() and does resize(X,Y,W,H).
- void fullscreen_screens (int top, int bottom, int left, int right)

  Sets which screens should be used when this window is in fullscreen mode.
- uchar get_size_range (int *minw, int *minh, int *maxw=NULL, int *maxh=NULL, int *dw=NULL, int *dh=NULL, int *aspect=NULL) const

  Gets the allowable range to which the user can resize this window.
- int handle (int) override

  Handles the specified event.
- void hide () override

  Removes the window from the screen.
- void **hotspot** (const Fl_Widget &p, int offscreen=0)

  See void Fl_Window::hotspot(int x, int y, int offscreen = 0)
- void **hotspot** (const Fl_Widget *, int offscreen=0)

  See void Fl_Window::hotspot(int x, int y, int offscreen = 0)
- void hotspot (int x, int y, int offscreen=0)

Positions the window so that the mouse is pointing at the given position, or at the center of the given widget, which may be the window itself.

- const void ∗ icon () const

  Gets the current icon window target dependent data.
- void icon (const Fl_RGB_Image ∗)

  Sets or resets a single window icon.
- void icon (const void ∗ic)

  Platform-specific method to set the window icon usable on Windows and X11 only.
- void iconize ()

  Iconifies the window.
- const char ∗ **iconlabel** () const

  See void Fl_Window::iconlabel(const char∗)
- void **iconlabel** (const char ∗)

  Sets the icon label.
- void icons (const Fl_RGB_Image ∗[ ], int)

  Sets the window icons.
- void icons (HICON big_icon, HICON small_icon)

  Sets the window icons using HICON handles (Windows platform only).
- const char ∗ **label** () const

  See void Fl_Window::label(const char∗)
- void **label** (const char ∗)

  Sets the window title bar label.
- void **label** (const char ∗label, const char ∗iconlabel)

  Sets the icon label.
- void make_current ()

  Sets things up so that the drawing functions in <FL/fl_draw.H> will go into this window.
- void maximize ()

  Maximizes a top-level window to its current screen.
- unsigned int **maximize_active** () const

  Returns whether the window is currently maximized.
- unsigned int **menu_window** () const

  Returns true if this window is a menu window.
- unsigned int modal () const

  Returns true if this window is modal.
- unsigned int **non_modal** () const

  Returns true if this window is modal or non-modal.
- fl_uintptr_t os_id () const

  Returns a platform-specific identification of a shown window, or 0 if not shown.
- unsigned int **override** () const

  Returns non zero if OVERRIDE flag is set, 0 otherwise.
- void resize (int X, int Y, int W, int H) override

  Changes the size and position of the window.
- int screen_num () const

  The number of the screen containing the mapped window.
- void screen_num (int screen_num)

  Set the number of the screen where to map the window.
- void set_menu_window ()

  Marks the window as a menu window.
- void set_modal ()

A "modal" window, when shown(), will prevent any events from being delivered to other windows in the same program, and will also remain on top of the other windows (if the X window manager supports the "transient for" property).

- void set_non_modal ()

  A "non-modal" window (terminology borrowed from Microsoft Windows) acts like a modal() one in that it remains on top, but it has no effect on event delivery.
- void **set_override** ()

  Activates the flags NOBORDER|OVERRIDE.
- void set_tooltip_window ()

  Marks the window as a tooltip window.
- const Fl_Image ∗ shape () const

  Returns the image controlling the window shape or NULL.
- void shape (const Fl_Image &b)

  Set the window's shape with an Fl_Image.
- void shape (const Fl_Image ∗img)

  Assigns a non-rectangular shape to the window.
- void show () override

  Puts the window on the screen.
- void show (int argc, char ∗∗argv)

  Puts the window on the screen with show() and parses command-line arguments.
- int shown () const

  Returns non-zero if show() has been called (but not hide()).
- void size_range (int minw, int minh, int maxw=0, int maxh=0, int dw=0, int dh=0, int aspect=0)

  Sets the allowable range to which the user can resize this window.
- unsigned int **tooltip_window** () const

  Returns true if this window is a tooltip window.
- void un_maximize ()

  Returns a previously maximized top-level window to its previous size.
- void wait_for_expose ()

  Waits for the window to be displayed after calling show().
- int **x_root** () const

  Gets the x position of the window on the screen.
- const char ∗ xclass () const

  Returns the xclass for this window, or a default.
- void xclass (const char ∗c)

  Sets the xclass for this window.
- int **y_root** () const

  Gets the y position of the window on the screen.
- virtual ∼Fl_Window ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Group

- Fl_Widget ∗& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget ∗o)

  See void Fl_Group::add(Fl_Widget &w)
- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget ∗const ∗ array () const

Returns a pointer to the internal array of children.

- Fl_Group const * as_group () const override
- Fl_Group * as_group () override

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const

  Returns the n'th child.
- int **children** () const

  Returns how many child widgets the group has.
- void clear ()

  Deletes all child widgets from memory recursively.
- unsigned int clip_children () const

  Returns the current clipping mode.
- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)

  Removes the widget at `index` from the group and deletes it.
- void end ()

  Exactly the same as current(this->parent()).
- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*
- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.
- void **forms_end** ()

  This is for forms compatibility only.
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

  This does insert(w, find(before)).
- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.
- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.
- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.

- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- int horizontal_label_margin ()

Get the spacing between the label and the horizontal edge of the widget.

- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.

- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Public Types inherited from Fl_Window

- typedef struct HICON__ ∗ **HICON**

Static Public Member Functions inherited from Fl_Window

- static Fl_Window ∗ current ()

  Returns the last window that was made current.
- static void **default_callback** (Fl_Window ∗, void ∗v)

  Back compatibility: Sets the default callback v for win to call on close event.
- static void default_icon (const Fl_RGB_Image ∗)

  Sets a single default window icon.
- static void default_icons (const Fl_RGB_Image ∗[ ], int)

  Sets the default window icons.
- static void default_icons (HICON big_icon, HICON small_icon)

  Sets the default window icons (Windows platform only).

- static const char * default_xclass ()

  Returns the default xclass.
- static void default_xclass (const char *)

  Sets the default window xclass.
- static bool is_a_rescale ()

  Returns true when a window is being rescaled.
- static char show_next_window_iconic ()

  Returns the static flag whether the next window should be opened iconified.
- static void show_next_window_iconic (char stat)

  Sets a static flag whether the next window should be opened iconified.

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()

  Returns the currently active group.
- static void current (Fl_Group *g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Window

- void default_size_range ()

  Protected method to calculate the default size range of a window.
- void draw () override

  Draws the widget.
- int force_position () const

  Returns the internal state of the window's `FORCE_POSITION` flag.
- void force_position (int force)

Sets an internal flag that tells FLTK and the window manager to honor position requests.

- void free_icons ()

  Deletes all icons previously attached to the window.
- int is_resizable ()

  Protected method to determine whether a window is resizable.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()

  Returns the internal array of widget sizes and positions.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget *, int)

  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)

  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)

  Allow derived groups to act when a child widget is removed from the group.
- int * sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

Gets the widget flags mask.

- void h (int v)

  Internal use only.

- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.

- void w (int v)

  Internal use only.

- void x (int v)

  Internal use only.

- void y (int v)

  Internal use only.

Static Protected Attributes inherited from Fl_Window

- static Fl_Window ∗ current_

  Stores the last window that was made current.

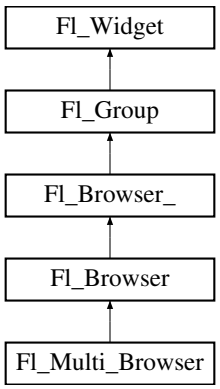
13.82.1 Detailed Description

The Fl_Menu_Window widget is a window type originally used for menus.
Historically this window is drawn in hardware overlay layers of the graphics card if they are available so that the menu doesn't force the rest of the window to redraw (older SGI machines running under IRIX and X11).
With improved hardware and operating systems handling layerd windows and defaulting to double buffering, overlay support was no longer useful and was removed from FLTK in version 1.4.
Fl_Menu_Window has currently no special behavior over Fl_Single_Window. In Fl_Menu.cxx, the local classes `Menu_Window_Basetype` and `Menu_Title_Window` are derived from `Fl_Menu_Window`, implementing additional popup functionality.
The documentation for this class was generated from the following files:

- Fl_Menu_Window.H
- Fl_Menu_Window.cxx

13.83 Fl_Multi_Browser Class Reference

The Fl_Multi_Browser class is a subclass of Fl_Browser which lets the user select any set of the lines.
`#include <Fl_Multi_Browser.H>`
Inheritance diagram for Fl_Multi_Browser:



Public Member Functions

- Fl_Multi_Browser (int X, int Y, int W, int H, const char ∗L=0)

  Creates a new Fl_Multi_Browser widget using the given position, size, and label string.

Public Member Functions inherited from Fl_Browser

- void add (const char *newtext, void *d=0)

  Adds a new line to the end of the browser.
- void bottomline (int line)

  Scrolls the browser so the bottom item in the browser is showing the specified `line`.
- void clear ()

  Removes all the lines in the browser.
- char column_char () const

  Gets the current column separator character.
- void column_char (char c)

  Sets the column separator to c.
- const int * column_widths () const

  Gets the current column width array.
- void column_widths (const int *arr)

  Sets the current array to `arr`.
- void * data (int line) const

  Returns the user data() for specified `line`.
- void data (int line, void *d)

  Sets the user data for specified `line` *to* `d`.
- void display (int line, int val=1)

  For back compatibility.
- int displayed (int line) const

  Returns non-zero if `line` *has been scrolled to a position where it is being displayed.*
- Fl_Browser (int X, int Y, int W, int H, const char *L=0)

  The constructor makes an empty browser.
- char format_char () const

  Gets the current format code prefix character, which by default is '@'.
- void format_char (char c)

  Sets the current format code prefix character to `c`.
- void hide () override

  Hides the entire Fl_Browser widget – opposite of show().
- void hide (int line)

  Makes `line` *invisible, preventing selection by the user.*
- Fl_Image * icon (int line) const

  Returns the icon currently defined for `line`.
- void icon (int line, Fl_Image *icon)

  Set the image icon for `line` *to the value* `icon`.
- void insert (int line, const char *newtext, void *d=0)

  Insert a new entry whose label is `newtext` *above given* `line`*, optional data* `d`.
- void lineposition (int line, Fl_Line_Position pos)

  Updates the browser so that `line` *is shown at position* `pos`.
- int load (const char *filename)

  Clears the browser and reads the file, adding each line from the file to the browser.
- void make_visible (int line)

  Make the item at the specified `line` *visible().*
- void middleline (int line)

  Scrolls the browser so the middle item in the browser is showing the specified `line`.
- void move (int to, int from)

  Line `from` *is removed and reinserted at* `to`.
- void remove (int line)

Remove entry for given `line` number, making the browser one line shorter.

- void remove_icon (int line)

  Removes the icon for `line`.
- void **replace** (int a, const char *b)

  For back compatibility only.
- int select (int line, int val=1)

  Sets the selection state of the item at `line` to the value `val`.
- int selected (int line) const

  Returns 1 if specified `line` is selected, 0 if not.
- void show () override

  Shows the entire Fl_Browser widget – opposite of hide().
- void show (int line)

  Makes `line` visible, and available for selection by user.
- int size () const

  Returns how many lines are in the browser.
- void **size** (int W, int H)
- void swap (int a, int b)

  Swaps two browser lines `a` and `b`.
- const char * text (int line) const

  Returns the label text for the specified `line`.
- void text (int line, const char *newtext)

  Sets the text for the specified `line` to `newtext`.
- Fl_Fontsize **textsize** () const

  Gets the default text size (in pixels) for the lines in the browser.
- void textsize (Fl_Fontsize newSize)

  Sets the default text size (in pixels) for the lines in the browser to `newSize`.
- int topline () const

  Returns the line that is currently visible at the top of the browser.
- void topline (int line)

  Scrolls the browser so the top item in the browser is showing the specified `line`.
- int value () const

  Returns the line number of the currently selected line, or 0 if none selected.
- void value (int line)

  Sets the browser's value(), which selects the specified `line`.
- int visible (int line) const

  Returns non-zero if the specified `line` is visible, 0 if hidden.
- ~**Fl_Browser** ()

  The destructor deletes all list items and destroys the browser.

Public Member Functions inherited from Fl_Browser_

- int deselect (int docallbacks=0)

  Deselects all items in the list and returns 1 if the state changed or 0 if it did not.
- void display (void *item)

  Displays the `item`, scrolling the list as necessary.
- int handle (int event) override

  Handles the `event` within the normal widget bounding box.
- uchar **has_scrollbar** () const

  Returns the current scrollbar mode, see Fl_Browser_::has_scrollbar(uchar)
- void has_scrollbar (uchar mode)

  Sets whether the widget should have scrollbars or not (default Fl_Browser_::BOTH).

- int hposition () const

  Gets the horizontal scroll position of the list as a pixel position `pos`.
- void hposition (int)

  Sets the horizontal scroll position of the list to pixel position `pos`.
- int linespacing () const

  Return the height of additional spacing between browser lines.
- void linespacing (int pixels)

  Add some space between browser lines.
- int position () const
- void position (int pos)
- void **position** (int x, int y)
- void resize (int X, int Y, int W, int H) override

  Repositions and/or resizes the browser.
- void scrollbar_left ()

  Moves the vertical scrollbar to the lefthand side of the list.
- void scrollbar_right ()

  Moves the vertical scrollbar to the righthand side of the list.
- int scrollbar_size () const

  Gets the current size of the scrollbars' troughs, in pixels.
- void scrollbar_size (int newSize)

  Sets the pixel size of the scrollbars' troughs to `newSize`, in pixels.
- int select (void *item, int val=1, int docallbacks=0)

  Sets the selection state of `item` to `val`, and returns 1 if the state changed or 0 if it did not.
- int select_only (void *item, int docallbacks=0)

  Selects `item` and returns 1 if the state changed or 0 if it did not.
- void sort (int flags=0)

  Sort the items in the browser based on `flags`.
- Fl_Color **textcolor** () const

  Gets the default text color for the lines in the browser.
- void **textcolor** (Fl_Color col)

  Sets the default text color for the lines in the browser to color `col`.
- Fl_Font textfont () const

  Gets the default text font for the lines in the browser.
- void **textfont** (Fl_Font font)

  Sets the default text font for the lines in the browser to `font`.
- Fl_Fontsize **textsize** () const

  Gets the default text size (in pixels) for the lines in the browser.
- void **textsize** (Fl_Fontsize newSize)

  Sets the default text size (in pixels) for the lines in the browser to `size`.
- int vposition () const

  Gets the vertical scroll position of the list as a pixel position `pos`.
- void vposition (int pos)

  Sets the vertical scroll position of the list to pixel position `pos`.

Public Member Functions inherited from Fl_Group

- Fl_Widget *& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget *o)

  See void Fl_Group::add(Fl_Widget &w)
- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget *const * array () const

  Returns a pointer to the internal array of children.
- Fl_Group const * as_group () const override
- Fl_Group * as_group () override

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const

  Returns the n'th child.
- int **children** () const

  Returns how many child widgets the group has.
- void clear ()

  Deletes all child widgets from memory recursively.
- unsigned int clip_children () const

  Returns the current clipping mode.
- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)

  Removes the widget at `index` from the group and deletes it.
- void end ()

  Exactly the same as current(this->parent()).
- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*
- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.
- void **forms_end** ()

  This is for forms compatibility only.
- int handle (int) override

  Handles the specified event.
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

  This does insert(w, find(before)).
- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.

- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.
- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- void resize (int, int, int, int) override

  Resizes the Fl_Group widget and all of its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.

- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

Sets the image to use as part of the widget label when in the inactive state.

- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.

- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

Returns if the widget is able to take events.

- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Public Types inherited from Fl_Browser

- enum Fl_Line_Position { **TOP** , **BOTTOM** , **MIDDLE** }

  For internal use only?

Public Types inherited from Fl_Browser_

- enum {
  HORIZONTAL = 1 , VERTICAL = 2 , BOTH = 3 , ALWAYS_ON = 4 ,
  HORIZONTAL_ALWAYS = 5 , VERTICAL_ALWAYS = 6 , BOTH_ALWAYS = 7 }

  Values for has_scrollbar().

Static Public Member Functions inherited from Fl_Group

- static Fl_Group ∗ current ()

  Returns the currently active group.
- static void current (Fl_Group ∗g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget ∗widget, void ∗data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char ∗t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char ∗, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Public Attributes inherited from Fl_Browser_

- Fl_Scrollbar hscrollbar

  Horizontal scrollbar.
- Fl_Scrollbar scrollbar

  Vertical scrollbar.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Browser

- FL_BLINE * _remove (int line)

  Removes the item at the specified `line`.
- const void * **bline_data** (const FL_BLINE *b) const

  Get FL_BLINE data.
- void *& **bline_data** (FL_BLINE *b) const

  Get writable reference to FL_BLINE data.
- char **bline_flags** (const FL_BLINE *b) const

  Get FL_BLINE flags.
- char & **bline_flags** (FL_BLINE *b) const

  Get writable reference to FL_BLINE flags.
- const Fl_Image * **bline_icon** (const FL_BLINE *b) const

  Get FL_BLINE icon.
- Fl_Image *& **bline_icon** (FL_BLINE *b) const

  Get writable reference to FL_BLINE icon.
- short bline_length (const FL_BLINE *b) const

  Get FL_BLINE allocated text buffer size (excl.
- const char * **bline_txt** (const FL_BLINE *b) const

  Get FL_BLINE text.
- char * **bline_txt** (FL_BLINE *b) const

  Get writable reference to FL_BLINE text.
- FL_BLINE * find_line (int line) const

  Returns the item for specified `line`.
- int full_height () const override

  The height of the entire list of all visible() items in pixels.
- int incr_height () const override

  The default 'average' item height (including inter-item spacing) in pixels.
- void insert (int line, FL_BLINE *item)

  Insert specified `item` above `line`.
- void * item_at (int line) const override

  Return the item at specified `line`.
- void item_draw (void *item, int X, int Y, int W, int H) const override

  Draws `item` at the position specified by `X Y W H`.
- void * item_first () const override

  Returns the very first item in the list.
- int item_height (void *item) const override

  Returns height of `item` in pixels.
- void * item_last () const override

  Returns the very last item in the list.
- void * item_next (void *item) const override

  Returns the next item after `item`.
- void * item_prev (void *item) const override

  Returns the previous item before `item`.
- void item_select (void *item, int val) override

  Change the selection state of `item` to the value `val`.
- int item_selected (void *item) const override

  See if `item` is selected.
- void item_swap (void *a, void *b) override

  Swap the items `a` and `b`.
- const char * item_text (void *item) const override

Returns the label text for `item`.

- int item_width (void *item) const override

  Returns width of `item` in pixels.

- int lineno (void *item) const

  Returns line number corresponding to `item`, or zero if not found.

- void swap (FL_BLINE *a, FL_BLINE *b)

  Swap the two items `a` and `b`.

Protected Member Functions inherited from Fl_Browser_

- void bbox (int &X, int &Y, int &W, int &H) const

  Returns the bounding box for the interior of the list's display window, inside the scrollbars.

- void deleting (void *item)

  This method should be used when `item` is being deleted from the list.

- int displayed (void *item) const

  Returns non-zero if `item` has been scrolled to a position where it is being displayed.

- void draw () override

  Draws the list within the normal widget bounding box.

- void * find_item (int ypos)

  This method returns the item under mouse y position `ypos`.

- Fl_Browser_ (int X, int Y, int W, int H, const char *L=0)

  The constructor makes an empty browser.

- virtual int full_width () const

  This method may be provided by the subclass to indicate the full width of the item list, in pixels.

- void inserting (void *a, void *b)

  This method should be used when an item is in the process of being inserted into the list.

- virtual int item_quick_height (void *item) const

  This method may be provided by the subclass to return the height of the `item`, in pixels.

- int leftedge () const

  This method returns the X position of the left edge of the list area after adjusting for the scrollbar and border, if any.

- void new_list ()

  This method should be called when the list data is completely replaced or cleared.

- void redraw_line (void *item)

  This method should be called when the contents of `item` has changed, but not its height.

- void redraw_lines ()

  This method will cause the entire list to be redrawn.

- void replacing (void *a, void *b)

  This method should be used when item `a` is being replaced by item `b`.

- void * selection () const

  Returns the item currently selected, or NULL if there is no selection.

- void swapping (void *a, void *b)

  This method should be used when two items `a` and `b` are being swapped.

- void * **top** () const

  Returns the item that appears at the top of the list.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()

  Returns the internal array of widget sizes and positions.
- void draw () override

  Draws the widget.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget *, int)

  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)

  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)

  Allow derived groups to act when a child widget is removed from the group.
- int * sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

Sets a flag in the flags mask.

- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Static Protected Attributes inherited from Fl_Browser

- static constexpr char **BLINE_NOTDISPLAYED** = 2
- static constexpr char **BLINE_SELECTED** = 1

13.83.1 Detailed Description

The Fl_Multi_Browser class is a subclass of Fl_Browser which lets the user select any set of the lines.

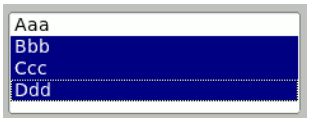


Figure 13.31 Fl_Multi_Browser

The user interface is Macintosh style: clicking an item turns off all the others and selects that one, dragging selects all the items the mouse moves over, and ctrl + click (Cmd+click on the Mac OS platform) toggles the items. Shift + click extends the selection until the clicked item. This is different from how forms did it. Normally the callback is done when the user releases the mouse, but you can change this with when().
See Fl_Browser for methods to add and remove lines from the browser.

13.83.2 Constructor & Destructor Documentation

13.83.2.1 Fl_Multi_Browser()

```
Fl_Multi_Browser::Fl_Multi_Browser (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

Creates a new Fl_Multi_Browser widget using the given position, size, and label string.
The default boxtype is FL_DOWN_BOX. The constructor specializes Fl_Browser() by setting the type to FL_↩ MULTI_BROWSER. The destructor destroys the widget and frees all memory that has been allocated.
The documentation for this class was generated from the following files:

- Fl_Multi_Browser.H
- Fl_Browser.cxx

13.84 Fl_Multi_Label Struct Reference

Allows a mixed text and/or graphics label to be applied to an Fl_Menu_Item or Fl_Widget.
`#include <Fl_Multi_Label.H>`

Public Member Functions

- void label (Fl_Menu_Item *)

  Associate an Fl_Multi_Label with an Fl_Menu_Item.
- void label (Fl_Widget *)

  Associate an Fl_Multi_Label with an Fl_Widget.

Public Attributes

- const char * labela

  Holds the "leftmost" of the two elements in the composite label.
- const char * labelb

  Holds the "rightmost" of the two elements in the composite label.
- uchar typea

  Holds the "type" of labela.
- uchar typeb

  Holds the "type" of labelb.

13.84.1 Detailed Description

Allows a mixed text and/or graphics label to be applied to an Fl_Menu_Item or Fl_Widget.
Most regular FLTK widgets now support the ability to associate both images and text with a label but some special cases, notably the non-widget Fl_Menu_Item objects, do not. Fl_Multi_Label may be used to create menu items that have an icon and text, which would not normally be possible for an Fl_Menu_Item. For example, Fl_Multi_Label is used in the New->Code submenu in fluid, and others.

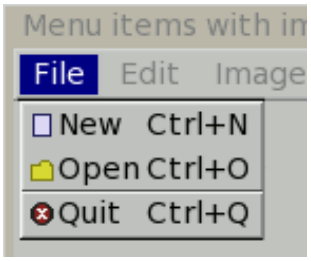


Figure 13.32 Menu items with icons using Fl_Multi_Label

Each Fl_Multi_Label holds two elements, `labela` and `labelb`; each may hold either a text label (const char*) or an image (Fl_Image*). When displayed, `labela` is drawn first and `labelb` is drawn immediately to its right.
More complex labels can be constructed by setting labelb as another Fl_Multi_Label and thus chaining up a series of label elements.
When assigning a label element to one of `labela` or `labelb`, they should be explicitly cast to (const char*) if they are not of that type already.

Note

An Fl_Multi_Label object and all its components (label text, images, chained Fl_Multi_Label's and their linked objects) must exist during the lifetime of the widget or menu item they are assigned to. It is the responsibility of the user's code to release these linked objects if necessary after the widget or menu is deleted.

Example Use: Fl_Menu_Bar

```
Fl_Pixmap *image = new Fl_Pixmap(..);     // image for menu item; any Fl_Image based widget
Fl_Menu_Bar *menu = new Fl_Menu_Bar(..);  // can be any Fl_Menu_ oriented widget (Fl_Choice,
      Fl_Menu_Button..)

// Create a menu item
int i = menu->add("File/New", ..);
Fl_Menu_Item *item = (Fl_Menu_Item*)&(menu->menu()[i]);

// Create a multi label, assign it an image + text
Fl_Multi_Label *ml = new Fl_Multi_Label;

// Left side of label is an image
ml->typea  = FL_IMAGE_LABEL;
ml->labela = (const char*)image;           // any Fl_Image widget: Fl_Pixmap, Fl_PNG_Image, etc..

// Right side of label is label text
ml->typeb  = FL_NORMAL_LABEL;
ml->labelb = item->label();

// Assign the multilabel to the menu item
// ml->label(item); // deprecated since 1.4.0; backwards compatible with 1.3.x
item->label(ml);    // new method since 1.4.0
```

See also

Fl_Label and Fl_Labeltype and examples/howto-menu-with-images.cxx

13.84.2 Member Function Documentation

13.84.2.1 label() [1/2]

```
void Fl_Multi_Label::label (
            Fl_Menu_Item * o )
```

Associate an Fl_Multi_Label with an Fl_Menu_Item.
This uses Fl_Menu_Item::label(Fl_Labeltype a, const char *b) internally to set the *label* and the label type of the menu item, i.e. it stores a *pointer* to the Fl_Multi_Label object (*this*). An existing label (pointer) will be overwritten.
This sets the type of the menu item's label to _FL_MULTI_LABEL - note the leading underscore ('_').
There is no way to use a method like Fl_Widget::copy_label() that transfers ownership of the Fl_Multi_Label and its linked objects (images, text, and chained Fl_Multi_Label's) to the menu item.
The Fl_Multi_Label and all linked images, text labels, or chained Fl_Multi_Label objects must exist during the lifetime of the menu and will not be released when the menu item is destroyed.

Note

The user's code is responsible for releasing the Fl_Multi_Label and all linked objects (images, text, chained Fl_Multi_Label's) after the menu has been deleted. This may cause memory leaks if Fl_Multi_Label is used and reassigned w/o releasing the objects assigned to it.

Deprecated since 1.4.0: please use Fl_Menu_Item::label(Fl_Multi_Label *)

See also

Fl_Menu_Item::label(Fl_Multi_Label *)

13.84.2.2 label() [2/2]

```
void Fl_Multi_Label::label (
            Fl_Widget * o )
```

Associate an Fl_Multi_Label with an Fl_Widget.
This method uses Fl_Widget::label(Fl_Labeltype, const char *) internally to set the *label* of the widget, i.e. it stores a *pointer* to the Fl_Multi_Label object (*this*). An existing label that has been set using Fl_Widget::copy_label() will be released prior to the assignment of the new label.
This sets the type of the widget's label to _FL_MULTI_LABEL - note the leading underscore ('_').
There is no way to use a method like Fl_Widget::copy_label() that transfers ownership of the Fl_Multi_Label and its linked objects (images, text, and chained Fl_Multi_Label's) to the widget.
The Fl_Multi_Label and all linked images, text labels, or chained Fl_Multi_Label objects must exist during the lifetime of the widget and will not be released when the widget is destroyed.

Note

The user's code is responsible for releasing the Fl_Multi_Label and all linked objects (images, text, chained Fl_Multi_Label's) after the widget has been deleted. This may cause memory leaks if Fl_Multi_Label is used and reassigned w/o releasing the objects assigned to it.

13.84.3 Member Data Documentation

13.84.3.1 labela

```
const char* Fl_Multi_Label::labela
```

Holds the "leftmost" of the two elements in the composite label.
Typically this would be assigned either a text string (const char*), a (Fl_Image*) or a (Fl_Multi_Label*).

13.84.3.2 labelb

```
const char* Fl_Multi_Label::labelb
```

Holds the "rightmost" of the two elements in the composite label.
Typically this would be assigned either a text string (const char*), a (Fl_Image*) or a (Fl_Multi_Label*).

13.84.3.3 typea

```
uchar Fl_Multi_Label::typea
```

Holds the "type" of labela.
Typically this is set to FL_NORMAL_LABEL for a text label, FL_IMAGE_LABEL for an image (based on Fl_image) or FL_MULTI_LABEL if "chaining" multiple Fl_Multi_Label elements together.

13.84.3.4 typeb

```
uchar Fl_Multi_Label::typeb
```

Holds the "type" of labelb.
Typically this is set to FL_NORMAL_LABEL for a text label, FL_IMAGE_LABEL for an image (based on Fl_image) or FL_MULTI_LABEL if "chaining" multiple Fl_Multi_Label elements together.
The documentation for this struct was generated from the following files:

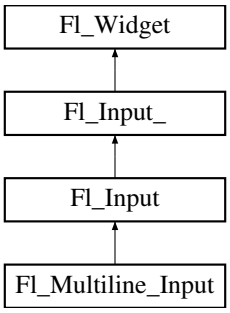
- Fl_Multi_Label.H
- Fl_Multi_Label.cxx

13.85 Fl_Multiline_Input Class Reference

This input field displays '\n' characters as new lines rather than ^J, and accepts the Return, Tab, and up and down arrow keys.

```
#include <Fl_Multiline_Input.H>
```

Inheritance diagram for Fl_Multiline_Input:



Public Member Functions

- Fl_Multiline_Input (int X, int Y, int W, int H, const char *l=0)

  Creates a new Fl_Multiline_Input widget using the given position, size, and label string.

Public Member Functions inherited from Fl_Input

- Fl_Input (int, int, int, int, const char *=0)

  Creates a new Fl_Input widget using the given position, size, and label string.
- int handle (int) override

  Handles the specified event.

Public Member Functions inherited from Fl_Input_

- int append (const char *t, int l=0, char keep_selection=0)

  Append text at the end.
- bool can_redo () const

  Check if there is a redo action available.
- bool can_undo () const

  Check if the last operation can be undone.

- int copy (int clipboard)

  Put the current selection into the clipboard.
- int copy_cuts ()

  Copies the yank buffer to the clipboard.
- Fl_Color cursor_color () const

  Gets the color of the cursor.
- void cursor_color (Fl_Color n)

  Sets the color of the cursor.
- int cut ()

  Deletes the current selection.
- int cut (int a, int b)

  Deletes all characters between index `a` and `b`.
- int cut (int n)

  Deletes the next `n` bytes rounded to characters before or after the cursor.
- double dvalue () const

  Returns the widget text interpreted as a floating point number.
- Fl_Input_ (int, int, int, int, const char *=0)

  Creates a new Fl_Input_ widget.
- unsigned int index (int i) const

  Returns the character at index `i`.
- int input_type () const

  Gets the input field type.
- void input_type (int t)

  Sets the input field type.
- int insert (const char *t, int l=0)

  Inserts text at the cursor position.
- int insert_position () const

  Gets the position of the text cursor.
- int insert_position (int p)

  Sets the cursor position and mark.
- int insert_position (int p, int m)

  Sets the index for the cursor and mark.
- int ivalue () const

  Returns the widget text interpreted as a signed integer.
- int mark () const

  Gets the current selection mark.
- int mark (int m)

  Sets the current selection mark.
- int maximum_size () const

  Gets the maximum length of the input field in characters.
- void maximum_size (int m)

  Sets the maximum length of the input field in characters.
- const char * placeholder () const

  Get the placeholder text.
- void placeholder (const char *text)

  Set the placeholder text.
- int position () const
- int position (int p)
- int position (int p, int m)
- int readonly () const

  Gets the read-only state of the input field.

- void readonly (int b)

  Sets the read-only state of the input field.
- int redo ()

  Redo previous undo operation.
- int replace (int b, int e, const char *text, int ilen=0)

  Deletes text from `b` to `e` and inserts the new string `text`.
- void resize (int, int, int, int) override

  Changes the size of the widget.
- int shortcut () const

  Return the shortcut key associated with this widget.
- void shortcut (int s)

  Sets the shortcut key associated with this widget.
- int size () const

  Returns the number of bytes in value().
- void size (int W, int H)

  Sets the width and height of this widget.
- int static_value (const char *)

  Changes the widget text.
- int static_value (const char *, int)

  Changes the widget text.
- int tab_nav () const

  Gets whether the Tab key causes focus navigation in multiline input fields or not.
- void tab_nav (int val)

  Sets whether the Tab key does focus navigation, or inserts tab characters into Fl_Multiline_Input.
- Fl_Color textcolor () const

  Gets the color of the text in the input field.
- void textcolor (Fl_Color n)

  Sets the color of the text in the input field.
- Fl_Font textfont () const

  Gets the font of the text in the input field.
- void textfont (Fl_Font s)

  Sets the font of the text in the input field.
- Fl_Fontsize textsize () const

  Gets the size of the text in the input field.
- void textsize (Fl_Fontsize s)

  Sets the size of the text in the input field.
- int undo ()

  Undoes previous changes to the text buffer.
- const char * value () const

  Returns the text displayed in the widget.
- int value (const char *)

  Changes the widget text.
- int value (const char *, int)

  Changes the widget text.
- int value (double value)

  Changes the widget text to a floating point number ("%g").
- int value (int value)

  Changes the widget text to a signed integer number.
- int wrap () const

  Gets the word wrapping state of the input field.
- void wrap (int b)

  Sets the word wrapping state of the input field.
- ~Fl_Input_ ()

  Destroys the widget.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

Checks if the widget value changed since the last callback.

- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.

- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

Sets width ww and height hh accordingly with the label size.

- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.

- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Static Public Attributes inherited from Fl_Input

- static const char * **copy_menu_text** = "Copy"

  [this text may be customized at run-time]
- static const char * **cut_menu_text** = "Cut"

  [this text may be customized at run-time]
- static const char * **paste_menu_text** = "Paste"

  [this text may be customized at run-time]

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Input

- void draw () override

  Draws the widget.
- int handle_key ()

  Handles a keystroke.
- int handle_rmb ()

  Handle right mouse button down events.

Protected Member Functions inherited from Fl_Input_

- int apply_undo ()

  Apply the current undo/redo operation.
- void drawtext (int, int, int, int)

  Draws the text in the passed bounding box.
- void drawtext (int, int, int, int, bool draw_active)

  Draws the text in the passed bounding box.
- void handle_mouse (int, int, int, int, int keepmark=0)

  Handles mouse clicks and mouse moves.
- int handletext (int e, int, int, int, int)

  Handles all kinds of text field related events.
- int line_end (int i) const

  Finds the end of a line.
- int line_start (int i) const

  Finds the start of a line.
- int **linesPerPage** ()
- void **maybe_do_callback** (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

- int up_down_position (int, int keepmark=0)

  Moves the cursor to the column given by `up_down_pos`.
- int word_end (int i) const

  Finds the end of a word.
- int word_start (int i) const

  Finds the start of a word.
- int **xscroll** () const
- int **yscroll** () const
- void **yscroll** (int yOffset)

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

13.85.1 Detailed Description

This input field displays '\n' characters as new lines rather than ^J, and accepts the Return, Tab, and up and down arrow keys.
This is for editing multiline text.
This is far from the nirvana of text editors, and is probably only good for small bits of text, 10 lines at most. Note that this widget does not support scrollbars or per-character color control.
If you are presenting large amounts of text and need scrollbars or full color control of characters, you probably want Fl_Text_Editor instead.
In FLTK 1.3.x, the default behavior of the 'Tab' key was changed to support consistent focus navigation. To get the older FLTK 1.1.x behavior, set Fl_Input_::tab_nav() to 0. Newer programs should consider using Fl_Text_Editor.

13.85.2 Constructor & Destructor Documentation

13.85.2.1 Fl_Multiline_Input()

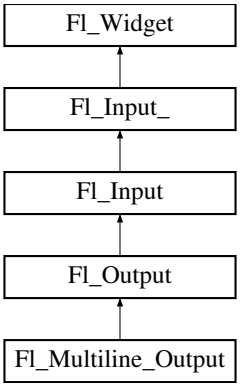
```
Fl_Multiline_Input::Fl_Multiline_Input (
            int X,
            int Y,
            int W,
            int H,
            const char * l = 0 )
```

Creates a new Fl_Multiline_Input widget using the given position, size, and label string.
The default boxtype is FL_DOWN_BOX.
Inherited destructor destroys the widget and any value associated with it.
The documentation for this class was generated from the following files:

- Fl_Multiline_Input.H
- Fl_Input.cxx

13.86 Fl_Multiline_Output Class Reference

This widget is a subclass of Fl_Output that displays multiple lines of text.
`#include <Fl_Multiline_Output.H>`
Inheritance diagram for Fl_Multiline_Output:



Public Member Functions

- Fl_Multiline_Output (int X, int Y, int W, int H, const char *l=0)

  Creates a new Fl_Multiline_Output widget using the given position, size, and label string.

Public Member Functions inherited from Fl_Output

- Fl_Output (int X, int Y, int W, int H, const char *l=0)

  Creates a new Fl_Output widget using the given position, size, and label string.

Public Member Functions inherited from Fl_Input

- Fl_Input (int, int, int, int, const char *=0)

  Creates a new Fl_Input widget using the given position, size, and label string.
- int handle (int) override

  Handles the specified event.

Public Member Functions inherited from Fl_Input_

- int append (const char *t, int l=0, char keep_selection=0)

  Append text at the end.
- bool can_redo () const

  Check if there is a redo action available.
- bool can_undo () const

  Check if the last operation can be undone.
- int copy (int clipboard)

  Put the current selection into the clipboard.
- int copy_cuts ()

  Copies the yank buffer to the clipboard.
- Fl_Color cursor_color () const

  Gets the color of the cursor.
- void cursor_color (Fl_Color n)

  Sets the color of the cursor.
- int cut ()

  Deletes the current selection.
- int cut (int a, int b)

  Deletes all characters between index `a` and `b`.
- int cut (int n)

  Deletes the next `n` bytes rounded to characters before or after the cursor.
- double dvalue () const

  Returns the widget text interpreted as a floating point number.
- Fl_Input_ (int, int, int, int, const char *=0)

  Creates a new Fl_Input_ widget.
- unsigned int index (int i) const

  Returns the character at index `i`.
- int input_type () const

  Gets the input field type.
- void input_type (int t)

  Sets the input field type.
- int insert (const char *t, int l=0)

  Inserts text at the cursor position.
- int insert_position () const

  Gets the position of the text cursor.
- int insert_position (int p)

  Sets the cursor position and mark.
- int insert_position (int p, int m)

  Sets the index for the cursor and mark.
- int ivalue () const

  Returns the widget text interpreted as a signed integer.
- int mark () const

  Gets the current selection mark.
- int mark (int m)

Sets the current selection mark.

- int maximum_size () const

  Gets the maximum length of the input field in characters.
- void maximum_size (int m)

  Sets the maximum length of the input field in characters.
- const char * placeholder () const

  Get the placeholder text.
- void placeholder (const char *text)

  Set the placeholder text.
- int position () const
- int position (int p)
- int position (int p, int m)
- int readonly () const

  Gets the read-only state of the input field.
- void readonly (int b)

  Sets the read-only state of the input field.
- int redo ()

  Redo previous undo operation.
- int replace (int b, int e, const char *text, int ilen=0)

  Deletes text from `b` to `e` and inserts the new string `text`.
- void resize (int, int, int, int) override

  Changes the size of the widget.
- int shortcut () const

  Return the shortcut key associated with this widget.
- void shortcut (int s)

  Sets the shortcut key associated with this widget.
- int size () const

  Returns the number of bytes in value().
- void size (int W, int H)

  Sets the width and height of this widget.
- int static_value (const char *)

  Changes the widget text.
- int static_value (const char *, int)

  Changes the widget text.
- int tab_nav () const

  Gets whether the Tab key causes focus navigation in multiline input fields or not.
- void tab_nav (int val)

  Sets whether the Tab key does focus navigation, or inserts tab characters into Fl_Multiline_Input.
- Fl_Color textcolor () const

  Gets the color of the text in the input field.
- void textcolor (Fl_Color n)

  Sets the color of the text in the input field.
- Fl_Font textfont () const

  Gets the font of the text in the input field.
- void textfont (Fl_Font s)

  Sets the font of the text in the input field.
- Fl_Fontsize textsize () const

  Gets the size of the text in the input field.
- void textsize (Fl_Fontsize s)

  Sets the size of the text in the input field.
- int undo ()

Undoes previous changes to the text buffer.

- const char * value () const

  Returns the text displayed in the widget.
- int value (const char *)

  Changes the widget text.
- int value (const char *, int)

  Changes the widget text.
- int value (double value)

  Changes the widget text to a floating point number ("%g").
- int value (int value)

  Changes the widget text to a signed integer number.
- int wrap () const

  Gets the word wrapping state of the input field.
- void wrap (int b)

  Sets the word wrapping state of the input field.
- ~Fl_Input_ ()

  Destroys the widget.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.

- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

Internal use only.

- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.

- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

Enables keyboard focus navigation with this widget.

- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.

- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Static Public Attributes inherited from Fl_Input

- static const char * **copy_menu_text** = "Copy"

  [this text may be customized at run-time]
- static const char * **cut_menu_text** = "Cut"

  [this text may be customized at run-time]
- static const char * **paste_menu_text** = "Paste"

  [this text may be customized at run-time]

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Input

- void draw () override

  Draws the widget.
- int handle_key ()

  Handles a keystroke.

- int handle_rmb ()

  Handle right mouse button down events.

Protected Member Functions inherited from Fl_Input_

- int apply_undo ()

  Apply the current undo/redo operation.
- void drawtext (int, int, int, int)

  Draws the text in the passed bounding box.
- void drawtext (int, int, int, int, bool draw_active)

  Draws the text in the passed bounding box.
- void handle_mouse (int, int, int, int, int keepmark=0)

  Handles mouse clicks and mouse moves.
- int handletext (int e, int, int, int, int)

  Handles all kinds of text field related events.
- int line_end (int i) const

  Finds the end of a line.
- int line_start (int i) const

  Finds the start of a line.
- int **linesPerPage** ()
- void **maybe_do_callback** (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
- int up_down_position (int, int keepmark=0)

  Moves the cursor to the column given by `up_down_pos`.
- int word_end (int i) const

  Finds the end of a word.
- int word_start (int i) const

  Finds the start of a word.
- int **xscroll** () const
- int **yscroll** () const
- void **yscroll** (int yOffset)

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

Draws the label in an arbitrary bounding box.

- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

13.86.1 Detailed Description

This widget is a subclass of Fl_Output that displays multiple lines of text.
It also displays tab characters as whitespace to the next column.
Note that this widget does not support scrollbars, or per-character color control.
If you are presenting large amounts of read-only text and need scrollbars, or full color control of characters, then use Fl_Text_Display. If you want to display HTML text, use Fl_Help_View.
A caret cursor (^) shows the keyboard navigation mark for keyboard selection of the output text, e.g. Arrow Keys to move the cursor, Shift + Arrow Keys to create a text selection, and '^C' to copy the selected text to the paste buffer. The caret cursor can be disabled by disabling the widget's "visible focus" using clear_visible_focus(), inherited from the Fl_Widget base class. Doing this also disables the widget's keyboard navigation.

13.86.2 Constructor & Destructor Documentation

13.86.2.1 Fl_Multiline_Output()

```
Fl_Multiline_Output::Fl_Multiline_Output (
            int X,
            int Y,
            int W,
            int H,
            const char * l = 0 )
```

Creates a new Fl_Multiline_Output widget using the given position, size, and label string.
The default boxtype is FL_DOWN_BOX.
Inherited destructor destroys the widget and any value associated with it.
The documentation for this class was generated from the following files:

- Fl_Multiline_Output.H
- Fl_Input.cxx

13.87 Fl_Native_File_Chooser Class Reference

This class lets an FLTK application easily and consistently access the operating system's native file chooser.
`#include <Fl_Native_File_Chooser.H>`

Public Types

- enum Option {
  NO_OPTIONS = 0x0000 , SAVEAS_CONFIRM = 0x0001 , NEW_FOLDER = 0x0002 , PREVIEW = 0x0004 , USE_FILTER_EXT = 0x0008 }

- enum Type {
  BROWSE_FILE = 0 , BROWSE_DIRECTORY , BROWSE_MULTI_FILE , BROWSE_MULTI_DIRECTORY , BROWSE_SAVE_FILE , BROWSE_SAVE_DIRECTORY }

Public Member Functions

- int count () const

  Returns the number of filenames (or directory names) the user selected.
- const char * **directory** () const

  Returns the current preset directory() value.
- void directory (const char *val)

  Preset the directory the browser will show when opened.
- const char * errmsg () const

  Returns a system dependent error message for the last method that failed.
- const char * filename () const

  Return the filename the user chose.
- const char * filename (int i) const

  Return one of the filenames the user selected.
- const char * filter () const

  Returns the filter string last set.
- void filter (const char *f)

  Sets the filename filters used for browsing.
- int filter_value () const

  Returns which filter value was last selected by the user.
- void filter_value (int i)

  Sets which filter will be initially selected.
- int **filters** () const

  Gets how many filters were available, not including "All Files".
- Fl_Native_File_Chooser (int val=BROWSE_FILE)

  The constructor.
- int **options** () const

  Gets the platform specific Fl_Native_File_Chooser::Option flags.
- void options (int o)

  Sets the platform specific chooser options to `val`.
- const char * **preset_file** () const

  Get the preset filename.
- void preset_file (const char *f)

  Sets the default filename for the chooser.
- int show ()

  Post the chooser's dialog.
- const char * title () const

  Get the title of the file chooser's dialog window.
- void title (const char *t)

  Set the title of the file chooser's dialog window.
- int **type** () const

  Gets the current Fl_Native_File_Chooser::Type of browser.
- void **type** (int t)

  Sets the current Fl_Native_File_Chooser::Type of browser.
- ~Fl_Native_File_Chooser ()

  Destructor.

Static Public Attributes

- static const char * **file_exists_message** = "File exists. Are you sure you want to overwrite?"

  Localizable message.

13.87.1 Detailed Description

This class lets an FLTK application easily and consistently access the operating system's native file chooser.
Some operating systems have very complex and specific file choosers that many users want access to specifically, instead of FLTK's default file chooser(s).
In cases where there is no native file browser, FLTK's own file browser is used instead.
To use this widget, use the following include in your code:

```
#include <FL/Fl_Native_File_Chooser.H>
```

The following example shows how to pick a single file:

```
// Create and post the local native file chooser
#include <FL/Fl_Native_File_Chooser.H>
[..]
Fl_Native_File_Chooser fnfc;
fnfc.title("Pick a file");
fnfc.type(Fl_Native_File_Chooser::BROWSE_FILE);
fnfc.filter("Text\t*.txt\n"
            "C Files\t*.{cxx,h,c}");
fnfc.directory("/var/tmp");           // default directory to use
// Show native chooser
switch ( fnfc.show() ) {
  case -1: printf("ERROR: %s\n", fnfc.errmsg());    break;  // ERROR
  case  1: printf("CANCEL\n");                      break;  // CANCEL
  default: printf("PICKED: %s\n", fnfc.filename()); break;  // FILE CHOSEN
}
```

The Fl_Native_File_Chooser widget transmits UTF-8 encoded filenames to its user. It is recommended to open files that may have non-ASCII names with the fl_fopen() or fl_open() utility functions that handle these names in a cross-platform way (whereas the standard fopen()/open() functions fail on the Windows platform to open files with a non-ASCII name).

Platform Specific Caveats

- Under X11/Wayland the dialog is chosen as follows:

1. If command `zenity` is available at run-time and if `Fl::option(OPTION_FNFC_USES_ZENITY)` is turned on, the `zenity` -based dialog opens. This is expected to be more appropriate than other dialog forms for sandboxed apps, but member function filter_value() is not effective.

2. Else if the app runs under the KDE desktop and if `Fl::option(OPTION_FNFC_USES_KDIALOG)` is turned on, and if command `kdialog` is available at run-time, the `kdialog` -based dialog opens. Member function filter_value() is not effective with this dialog.

3. Else if the GTK library is available at run-time on the computer and if `Fl::option(OPTION_FNFC_USES_GTK)` is not turned off, the GTK-styled dialog opens. Call fl_register_images() to add a "Preview" button to this dialog. Use the static public attributes of class Fl_File_Chooser to localize the browser.

4. Otherwise, FLTK's own dialog produced by the Fl_File_Chooser class opens. Call fl_register_images() to add a "Preview" button to it. It's best if you also call Fl_File_Icon::load_system_icons() at the start of main(), to enable the nicer looking file browser widgets. Use the static public attributes of class Fl_File_Chooser to localize the browser.

- Some operating systems support certain OS specific options; see Fl_Native_File_Chooser::options() for a list.

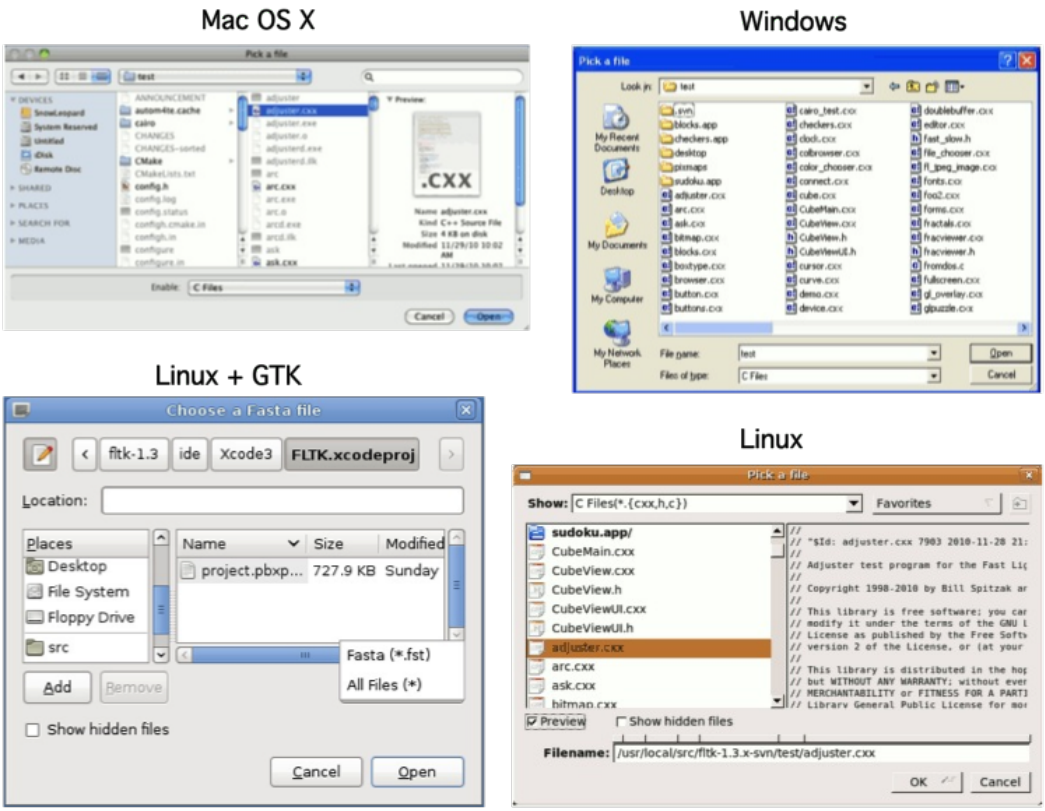


Figure 13.33 The Fl_Native_File_Chooser on different platforms

13.87.2 Member Enumeration Documentation

13.87.2.1 Option

enum Fl_Native_File_Chooser::Option

Enumerator

NO_OPTIONS	no options enabled
SAVEAS_CONFIRM	Show native 'Save As' overwrite confirm dialog.
NEW_FOLDER	Show 'New Folder' icon (if supported)
PREVIEW	enable preview mode (if supported)
USE_FILTER_EXT	Chooser filter presets the output file extension (if supported)

13.87.2.2 Type

enum Fl_Native_File_Chooser::Type

Enumerator

BROWSE_FILE	browse files (lets user choose one file)
BROWSE_DIRECTORY	browse directories (lets user choose one directory)
BROWSE_MULTI_FILE	browse files (lets user choose multiple files)
BROWSE_MULTI_DIRECTORY	browse directories (lets user choose multiple directories)
BROWSE_SAVE_FILE	browse to save a file
BROWSE_SAVE_DIRECTORY	browse to save a directory

13.87.3 Constructor & Destructor Documentation

13.87.3.1 Fl_Native_File_Chooser()

```
Fl_Native_File_Chooser::Fl_Native_File_Chooser (
            int val = BROWSE_FILE )
```

The constructor.
Internally allocates the native widgets. Optional `val` presets the type of browser this will be, which can also be changed with type().

13.87.3.2 ~Fl_Native_File_Chooser()

```
Fl_Native_File_Chooser::~Fl_Native_File_Chooser ( )
```

Destructor.
Deallocates any resources allocated to this widget.

13.87.4 Member Function Documentation

13.87.4.1 count()

```
int Fl_Native_File_Chooser::count ( ) const
```

Returns the number of filenames (or directory names) the user selected.
Example:

```
if ( fnfc->show() == 0 ) {
    // Print all filenames user selected
    for (int n=0; n<fnfc->count(); n++ ) {
        printf("%d) '%s'\n", n, fnfc->filename(n));
    }
}
```

13.87.4.2 directory()

```
void Fl_Native_File_Chooser::directory (
            const char * val )
```

Preset the directory the browser will show when opened.
If `val` is NULL, or no directory is specified, the chooser will attempt to use the last non-cancelled folder.

13.87.4.3 errmsg()

```
const char * Fl_Native_File_Chooser::errmsg ( ) const
```

Returns a system dependent error message for the last method that failed.
This message should at least be flagged to the user in a dialog box, or to some kind of error log. Contents will be valid only for methods that document errmsg() will have info on failures.

13.87.4.4 filename() [1/2]

```
const char * Fl_Native_File_Chooser::filename ( ) const
```

Return the filename the user chose.
Use this if only expecting a single filename. If more than one filename is expected, use filename(int) instead. Return value may be "" if no filename was chosen (eg. user cancelled).

13.87.4.5 filename() [2/2]

```
const char * Fl_Native_File_Chooser::filename (
            int i ) const
```

Return one of the filenames the user selected.
Use count() to determine how many filenames the user selected.
Example:

```
if ( fnfc->show() == 0 ) {
    // Print all filenames user selected
    for (int n=0; n<fnfc->count(); n++ ) {
        printf("%d) '%s'\n", n, fnfc->filename(n));
    }
}
```

13.87.4.6 filter() [1/2]

```
const char * Fl_Native_File_Chooser::filter ( ) const
```

Returns the filter string last set.
Can be NULL if no filter was set.

13.87.4.7 filter() [2/2]

```
void Fl_Native_File_Chooser::filter (
            const char * f )
```

Sets the filename filters used for browsing.
The default is NULL, which browses all files.
The filter string can be any of:

- A single wildcard (eg. "∗.txt")
- Multiple wildcards (eg. "∗.{cxx,h,H}")
- A descriptive name followed by a "\t" and a wildcard (eg. "Text Files\t∗.txt")
- A list of separate wildcards with a "\n" between each (eg. "∗.{cxx,H}\n∗.txt")
- A list of descriptive names and wildcards (eg. "C++ Files\t∗.{cxx,H}\nTxt Files\t∗.txt")

The format of each filter is a wildcard, or an optional user description followed by '\t' and the wildcard.
On most platforms, each filter is available to the user via a pulldown menu in the file chooser. The 'All Files' option is always available to the user.

13.87.4.8 filter_value() [1/2]

```
int Fl_Native_File_Chooser::filter_value ( ) const
```

Returns which filter value was last selected by the user.
This is only valid if the chooser returns success and if the particular file chooser supports it. Otherwise the value is not changed.
Some "native" file choosers don't support returning the filter selection by the user, particularly the system dialog based browsers:

- kdialog (KDE system dialog)
- zenity (another system dialog based chooser).

Note: this list may not be complete.
These system file chooser dialogs don't return the filter value chosen by the user.

See also

filter(const char ∗f)

13.87.4.9 filter_value() [2/2]

```
void Fl_Native_File_Chooser::filter_value (
            int i )
```

Sets which filter will be initially selected.
The first filter is indexed as 0. If filter_value() == filters(), then "All Files" was chosen. If filter_value() > filters(), then a custom filter was set.
Some "native" file choosers don't support this way to set the initial filter type, particularly the system dialog based browsers:

- kdialog (KDE system dialog)
- zenity (another system dialog based chooser).

Note: this list may not be complete.
As far as we know these dialogs use the **first** item in the list of filter values as the initial filter presented to the user.

See also

filter(const char *f)

13.87.4.10 options()

```
void Fl_Native_File_Chooser::options (
            int o )
```

Sets the platform specific chooser options to `val`.

`val` is expected to be one or more Fl_Native_File_Chooser::Option flags ORed together. Some platforms have OS-specific functions that can be enabled/disabled via this method.

Flag	Description	Win	Mac	Other
NEW_FOLDER	Shows the 'New Folder' button.	Ignored	Used	Used
PREVIEW	Enables the 'Preview' mode by default.	Ignored	Ignored	Used
SAVEAS_CONFIRM	Confirm dialog if BROWSE_SAVE_FILE file exists.	Used	Used	Used
USE_FILTER_EXT	Chooser filter presets the output file extension.	Ignored	Used	Used (GTK)

13.87.4.11 preset_file()

```
void Fl_Native_File_Chooser::preset_file (
            const char * f )
```

Sets the default filename for the chooser.

Use directory() to set the default directory. Mainly used to preset the filename for save dialogs, and on most platforms can be used for opening files as well.

13.87.4.12 show()

```
int Fl_Native_File_Chooser::show ( )
```

Post the chooser's dialog.

Blocks until dialog has been completed or cancelled.

Returns

- 0 – user picked a file
- 1 – user cancelled
- -1 – failed; errmsg() has reason

13.87.4.13 title() [1/2]

```
const char * Fl_Native_File_Chooser::title ( ) const
```

Get the title of the file chooser's dialog window.

Return value may be NULL if no title was set.

13.87.4.14 title() [2/2]

```
void Fl_Native_File_Chooser::title (
            const char * t )
```

Set the title of the file chooser's dialog window.

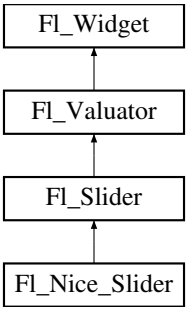
Can be NULL if no title desired. The default title varies according to the platform, so you are advised to set the title explicitly.

The documentation for this class was generated from the following files:

- Fl_Native_File_Chooser.H
- Fl_Native_File_Chooser.cxx
- Fl_Native_File_Chooser_GTK.cxx

13.88 Fl_Nice_Slider Class Reference

Inheritance diagram for Fl_Nice_Slider:



Public Member Functions

- **Fl_Nice_Slider** (int X, int Y, int W, int H, const char ∗L=0)

Public Member Functions inherited from Fl_Slider

- void bounds (double a, double b)

  Sets the minimum (a) and maximum (b) values for the valuator widget.
- Fl_Slider (int X, int Y, int W, int H, const char ∗L=0)

  Creates a new Fl_Slider widget using the given position, size, and label string.
- **Fl_Slider** (uchar t, int X, int Y, int W, int H, const char ∗L)

  Creates a new Fl_Slider widget using the given type, position, size, and label string.
- int handle (int) override

  Handles the specified event.
- uchar num_ticks () const

  Gets the number of tick marks.
- Scale_Type scale () const

  Gets the scale type.
- void scale (Scale_Type s)

  Sets the scale type.
- int scrollvalue (int pos, int size, int first, int total)

  Sets the size and position of the sliding knob in the box.
- Fl_Boxtype **slider** () const

  Gets the slider box type.
- void **slider** (Fl_Boxtype c)

  Sets the slider box type.
- float **slider_size** () const

  Get the dimensions of the moving piece of slider.
- void slider_size (double v)

  Set the dimensions of the moving piece of slider.
- uchar ticks () const

  Gets the tick mark position.
- void ticks (uchar t, uchar num_ticks=11)

  Sets the tick mark position for Fl_Nice_Slider.

Public Member Functions inherited from Fl_Valuator

- void **bounds** (double a, double b)

  Sets the minimum (a) and maximum (b) values for the valuator widget.
- double **clamp** (double)

  Clamps the passed value to the valuator range.
- virtual int format (char ∗)

  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.
- virtual std::string format_str ()

  C++11 API for Fl_Valuator::format(char∗ buffer).
- double increment (double, int)

  Adds n times the step value to the passed value.
- double maximum () const

  Gets the maximum value for the valuator.
- void maximum (double a)

  Sets the maximum value for the valuator.
- double minimum () const

  Gets the minimum value for the valuator.
- void minimum (double a)

  Sets the minimum value for the valuator.
- void precision (int digits)

  Sets the step value to 1.0 / 10^{digits}.
- void range (double a, double b)

  Sets the minimum and maximum values for the valuator.
- double round (double)

  Round the passed value to the nearest step increment.
- double step () const

  Gets or sets the step value.
- void **step** (double a, int b)

  See double Fl_Valuator::step() const
- void **step** (double s)

  See double Fl_Valuator::step() const.
- void **step** (int a)

  See double Fl_Valuator::step() const
- double value () const

  Gets the floating point(double) value.
- int value (double)

  Sets the current value.
- ∼**Fl_Valuator** () override

  Destructor is accessible despite protected constructor.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

Returns whether the widget and all of its parents are active.

- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ **as_gl_window** () const
- virtual Fl_Group ∗ as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const ∗ **as_group** () const
- virtual Fl_Window ∗ as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ **as_window** () const
- void bind_deimage (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗cb, void ∗p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

Sets a widget to accept input.

- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.

- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

Returns a pointer to the parent widget.

- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.

- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Public Types inherited from Fl_Slider

- enum Scale_Type : uchar { LINEAR_SCALE = 0 , LOG_SCALE }

  Bitset for tick mark positions for Fl_Nice_Slider.
- enum Tick_Position : uchar {
  TICKS_NONE = 0x00 , TICKS_BELOW = 0x01 , TICKS_ABOVE = 0x02 , TICKS_LEFT = 0x01 ,
  TICKS_RIGHT = 0x02 }

  Bitset for tick mark positions for Fl_Nice_Slider.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Slider

- void draw () override

  Draws the widget.
- void **draw** (int, int, int, int)
- void draw_ticks (const Fl_Rect &r, int S)

  Draw tick marks.
- int **handle** (int, int, int, int, int)
- double increment_lin_log (double v, int n, int range)

  Adds a sensible step value to the passed value.
- double position_to_value (double pos) const

  Converts a normalized position (0.0 to 1.0) to a value based on the current scale type.
- double value_to_position (double val) const

  Converts a value to a normalized position (0.0 to 1.0) based on the current scale type.

Protected Member Functions inherited from Fl_Valuator

- Fl_Valuator (int X, int Y, int W, int H, const char *L)

  Creates a new Fl_Valuator widget using the given position, size, and label string.
- void **handle_drag** (double newvalue)

  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.
- void **handle_push** ()

  Stores the current value in the previous value.
- void **handle_release** ()

  Called after an FL_WHEN_RELEASE event is received and before the callback.
- int **horizontal** () const

  Tells if the valuator is an FL_HORIZONTAL one.

- double **previous_value** () const

  Gets the previous floating point value before an event changed it.
- void **set_value** (double v)

  Sets the current floating point value.
- double **softclamp** (double)

  Clamps the value, but accepts v if the previous value is not already out of range.
- virtual void value_damage ()

  Asks for partial redraw.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

The documentation for this class was generated from the following files:

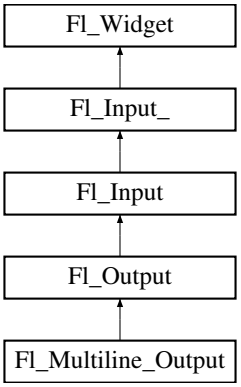
- Fl_Nice_Slider.H
- Fl_Slider.cxx

13.89 Fl_Output Class Reference

This widget displays a piece of text.

`#include <Fl_Output.H>`

Inheritance diagram for Fl_Output:



Public Member Functions

- Fl_Output (int X, int Y, int W, int H, const char *l=0)

  Creates a new Fl_Output widget using the given position, size, and label string.

Public Member Functions inherited from Fl_Input

- Fl_Input (int, int, int, int, const char *=0)

  Creates a new Fl_Input widget using the given position, size, and label string.
- int handle (int) override

  Handles the specified event.

Public Member Functions inherited from Fl_Input_

- int append (const char *t, int l=0, char keep_selection=0)

  Append text at the end.
- bool can_redo () const

  Check if there is a redo action available.
- bool can_undo () const

  Check if the last operation can be undone.
- int copy (int clipboard)

  Put the current selection into the clipboard.
- int copy_cuts ()

  Copies the yank buffer to the clipboard.
- Fl_Color cursor_color () const

  Gets the color of the cursor.
- void cursor_color (Fl_Color n)

  Sets the color of the cursor.
- int cut ()

  Deletes the current selection.
- int cut (int a, int b)

  Deletes all characters between index `a` and `b`.
- int cut (int n)

  Deletes the next `n` bytes rounded to characters before or after the cursor.
- double dvalue () const

Returns the widget text interpreted as a floating point number.

- Fl_Input_ (int, int, int, int, const char *=0)

  Creates a new Fl_Input_ widget.
- unsigned int index (int i) const

  Returns the character at index `i`.
- int input_type () const

  Gets the input field type.
- void input_type (int t)

  Sets the input field type.
- int insert (const char *t, int l=0)

  Inserts text at the cursor position.
- int insert_position () const

  Gets the position of the text cursor.
- int insert_position (int p)

  Sets the cursor position and mark.
- int insert_position (int p, int m)

  Sets the index for the cursor and mark.
- int ivalue () const

  Returns the widget text interpreted as a signed integer.
- int mark () const

  Gets the current selection mark.
- int mark (int m)

  Sets the current selection mark.
- int maximum_size () const

  Gets the maximum length of the input field in characters.
- void maximum_size (int m)

  Sets the maximum length of the input field in characters.
- const char * placeholder () const

  Get the placeholder text.
- void placeholder (const char *text)

  Set the placeholder text.
- int position () const
- int position (int p)
- int position (int p, int m)
- int readonly () const

  Gets the read-only state of the input field.
- void readonly (int b)

  Sets the read-only state of the input field.
- int redo ()

  Redo previous undo operation.
- int replace (int b, int e, const char *text, int ilen=0)

  Deletes text from `b` to `e` and inserts the new string `text`.
- void resize (int, int, int, int) override

  Changes the size of the widget.
- int shortcut () const

  Return the shortcut key associated with this widget.
- void shortcut (int s)

  Sets the shortcut key associated with this widget.
- int size () const

  Returns the number of bytes in value().
- void size (int W, int H)

Sets the width and height of this widget.

- int static_value (const char ∗)

  Changes the widget text.
- int static_value (const char ∗, int)

  Changes the widget text.
- int tab_nav () const

  Gets whether the Tab key causes focus navigation in multiline input fields or not.
- void tab_nav (int val)

  Sets whether the Tab key does focus navigation, or inserts tab characters into Fl_Multiline_Input.
- Fl_Color textcolor () const

  Gets the color of the text in the input field.
- void textcolor (Fl_Color n)

  Sets the color of the text in the input field.
- Fl_Font textfont () const

  Gets the font of the text in the input field.
- void textfont (Fl_Font s)

  Sets the font of the text in the input field.
- Fl_Fontsize textsize () const

  Gets the size of the text in the input field.
- void textsize (Fl_Fontsize s)

  Sets the size of the text in the input field.
- int undo ()

  Undoes previous changes to the text buffer.
- const char ∗ value () const

  Returns the text displayed in the widget.
- int value (const char ∗)

  Changes the widget text.
- int value (const char ∗, int)

  Changes the widget text.
- int value (double value)

  Changes the widget text to a floating point number ("%g").
- int value (int value)

  Changes the widget text to a signed integer number.
- int wrap () const

  Gets the word wrapping state of the input field.
- void wrap (int b)

  Sets the word wrapping state of the input field.
- ∼Fl_Input_ ()

  Destroys the widget.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

Gets the label alignment.

- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

Hides the widget.

- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.

- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

Internal use only - "for hacks only".

- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget ∗widget, void ∗data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char ∗t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char ∗, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Static Public Attributes inherited from Fl_Input

- static const char ∗ **copy_menu_text** = "Copy"

  [this text may be customized at run-time]
- static const char ∗ **cut_menu_text** = "Cut"

  [this text may be customized at run-time]
- static const char ∗ **paste_menu_text** = "Paste"

  [this text may be customized at run-time]

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Input

- void draw () override

  Draws the widget.
- int handle_key ()

  Handles a keystroke.
- int handle_rmb ()

  Handle right mouse button down events.

Protected Member Functions inherited from Fl_Input_

- int apply_undo ()

  Apply the current undo/redo operation.
- void drawtext (int, int, int, int)

  Draws the text in the passed bounding box.
- void drawtext (int, int, int, int, bool draw_active)

  Draws the text in the passed bounding box.
- void handle_mouse (int, int, int, int, int keepmark=0)

  Handles mouse clicks and mouse moves.
- int handletext (int e, int, int, int, int)

  Handles all kinds of text field related events.
- int line_end (int i) const

  Finds the end of a line.
- int line_start (int i) const

  Finds the start of a line.
- int **linesPerPage** ()
- void **maybe_do_callback** (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
- int up_down_position (int, int keepmark=0)

  Moves the cursor to the column given by `up_down_pos`.
- int word_end (int i) const

  Finds the end of a word.
- int word_start (int i) const

  Finds the start of a word.
- int **xscroll** () const
- int **yscroll** () const
- void **yscroll** (int yOffset)

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

13.89.1 Detailed Description

This widget displays a piece of text.
When you set the value() , Fl_Output does a strcpy() to its own storage, which is useful for program-generated values. The user may select portions of the text using the mouse and paste the contents into other fields or programs.

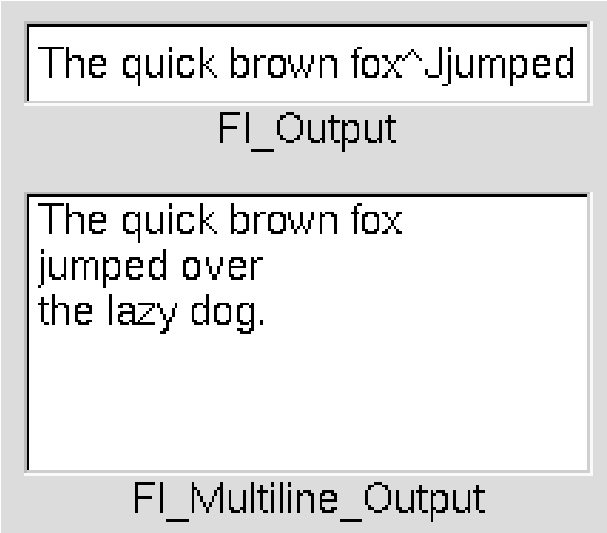


Figure 13.34 Fl_Output

There is a single subclass, Fl_Multiline_Output, which allows you to display multiple lines of text. Fl_Multiline_Output does not provide scroll bars. If a more complete text editing widget is needed, use Fl_Text_Display instead.
The text may contain any characters except \0, and will correctly display anything, using ^X notation for unprintable control characters and \nnn notation for unprintable characters with the high bit set. It assumes the font can draw any characters in the ISO-Latin1 character set.

13.89.2 Constructor & Destructor Documentation

13.89.2.1 Fl_Output()

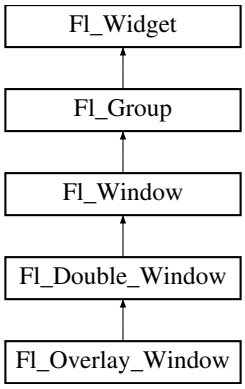
```
Fl_Output::Fl_Output (
            int X,
            int Y,
            int W,
            int H,
            const char * l = 0 )
```

Creates a new Fl_Output widget using the given position, size, and label string.
The default boxtype is FL_DOWN_BOX.
Inherited destructor destroys the widget and any value associated with it.
The documentation for this class was generated from the following files:

- Fl_Output.H
- Fl_Input.cxx

13.90 Fl_Overlay_Window Class Reference

This window provides double buffering and also the ability to draw the "overlay" which is another picture placed on top of the main image.
`#include <Fl_Overlay_Window.H>`
Inheritance diagram for Fl_Overlay_Window:



Public Member Functions

- Fl_Overlay_Window ∗ as_overlay_window () override

  Return non-null if this is an Fl_Overlay_Window object.
- int **can_do_overlay** ()

  Returns non-zero if there's hardware overlay support.
- virtual void draw_overlay ()=0

  You must subclass Fl_Overlay_Window and provide this method.
- void flush () override

  Forces the window to be drawn, this window is also made current and calls draw().
- void hide () override

  Makes a widget invisible.
- void redraw_overlay ()

  Call this to indicate that the overlay data has changed and needs to be redrawn.
- void resize (int, int, int, int) override

  Changes the size or position of the widget.
- void show () override

  Makes a widget visible.
- void **show** (int a, char ∗∗b)

  Same as Fl_Window::show(int a, char ∗∗b)
- ∼**Fl_Overlay_Window** ()

  Destroys the window and all child widgets.

Public Member Functions inherited from Fl_Double_Window

- Fl_Double_Window ∗ as_double_window () override

  Return non-null if this is an Fl_Double_Window object.
- **Fl_Double_Window** (int W, int H, const char ∗l=0)

  Creates a new Fl_Double_Window widget using the given position, size, and label (title) string.
- **Fl_Double_Window** (int X, int Y, int W, int H, const char ∗l=0)

  See Fl_Double_Window::Fl_Double_Window(int w, int h, const char ∗label = 0)
- void **show** (int a, char ∗∗b)

  Same as Fl_Window::show(int a, char ∗∗b)
- ∼Fl_Double_Window ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Window

- void allow_expand_outside_parent ()

  Allow this subwindow to expand outside the area of its parent window.
- Fl_Window const * as_window () const override
- Fl_Window * as_window () override

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- unsigned int **border** () const

  Returns whether the window possesses a border.
- void border (int b)

  Sets whether or not the window manager border is around the window.
- void clear_border ()

  Fast inline function to turn the window manager border off.
- void clear_modal_states ()

  Clears the "modal" flags and converts a "modal" or "non-modal" window back into a "normal" window.
- void **copy_label** (const char *a)

  Sets the window titlebar label to a copy of a character string.
- void cursor (const Fl_RGB_Image *, int, int)

  Changes the cursor for this window using the provided image as cursor's shape.
- void cursor (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)

  For back compatibility only.
- void cursor (Fl_Cursor)

  Changes the cursor for this window.
- int decorated_h () const

  Returns the window height including any window title bar and any frame added by the window manager.
- int decorated_w () const

  Returns the window width including any frame added by the window manager.
- void default_cursor (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)

  For back compatibility only.
- void default_cursor (Fl_Cursor)

  Sets the default window cursor.
- void **draw_backdrop** ()

  Draw the background image if one is set and is aligned inside.
- Fl_Window (int w, int h, const char *title=0)

  Creates a window from the given width `w`, height `h`, and `title`.
- Fl_Window (int x, int y, int w, int h, const char *title=0)

  Creates a window from the given position (x, y), size (w, h) and title.
- void free_position ()

  Undoes the effect of a previous resize() or show() so that the next time show() is called the window manager is free to position the window.
- void fullscreen ()

  Makes the window completely fill one or more screens, without any window manager border visible.
- unsigned int **fullscreen_active** () const

  Returns non zero if FULLSCREEN flag is set, 0 otherwise.
- void **fullscreen_off** ()

  Turns off any side effects of fullscreen()
- void **fullscreen_off** (int X, int Y, int W, int H)

  Turns off any side effects of fullscreen() and does resize(X,Y,W,H).
- void fullscreen_screens (int top, int bottom, int left, int right)

  Sets which screens should be used when this window is in fullscreen mode.
- uchar get_size_range (int *minw, int *minh, int *maxw=NULL, int *maxh=NULL, int *dw=NULL, int *dh=NULL, int *aspect=NULL) const

Gets the allowable range to which the user can resize this window.

- int handle (int) override

  Handles the specified event.
- void hide () override

  Removes the window from the screen.
- void **hotspot** (const Fl_Widget &p, int offscreen=0)

  See void Fl_Window::hotspot(int x, int y, int offscreen = 0)
- void **hotspot** (const Fl_Widget ∗, int offscreen=0)

  See void Fl_Window::hotspot(int x, int y, int offscreen = 0)
- void hotspot (int x, int y, int offscreen=0)

  Positions the window so that the mouse is pointing at the given position, or at the center of the given widget, which may be the window itself.
- const void ∗ icon () const

  Gets the current icon window target dependent data.
- void icon (const Fl_RGB_Image ∗)

  Sets or resets a single window icon.
- void icon (const void ∗ic)

  Platform-specific method to set the window icon usable on Windows and X11 only.
- void iconize ()

  Iconifies the window.
- const char ∗ **iconlabel** () const

  See void Fl_Window::iconlabel(const char∗)
- void **iconlabel** (const char ∗)

  Sets the icon label.
- void icons (const Fl_RGB_Image ∗[ ], int)

  Sets the window icons.
- void icons (HICON big_icon, HICON small_icon)

  Sets the window icons using HICON handles (Windows platform only).
- const char ∗ **label** () const

  See void Fl_Window::label(const char∗)
- void **label** (const char ∗)

  Sets the window title bar label.
- void **label** (const char ∗label, const char ∗iconlabel)

  Sets the icon label.
- void make_current ()

  Sets things up so that the drawing functions in <FL/fl_draw.H> will go into this window.
- void maximize ()

  Maximizes a top-level window to its current screen.
- unsigned int **maximize_active** () const

  Returns whether the window is currently maximized.
- unsigned int **menu_window** () const

  Returns true if this window is a menu window.
- unsigned int modal () const

  Returns true if this window is modal.
- unsigned int **non_modal** () const

  Returns true if this window is modal or non-modal.
- fl_uintptr_t os_id () const

  Returns a platform-specific identification of a shown window, or 0 if not shown.
- unsigned int **override** () const

Returns non zero if OVERRIDE flag is set, 0 otherwise.

- void resize (int X, int Y, int W, int H) override

  Changes the size and position of the window.
- int screen_num () const

  The number of the screen containing the mapped window.
- void screen_num (int screen_num)

  Set the number of the screen where to map the window.
- void set_menu_window ()

  Marks the window as a menu window.
- void set_modal ()

  A "modal" window, when shown(), will prevent any events from being delivered to other windows in the same program, and will also remain on top of the other windows (if the X window manager supports the "transient for" property).
- void set_non_modal ()

  A "non-modal" window (terminology borrowed from Microsoft Windows) acts like a modal() one in that it remains on top, but it has no effect on event delivery.
- void **set_override** ()

  Activates the flags NOBORDER|OVERRIDE.
- void set_tooltip_window ()

  Marks the window as a tooltip window.
- const Fl_Image * shape () const

  Returns the image controlling the window shape or NULL.
- void shape (const Fl_Image &b)

  Set the window's shape with an Fl_Image.
- void shape (const Fl_Image *img)

  Assigns a non-rectangular shape to the window.
- void show () override

  Puts the window on the screen.
- void show (int argc, char **argv)

  Puts the window on the screen with show() and parses command-line arguments.
- int shown () const

  Returns non-zero if show() has been called (but not hide()).
- void size_range (int minw, int minh, int maxw=0, int maxh=0, int dw=0, int dh=0, int aspect=0)

  Sets the allowable range to which the user can resize this window.
- unsigned int **tooltip_window** () const

  Returns true if this window is a tooltip window.
- void un_maximize ()

  Returns a previously maximized top-level window to its previous size.
- void wait_for_expose ()

  Waits for the window to be displayed after calling show().
- int **x_root** () const

  Gets the x position of the window on the screen.
- const char * xclass () const

  Returns the xclass for this window, or a default.
- void xclass (const char *c)

  Sets the xclass for this window.
- int **y_root** () const

  Gets the y position of the window on the screen.
- virtual ~Fl_Window ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Group

- Fl_Widget *& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget *o)

  See void Fl_Group::add(Fl_Widget &w)
- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget *const * array () const

  Returns a pointer to the internal array of children.
- Fl_Group const * as_group () const override
- Fl_Group * as_group () override

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const

  Returns the n'th child.
- int **children** () const

  Returns how many child widgets the group has.
- void clear ()

  Deletes all child widgets from memory recursively.
- unsigned int clip_children () const

  Returns the current clipping mode.
- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)

  Removes the widget at `index` from the group and deletes it.
- void end ()

  Exactly the same as current(this->parent()).
- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*
- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.
- void **forms_end** ()

  This is for forms compatibility only.
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

  This does insert(w, find(before)).
- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.
- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.

- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

Sets the current callback function for the widget.

- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.

- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

Sets the label type.

- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.

- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- **Fl_Overlay_Window** (int W, int H, const char ∗l=0)

  See Fl_Overlay_Window::Fl_Overlay_Window(int X, int Y, int W, int H, const char ∗l=0)
- Fl_Overlay_Window (int X, int Y, int W, int H, const char ∗l=0)

  Creates a new Fl_Overlay_Window widget using the given position, size, and label (title) string.

Protected Member Functions inherited from Fl_Window

- void default_size_range ()

  Protected method to calculate the default size range of a window.
- void draw () override

  Draws the widget.
- int force_position () const

  Returns the internal state of the window's `FORCE_POSITION` flag.
- void force_position (int force)

  Sets an internal flag that tells FLTK and the window manager to honor position requests.
- void free_icons ()

  Deletes all icons previously attached to the window.
- int is_resizable ()

  Protected method to determine whether a window is resizable.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()

  Returns the internal array of widget sizes and positions.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget *, int)

  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)

  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)

  Allow derived groups to act when a child widget is removed from the group.
- int * sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.

- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Public Types inherited from Fl_Window

- typedef struct HICON__ * **HICON**

Static Public Member Functions inherited from Fl_Window

- static Fl_Window * current ()

  Returns the last window that was made current.
- static void **default_callback** (Fl_Window *, void *v)

  Back compatibility: Sets the default callback v for win to call on close event.
- static void default_icon (const Fl_RGB_Image *)

  Sets a single default window icon.
- static void default_icons (const Fl_RGB_Image *[ ], int)

  Sets the default window icons.
- static void default_icons (HICON big_icon, HICON small_icon)

  Sets the default window icons (Windows platform only).
- static const char * default_xclass ()

  Returns the default xclass.
- static void default_xclass (const char *)

  Sets the default window xclass.
- static bool is_a_rescale ()

  Returns true when a window is being rescaled.
- static char show_next_window_iconic ()

  Returns the static flag whether the next window should be opened iconified.
- static void show_next_window_iconic (char stat)

  Sets a static flag whether the next window should be opened iconified.

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()

  Returns the currently active group.
- static void current (Fl_Group *g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Static Protected Attributes inherited from Fl_Window

- static Fl_Window * current_

  Stores the last window that was made current.

13.90.1 Detailed Description

This window provides double buffering and also the ability to draw the "overlay" which is another picture placed on top of the main image.
The overlay is designed to be a rapidly-changing but simple graphic such as a mouse selection box. Fl_Overlay_Window uses the overlay planes provided by your graphics hardware if they are available.
If no hardware support is found the overlay is simulated by drawing directly into the on-screen copy of the double-buffered window, and "erased" by copying the backbuffer over it again. This means the overlay will blink if you change the image in the window.

13.90.2 Constructor & Destructor Documentation

13.90.2.1 Fl_Overlay_Window()

```
Fl_Overlay_Window::Fl_Overlay_Window (
            int X,
            int Y,
            int W,
```

```
            int H,
            const char * l = 0 )  [protected]
```

Creates a new Fl_Overlay_Window widget using the given position, size, and label (title) string.
If the positions (x,y) are not given, then the window manager will choose them.

13.90.3 Member Function Documentation

13.90.3.1 as_overlay_window()

```
Fl_Overlay_Window * Fl_Overlay_Window::as_overlay_window ( )  [inline], [override], [virtual]
```

Return non-null if this is an Fl_Overlay_Window object.
Reimplemented from Fl_Window.

13.90.3.2 draw_overlay()

```
virtual void Fl_Overlay_Window::draw_overlay ( )  [pure virtual]
```

You must subclass Fl_Overlay_Window and provide this method.
It is just like a draw() method, except it draws the overlay. The overlay will have already been "cleared" when this is called. You can use any of the routines described in <FL/fl_draw.H>.

13.90.3.3 flush()

```
void Fl_Overlay_Window::flush ( )  [override], [virtual]
```

Forces the window to be drawn, this window is also made current and calls draw().
Reimplemented from Fl_Double_Window.

13.90.3.4 hide()

```
void Fl_Overlay_Window::hide ( )  [override], [virtual]
```

Makes a widget invisible.

See also

show(), visible(), visible_r()

Reimplemented from Fl_Double_Window.

13.90.3.5 redraw_overlay()

```
void Fl_Overlay_Window::redraw_overlay ( )
```

Call this to indicate that the overlay data has changed and needs to be redrawn.
The overlay will be clear until the first time this is called, so if you want an initial display you must call this after calling show().

13.90.3.6 resize()

```
void Fl_Overlay_Window::resize (
            int x,
            int y,
            int w,
            int h )  [override], [virtual]
```

Changes the size or position of the widget.
This is a virtual function so that the widget may implement its own handling of resizing. The default version does *not* call the redraw() method, but instead relies on the parent widget to do so because the parent may know a faster way to update the display, such as scrolling from the old position.
Some window managers under X11 call resize() a lot more often than needed. Please verify that the position or size of a widget did actually change before doing any extensive calculations.
position(X, Y) is a shortcut for resize(X, Y, w(), h()), and size(W, H) is a shortcut for resize(x(), y(), W, H).

Parameters

in	*x,y*	new position relative to the parent window
in	*w,h*	new size

See also

position(int,int), size(int,int)

Reimplemented from Fl_Double_Window.

13.90.3.7 show()

`void Fl_Overlay_Window::show ( ) [override], [virtual]`

Makes a widget visible.

An invisible widget never gets redrawn and does not get keyboard or mouse events, but can receive a few other events like FL_SHOW.

The visible() method returns true if the widget is set to be visible. The visible_r() method returns true if the widget and all of its parents are visible. A widget is only visible if visible() is true on it *and all of its parents*.

Changing it will send FL_SHOW or FL_HIDE events to the widget. *Do not change it if the parent is not visible, as this will send false FL_SHOW or FL_HIDE events to the widget.* redraw() is called if necessary on this or the parent.

See also

hide(), visible(), visible_r()

Reimplemented from Fl_Double_Window.

The documentation for this class was generated from the following files:

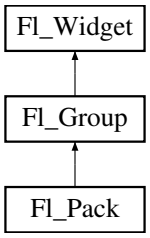
- Fl_Overlay_Window.H
- Fl_Overlay_Window.cxx

13.91 Fl_Pack Class Reference

This widget was designed to add the functionality of compressing and aligning widgets.

`#include <Fl_Pack.H>`

Inheritance diagram for Fl_Pack:



Public Types

- enum { **VERTICAL** = 0 , **HORIZONTAL** = 1 }

Public Member Functions

- void clear ()

  Deletes all child widgets with Fl_Group::clear().
- Fl_Pack (int X, int Y, int W, int H, const char ∗L=0)

  Creates a new Fl_Pack widget using the given position, size, and label string.
- uchar horizontal () const

Returns non-zero if Fl_Pack alignment is horizontal.

- void resize (int X, int Y, int W, int H) override

  Override Fl_Group resize behavior.
- int **spacing** () const

  Gets the number of extra pixels of blank space that are added between the children.
- void **spacing** (int i)

  Sets the number of extra pixels of blank space that are added between the children.

Public Member Functions inherited from Fl_Group

- Fl_Widget *& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget *o)

  See void Fl_Group::add(Fl_Widget &w)
- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget *const * array () const

  Returns a pointer to the internal array of children.
- Fl_Group const * as_group () const override
- Fl_Group * as_group () override

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const

  Returns the n'th child.
- int **children** () const

  Returns how many child widgets the group has.
- void clear ()

  Deletes all child widgets from memory recursively.
- unsigned int clip_children () const

  Returns the current clipping mode.
- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)

  Removes the widget at `index` from the group and deletes it.
- void end ()

  Exactly the same as current(this->parent()).
- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*
- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.
- void **forms_end** ()

  This is for forms compatibility only.
- int handle (int) override

  Handles the specified event.
- void init_sizes ()

  Resets the internal array of widget sizes and positions.

- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

  This does insert(w, find(before)).
- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.
- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.
- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

Deactivates the widget.

- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.

- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

Returns whether the widget's label uses '&' to indicate shortcuts.

- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.

- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void draw () override

  Draws the widget.
- void draw_filler_ (const Fl_Rect &rect)

  Draw a rectangular area as a background filler.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()

  Returns the internal array of widget sizes and positions.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget *, int)

  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)

  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)

  Allow derived groups to act when a child widget is removed from the group.
- int * sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

Draws a focus rectangle around the widget.

- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()

  Returns the currently active group.
- static void current (Fl_Group *g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

flags possible values enumeration.

13.91.1 Detailed Description

This widget was designed to add the functionality of compressing and aligning widgets.
If type() is Fl_Pack::HORIZONTAL all the children are resized to the height of the Fl_Pack, and are moved next to each other horizontally. If type() is not Fl_Pack::HORIZONTAL then the children are resized to the width and are stacked below each other. Then the Fl_Pack resizes itself to surround the child widgets.
You may want to put the Fl_Pack inside an Fl_Scroll.
The `'resizable()'` for Fl_Pack is set to NULL by default. Its behavior is slightly different than in a normal Fl_Group widget: only if the resizable() widget is the last widget in the group it is extended to take the full available width or height, respectively, of the Fl_Pack group.

Note

Fl_Pack is optimized to use a frame-only (`..._FRAME`) box type. Box types with background graphics (`..._BOX`) generally work, but can be slower witch a large number of children. Not all box types work well with FL_Pack. Avoid irregular graphics like FL_DIAMOND_BOX and background images.

You can nest Fl_Pack widgets or put them inside Fl_Scroll widgets or inside other group widgets but their behavior can sometimes be *"surprising"*. This is partly due to the fact that Fl_Pack widgets resize themselves during their draw() operation, trying to react on their child widgets resizing themselves during **their** draw() operations which can be confusing. If you want to achieve special resize behavior of nested group widgets it can sometimes be easier to derive your own specialized group widget than to try to make nested Fl_Pack widgets behave as expected.

See also

Fl_Group::resizable()

13.91.2 Constructor & Destructor Documentation

13.91.2.1 Fl_Pack()

```
Fl_Pack::Fl_Pack (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

Creates a new Fl_Pack widget using the given position, size, and label string.
The default boxtype is FL_NO_BOX.
The default type() is Fl_Pack::VERTICAL.
The destructor *also deletes all the children*. This allows a whole tree to be deleted at once, without having to keep a pointer to all the children in the user code. A kludge has been done so the Fl_Pack and all of its children can be automatic (local) variables, but you must declare the Fl_Pack *first*, so that it is destroyed last.

Parameters

in	*X,Y*	X and Y coordinates (position)
in	*W,H*	width and height, respectively
in	*L*	label (optional)

13.91.3 Member Function Documentation

13.91.3.1 clear()

```
void Fl_Pack::clear ( ) [inline]
```

Deletes all child widgets with Fl_Group::clear().

And sets to NULL the resizable() widget.

13.91.3.2 draw()

```
void Fl_Pack::draw ( ) [override], [protected], [virtual]
```

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();                 // calls Fl_Scrollbar::draw()
```

Reimplemented from Fl_Group.

13.91.3.3 draw_filler_()

```
void Fl_Pack::draw_filler_ (
            const Fl_Rect & rect ) [protected]
```

Draw a rectangular area as a background filler.
If the box type has a background (is not a frame), this will clip to the rectangular area and draw the box. Otherwise, it will draw a rectangle in widget color.

Parameters

in	*rect*	fill this rectangle inside the widget bounds

13.91.3.4 horizontal()

```
uchar Fl_Pack::horizontal ( ) const [inline]
```

Returns non-zero if Fl_Pack alignment is horizontal.

Returns

non-zero if Fl_Pack alignment is horizontal (Fl_Pack::HORIZONTAL)

Note

Currently the return value is the same as Fl_Group::type(), but this may change in the future. Do not set any other values than the following with Fl_Pack::type():

- Fl_Pack::VERTICAL (Default)
- Fl_Pack::HORIZONTAL

See class Fl_Pack documentation for details.

13.91.3.5 resize()

```
void Fl_Pack::resize (
            int X,
            int Y,
            int W,
            int H ) [override], [virtual]
```

Override Fl_Group resize behavior.
Resizing an Fl_Pack will not resize any of its children, but trigger a redraw, which in turn recalculates the dimensions of all children.

Parameters

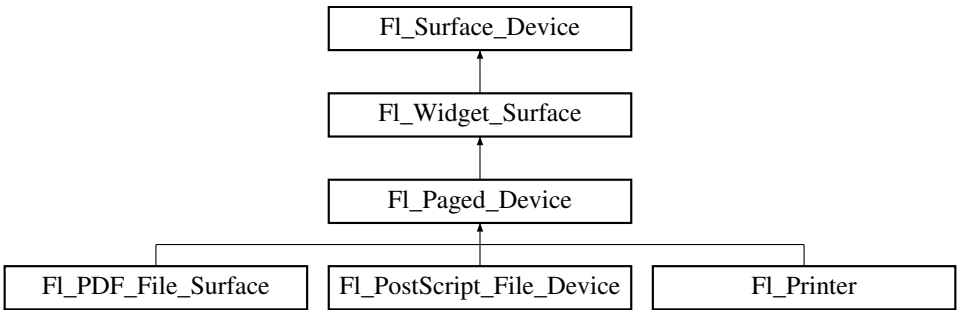
in	*X,Y,W,H*	new position and size of the Fl_Pack widget

Reimplemented from Fl_Group.
The documentation for this class was generated from the following files:

- Fl_Pack.H
- Fl_Pack.cxx

13.92 Fl_Paged_Device Class Reference

Represents page-structured drawing surfaces.
`#include <Fl_Paged_Device.H>`
Inheritance diagram for Fl_Paged_Device:



Classes

- struct page_format

  width, height and name of a page format

Public Types

- enum Page_Format {
  A0 = 0 , A1 , A2 , A3 ,
  A4 , A5 , A6 , A7 ,
  A8 , A9 , B0 , B1 ,
  B2 , B3 , B4 , B5 ,
  B6 , B7 , B8 , B9 ,
  B10 , **C5E** , **DLE** , EXECUTIVE ,
  FOLIO , LEDGER , LEGAL , LETTER ,
  TABLOID , **ENVELOPE** , **MEDIA** = 0x1000 }

  Possible page formats.
- enum Page_Layout { PORTRAIT = 0 , LANDSCAPE = 0x100 , REVERSED = 0x200 , ORIENTATION = 0x300 }

  Possible page layouts.

Public Member Functions

- virtual int begin_job (int pagecount=0, int *frompage=NULL, int *topage=NULL, char **perr_message=NULL)

  Begins a print job.
- virtual int begin_page (void)

  Begins a new printed page.
- virtual void end_job (void)

  To be called at the end of a print job.
- virtual int end_page (void)

  To be called at the end of each page.
- virtual void margins (int *left, int *top, int *right, int *bottom)

Computes the dimensions of margins that lie between the printable page area and the full page.

- void **print_widget** (Fl_Widget ∗widget, int delta_x=0, int delta_y=0)

  Synonym of draw(Fl_Widget∗, int, int)
- void **print_window** (Fl_Window ∗win, int x_off=0, int y_off=0)

  Synonym of draw_decorated_window(Fl_Window∗, int, int)
- virtual void rotate (float angle)

  Rotates the graphics operations relatively to paper.
- virtual void scale (float scale_x, float scale_y=0.)

  Changes the scaling of page coordinates.
- int start_job (int pagecount=0, int ∗frompage=NULL, int ∗topage=NULL, char ∗∗perr_message=NULL)

  Synonym of begin_job(int pagecount, int ∗frompage, int ∗topage, char ∗∗perr_message).
- int start_page ()

  Synonym of begin_page().
- virtual ∼**Fl_Paged_Device** ()

  The destructor.

Public Member Functions inherited from Fl_Widget_Surface

- void draw (Fl_Widget ∗widget, int delta_x=0, int delta_y=0)

  Draws the widget on the drawing surface.
- void draw_decorated_window (Fl_Window ∗win, int x_offset=0, int y_offset=0)

  Draws a window with its title bar and frame if any.
- virtual void origin (int ∗x, int ∗y)

  Computes the coordinates of the current origin of graphics functions.
- virtual void origin (int x, int y)

  Sets the position of the origin of graphics in the drawable part of the drawing surface.
- void print_window_part (Fl_Window ∗win, int x, int y, int w, int h, int delta_x=0, int delta_y=0)

  Draws a rectangular part of an on-screen window.
- virtual int printable_rect (int ∗w, int ∗h)

  Computes the width and height of the drawable area of the drawing surface.
- virtual void translate (int x, int y)

  Translates the current graphics origin accounting for the current rotation.
- virtual void untranslate ()

  Undoes the effect of a previous translate() call.

Public Member Functions inherited from Fl_Surface_Device

- Fl_Graphics_Driver ∗ **driver** ()

  Returns the graphics driver of this drawing surface.
- virtual bool is_current ()

  Is this surface the current drawing surface?
- virtual void set_current (void)

  Make this surface the current drawing surface.
- virtual ∼**Fl_Surface_Device** ()

  The destructor.

Static Public Attributes

- static const page_format **page_formats** [NO_PAGE_FORMATS]

  width, height and name of all elements of the enum Page_Format.

Protected Member Functions

- **Fl_Paged_Device** ()

  The constructor.

Protected Member Functions inherited from Fl_Widget_Surface

- Fl_Widget_Surface (Fl_Graphics_Driver *d)

  The constructor.

Protected Member Functions inherited from Fl_Surface_Device

- void **driver** (Fl_Graphics_Driver *graphics_driver)

  Sets the graphics driver of this drawing surface.
- virtual void end_current ()

  FLTK calls this each time a surface ceases to be the current drawing surface.
- **Fl_Surface_Device** (Fl_Graphics_Driver *graphics_driver)

  Constructor that sets the graphics driver to use for the created surface.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Surface_Device

- static Fl_Surface_Device * pop_current ()

  Removes the top element from the current drawing surface stack, and makes the new top element current.
- static void push_current (Fl_Surface_Device *new_current)

  Pushes `new_current` on top of the stack of current drawing surfaces, and makes it current.
- static Fl_Surface_Device * surface ()

  The current drawing surface.

Protected Attributes inherited from Fl_Widget_Surface

- int **x_offset**

  horizontal offset to the origin of graphics coordinates
- int **y_offset**

  vertical offset to the origin of graphics coordinates

13.92.1 Detailed Description

Represents page-structured drawing surfaces.
This class has no public constructor: don't instantiate it; use Fl_Printer or Fl_PostScript_File_Device instead.

13.92.2 Member Enumeration Documentation

13.92.2.1 Page_Format

`enum Fl_Paged_Device::Page_Format`
Possible page formats.
All paper formats with pre-defined width and height. The Fl_Paged_Device::page_formats array gives these widths and heights.

Enumerator

A0	A0 format.
A1	A1 format.
A2	A2 format.
A3	A3 format.

Enumerator

A4	A4 format.
A5	A5 format.
A6	A6 format.
A7	A7 format.
A8	A8 format.
A9	A9 format.
B0	B0 format.
B1	B1 format.
B2	B2 format.
B3	B3 format.
B4	B4 format.
B5	B5 format.
B6	B6 format.
B7	B7 format.
B8	B8 format.
B9	B9 format.
B10	B10 format.
EXECUTIVE	Executive format.
FOLIO	Folio format.
LEDGER	Ledger format.
LEGAL	Legal format.
LETTER	Letter format.
TABLOID	Tabloid format.

13.92.2.2 Page_Layout

`enum Fl_Paged_Device::Page_Layout`

Possible page layouts.

Enumerator

PORTRAIT	Portrait orientation.
LANDSCAPE	Landscape orientation.
REVERSED	Reversed orientation.
ORIENTATION	orientation

13.92.3 Member Function Documentation

13.92.3.1 begin_job()

```
int Fl_Paged_Device::begin_job (
            int pagecount = 0,
            int * frompage = NULL,
            int * topage = NULL,
            char ** perr_message = NULL )  [virtual]
```

Begins a print job.

Parameters

in	*pagecount*	the total number of pages of the job (or 0 if you don't know the number of pages)
out	*frompage*	if non-null, *frompage is set to the first page the user wants printed

Parameters

out	*topage*	if non-null, *topage is set to the last page the user wants printed
out	*perr_message*	if non-null and if the returned value is 2, *perr_message is set to a string describing the error. That string can be delete[]'d after use.

Returns

0 if OK, 1 if user cancelled the job, 2 if any error.

Reimplemented in Fl_PostScript_File_Device, Fl_Printer, and Fl_PDF_File_Surface.

13.92.3.2 begin_page()

```
int Fl_Paged_Device::begin_page (
            void ) [virtual]
```

Begins a new printed page.
The page coordinates are initially in points, i.e., 1/72 inch, and with origin at the top left of the printable page area.
This function also makes this surface the current drawing surface with Fl_Surface_Device::push_current().

Note

begin_page() calls Fl_Surface_Device::push_current() and leaves this device as the active surface. If any calls between begin_page() and end_page() open dialog boxes or will otherwise draw into FLTK windows, those calls must be put between a call to Fl_Surface_Device::pop_current() and a call to Fl_Surface_Device::push_current(), or the content of the dialog box will be rendered to the printer instead of the screen.

Returns

0 if OK, non-zero if any error

Reimplemented in Fl_PDF_File_Surface, Fl_PostScript_File_Device, and Fl_Printer.

13.92.3.3 end_job()

```
void Fl_Paged_Device::end_job (
            void ) [virtual]
```

To be called at the end of a print job.
Reimplemented in Fl_PDF_File_Surface, Fl_PostScript_File_Device, and Fl_Printer.

13.92.3.4 end_page()

```
int Fl_Paged_Device::end_page (
            void ) [virtual]
```

To be called at the end of each page.
This function also stops this surface from being the current drawing surface with Fl_Surface_Device::pop_current().

Note

end_page() calls Fl_Surface_Device::pop_current(). If any calls between begin_page() and end_page() open dialog boxes or will otherwise draw into FLTK windows, those calls must be put between a call to Fl_Surface_Device::pop_current() and a call to Fl_Surface_Device::push_current().

Returns

0 if OK, non-zero if any error.

Reimplemented in Fl_PDF_File_Surface, Fl_PostScript_File_Device, and Fl_Printer.

13.92.3.5 margins()

```
void Fl_Paged_Device::margins (
            int * left,
            int * top,
            int * right,
            int * bottom )  [virtual]
```

Computes the dimensions of margins that lie between the printable page area and the full page.
Values are in the same unit as that used by FLTK drawing functions. They are changed by scale() calls.

Parameters

out	*left*	If non-null, *left is set to the left margin size.
out	*top*	If non-null, *top is set to the top margin size.
out	*right*	If non-null, *right is set to the right margin size.
out	*bottom*	If non-null, *bottom is set to the bottom margin size.

Reimplemented in Fl_PDF_File_Surface, Fl_PostScript_File_Device, and Fl_Printer.

13.92.3.6 rotate()

```
void Fl_Paged_Device::rotate (
            float angle )  [virtual]
```

Rotates the graphics operations relatively to paper.
The rotation is centered on the current graphics origin. Successive rotate() calls don't combine their effects.

Parameters

angle	Rotation angle in counter-clockwise degrees.

Reimplemented in Fl_PDF_File_Surface, Fl_PostScript_File_Device, and Fl_Printer.

13.92.3.7 scale()

```
void Fl_Paged_Device::scale (
            float scale_x,
            float scale_y = 0.  )  [virtual]
```

Changes the scaling of page coordinates.
This function also resets the origin of graphics functions at top left of printable page area. After a scale() call, do a printable_rect() call to get the new dimensions of the printable page area. Successive scale() calls don't combine their effects.

Parameters

scale↩ _x	Horizontal dimensions of plot are multiplied by this quantity.
scale↩ _y	Same as above, vertically. The value 0. is equivalent to setting `scale_y` = `scale_x`. Thus, scale(factor); is equivalent to scale(factor, factor);

Reimplemented in Fl_PDF_File_Surface, Fl_PostScript_File_Device, and Fl_Printer.

13.92.3.8 start_job()

```
int Fl_Paged_Device::start_job (
            int pagecount = 0,
            int * frompage = NULL,
            int * topage = NULL,
            char ** perr_message = NULL )  [inline]
```

Synonym of begin_job(int pagecount, int *frompage, int *topage, char **perr_message).
For API compatibility with FLTK 1.3.x

13.92.3.9 start_page()

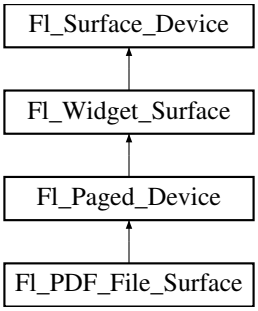
```
int Fl_Paged_Device::start_page ( ) [inline]
```

Synonym of begin_page().
For API compatibility with FLTK 1.3.x
The documentation for this class was generated from the following files:

- Fl_Paged_Device.H
- Fl_Paged_Device.cxx

13.93 Fl_PDF_File_Surface Class Reference

To send graphical output to a PDF file.
`#include <Fl_PDF_File_Surface.H>`
Inheritance diagram for Fl_PDF_File_Surface:



Public Member Functions

- int begin_document (const char *pathname, enum Fl_Paged_Device::Page_Format format=Fl_Paged_Device::A4, enum Fl_Paged_Device::Page_Layout layout=Fl_Paged_Device::PORTRAIT, char **perr=NULL)

  Prepare to draw to a PDF document identified by its pathname.
- int begin_job (const char *defaultfilename, char **perr=NULL)

  Prepare to draw to a PDF document identified with a file chooser.
- int begin_job (int, int *, int *, char **) override

  Don't use for this class.
- int begin_page (void) override

  Begins a new printed page.
- void end_job (void) override

  To be called at the end of a print job.
- int end_page (void) override

  To be called at the end of each page.
- bool is_current () override

  Is this surface the current drawing surface?
- void margins (int *left, int *top, int *right, int *bottom) override

  Computes the dimensions of margins that lie between the printable page area and the full page.
- void origin (int *x, int *y) override

  Computes the coordinates of the current origin of graphics functions.
- void origin (int x, int y) override

  Sets the position of the origin of graphics in the drawable part of the drawing surface.
- const char * **pdf_filename** () const

Returns the name of the PDF document.

- int printable_rect (int ∗w, int ∗h) override

  Computes the width and height of the drawable area of the drawing surface.
- void rotate (float angle) override

  Rotates the graphics operations relatively to paper.
- void scale (float s_x, float s_y=0) override

  Changes the scaling of page coordinates.
- void set_current () override

  Make this surface the current drawing surface.
- void translate (int x, int y) override

  Translates the current graphics origin accounting for the current rotation.
- void untranslate () override

  Undoes the effect of a previous translate() call.

Public Member Functions inherited from Fl_Paged_Device

- void **print_widget** (Fl_Widget ∗widget, int delta_x=0, int delta_y=0)

  Synonym of draw(Fl_Widget∗, int, int)
- void **print_window** (Fl_Window ∗win, int x_off=0, int y_off=0)

  Synonym of draw_decorated_window(Fl_Window∗, int, int)
- int start_job (int pagecount=0, int ∗frompage=NULL, int ∗topage=NULL, char ∗∗perr_message=NULL)

  Synonym of begin_job(int pagecount, int ∗frompage, int ∗topage, char ∗∗perr_message).
- int start_page ()

  Synonym of begin_page().
- virtual ∼**Fl_Paged_Device** ()

  The destructor.

Public Member Functions inherited from Fl_Widget_Surface

- void draw (Fl_Widget ∗widget, int delta_x=0, int delta_y=0)

  Draws the widget on the drawing surface.
- void draw_decorated_window (Fl_Window ∗win, int x_offset=0, int y_offset=0)

  Draws a window with its title bar and frame if any.
- void print_window_part (Fl_Window ∗win, int x, int y, int w, int h, int delta_x=0, int delta_y=0)

  Draws a rectangular part of an on-screen window.

Public Member Functions inherited from Fl_Surface_Device

- Fl_Graphics_Driver ∗ **driver** ()

  Returns the graphics driver of this drawing surface.
- virtual ∼**Fl_Surface_Device** ()

  The destructor.

Static Public Attributes

These attributes are useful for the Wayland/X11 platform only.

- static const char ∗ **format_dialog_title** = "PDF document settings"

  Localizable text of the "PDF document settings" dialog.
- static const char ∗ **format_dialog_page_size** = "Page Size:"

  Localizable text of the "PDF document settings" dialog.
- static const char ∗ **format_dialog_orientation** = "Orientation:"

  Localizable text of the "PDF document settings" dialog.
- static const char ∗ **format_dialog_default** = "Set as default"

  Localizable text of the "PDF document settings" dialog.

Static Public Attributes inherited from Fl_Paged_Device

- static const page_format **page_formats** [NO_PAGE_FORMATS]

  width, height and name of all elements of the enum Page_Format.

Additional Inherited Members

Public Types inherited from Fl_Paged_Device

- enum Page_Format {
  A0 = 0 , A1 , A2 , A3 ,
  A4 , A5 , A6 , A7 ,
  A8 , A9 , B0 , B1 ,
  B2 , B3 , B4 , B5 ,
  B6 , B7 , B8 , B9 ,
  B10 , **C5E** , **DLE** , EXECUTIVE ,
  FOLIO , LEDGER , LEGAL , LETTER ,
  TABLOID , **ENVELOPE** , **MEDIA** = 0x1000 }

  Possible page formats.
- enum Page_Layout { PORTRAIT = 0 , LANDSCAPE = 0x100 , REVERSED = 0x200 , ORIENTATION = 0x300 }

  Possible page layouts.

Static Public Member Functions inherited from Fl_Surface_Device

- static Fl_Surface_Device * pop_current ()

  Removes the top element from the current drawing surface stack, and makes the new top element current.
- static void push_current (Fl_Surface_Device *new_current)

  Pushes `new_current` on top of the stack of current drawing surfaces, and makes it current.
- static Fl_Surface_Device * surface ()

  The current drawing surface.

Protected Member Functions inherited from Fl_Paged_Device

- **Fl_Paged_Device** ()

  The constructor.

Protected Member Functions inherited from Fl_Widget_Surface

- Fl_Widget_Surface (Fl_Graphics_Driver *d)

  The constructor.

Protected Member Functions inherited from Fl_Surface_Device

- void **driver** (Fl_Graphics_Driver *graphics_driver)

  Sets the graphics driver of this drawing surface.
- virtual void end_current ()

  FLTK calls this each time a surface ceases to be the current drawing surface.
- **Fl_Surface_Device** (Fl_Graphics_Driver *graphics_driver)

  Constructor that sets the graphics driver to use for the created surface.

Protected Attributes inherited from Fl_Widget_Surface

- int **x_offset**

  horizontal offset to the origin of graphics coordinates
- int **y_offset**

  vertical offset to the origin of graphics coordinates

13.93.1 Detailed Description

To send graphical output to a PDF file.
Class Fl_PDF_File_Surface is used exactly as the Fl_Printer class except for its 2 member functions begin_job() and begin_document().
Platform notes:

- Windows: requires "Microsoft Print to PDF" available in Windows 10 and later.
- Wayland/X11: requires the FLTK library was built with FLTK_USE_PANGO=1.
- macOS: requires macOS 10.9 or later.

If the running platform doesn't fulfill the requirement above, the program runs but doesn't output any PDF.

13.93.2 Member Function Documentation

13.93.2.1 begin_document()

```
int Fl_PDF_File_Surface::begin_document (
            const char * pathname,
            enum Fl_Paged_Device::Page_Format format = Fl_Paged_Device::A4,
            enum Fl_Paged_Device::Page_Layout layout = Fl_Paged_Device::PORTRAIT,
            char ** perr = NULL )
```

Prepare to draw to a PDF document identified by its pathname.

Parameters

pathname	Path name for the PDF document
format	The paper format for the PDF document
layout	The orientation for the PDF document
perr	NULL or address of a string that receives a message in case of error. To be deleted[] after use.

Returns

0 for success, 2 when an error occurred.

13.93.2.2 begin_job() [1/2]

```
int Fl_PDF_File_Surface::begin_job (
            const char * defaultfilename,
            char ** perr = NULL )
```

Prepare to draw to a PDF document identified with a file chooser.
A dialog opens to select the location and name of the output PDF document as well as its page format and orientation.

Parameters

defaultfilename	Default name for the PDF document
perr	NULL or address of a string that receives a message in case of error. To be deleted[] after use.

Returns

0 for success, 1 when the user cancelled the operation, 2 when an error occurred.

13.93.2.3 begin_job() [2/2]

```
int Fl_PDF_File_Surface::begin_job (
            int ,
```

```
            int * ,
            int * ,
            char **  )  [inline], [override], [virtual]
```

Don't use for this class.
Reimplemented from Fl_Paged_Device.

13.93.2.4 begin_page()

```
int Fl_PDF_File_Surface::begin_page (
            void  )  [inline], [override], [virtual]
```

Begins a new printed page.
The page coordinates are initially in points, i.e., 1/72 inch, and with origin at the top left of the printable page area.
This function also makes this surface the current drawing surface with Fl_Surface_Device::push_current().

Note

begin_page() calls Fl_Surface_Device::push_current() and leaves this device as the active surface. If any calls between begin_page() and end_page() open dialog boxes or will otherwise draw into FLTK windows, those calls must be put between a call to Fl_Surface_Device::pop_current() and a call to Fl_Surface_Device::push_current(), or the content of the dialog box will be rendered to the printer instead of the screen.

Returns

0 if OK, non-zero if any error

Reimplemented from Fl_Paged_Device.

13.93.2.5 end_job()

```
void Fl_PDF_File_Surface::end_job (
            void  )  [inline], [override], [virtual]
```

To be called at the end of a print job.
Reimplemented from Fl_Paged_Device.

13.93.2.6 end_page()

```
int Fl_PDF_File_Surface::end_page (
            void  )  [inline], [override], [virtual]
```

To be called at the end of each page.
This function also stops this surface from being the current drawing surface with Fl_Surface_Device::pop_current().

Note

end_page() calls Fl_Surface_Device::pop_current(). If any calls between begin_page() and end_page() open dialog boxes or will otherwise draw into FLTK windows, those calls must be put between a call to Fl_Surface_Device::pop_current() and a call to Fl_Surface_Device::push_current().

Returns

0 if OK, non-zero if any error.

Reimplemented from Fl_Paged_Device.

13.93.2.7 is_current()

```
bool Fl_PDF_File_Surface::is_current ( )  [inline], [override], [virtual]
```

Is this surface the current drawing surface?
Reimplemented from Fl_Surface_Device.

13.93.2.8 margins()

```
void Fl_PDF_File_Surface::margins (
            int * left,
            int * top,
            int * right,
            int * bottom )  [inline], [override], [virtual]
```

Computes the dimensions of margins that lie between the printable page area and the full page.
Values are in the same unit as that used by FLTK drawing functions. They are changed by scale() calls.

Parameters

out	*left*	If non-null, ∗left is set to the left margin size.
out	*top*	If non-null, ∗top is set to the top margin size.
out	*right*	If non-null, ∗right is set to the right margin size.
out	*bottom*	If non-null, ∗bottom is set to the bottom margin size.

Reimplemented from Fl_Paged_Device.

13.93.2.9 origin() [1/2]

```
void Fl_PDF_File_Surface::origin (
            int * x,
            int * y )  [inline], [override], [virtual]
```

Computes the coordinates of the current origin of graphics functions.

Parameters

out	*x,y*	If non-null, ∗x and ∗y are set to the horizontal and vertical coordinates of the graphics origin.

Reimplemented from Fl_Widget_Surface.

13.93.2.10 origin() [2/2]

```
void Fl_PDF_File_Surface::origin (
            int x,
            int y )  [inline], [override], [virtual]
```

Sets the position of the origin of graphics in the drawable part of the drawing surface.
Arguments should be expressed relatively to the result of a previous printable_rect() call. That is, `printable_rect(&w, &h); origin(w/2, 0);` sets the graphics origin at the top center of the drawable area. Successive origin() calls don't combine their effects. Origin() calls are not affected by rotate() calls (for classes derived from Fl_Paged_Device).

Parameters

in	*x,y*	Horizontal and vertical positions in the drawing surface of the desired origin of graphics.

Reimplemented from Fl_Widget_Surface.

13.93.2.11 printable_rect()

```
int Fl_PDF_File_Surface::printable_rect (
            int * w,
            int * h )  [inline], [override], [virtual]
```

Computes the width and height of the drawable area of the drawing surface.
Values are in the same unit as that used by FLTK drawing functions and are unchanged by calls to origin(). If the object is derived from class Fl_Paged_Device, values account for the user-selected paper type and print orientation and are changed by scale() calls.

Returns

0 if OK, non-zero if any error

Reimplemented from Fl_Widget_Surface.

13.93.2.12 rotate()

```
void Fl_PDF_File_Surface::rotate (
            float angle )  [inline], [override], [virtual]
```

Rotates the graphics operations relatively to paper.
The rotation is centered on the current graphics origin. Successive rotate() calls don't combine their effects.

Parameters

angle	Rotation angle in counter-clockwise degrees.

Reimplemented from Fl_Paged_Device.

13.93.2.13 scale()

```
void Fl_PDF_File_Surface::scale (
            float scale_x,
            float scale_y = 0 )  [inline], [override], [virtual]
```

Changes the scaling of page coordinates.
This function also resets the origin of graphics functions at top left of printable page area. After a scale() call, do a printable_rect() call to get the new dimensions of the printable page area. Successive scale() calls don't combine their effects.

Parameters

scale↩ _x	Horizontal dimensions of plot are multiplied by this quantity.
scale↩ _y	Same as above, vertically. The value 0. is equivalent to setting `scale_y` = `scale_x`. Thus, scale(factor); is equivalent to scale(factor, factor);

Reimplemented from Fl_Paged_Device.

13.93.2.14 set_current()

```
void Fl_PDF_File_Surface::set_current (
            void )  [inline], [override], [virtual]
```

Make this surface the current drawing surface.
This surface will receive all future graphics requests.
Since FLTK 1.4.0 the preferred API to change the current drawing surface is Fl_Surface_Device::push_current( ) / Fl_Surface_Device::pop_current().

Note

It is recommended to use this function only as follows :

- The current drawing surface is the display;
- make current another surface, e.g., an Fl_Printer or an Fl_Image_Surface object, calling set_current() on this object;
- draw to that surface;
- make the display current again with Fl_Display_Device::display_device()->set_current();
  don't do any other call to set_current() before this one.

Other scenarios of drawing surface changes should be performed via Fl_Surface_Device::push_current() and Fl_Surface_Device::pop_current().
Reimplemented from Fl_Surface_Device.

13.93.2.15 translate()

```
void Fl_PDF_File_Surface::translate (
            int x,
            int y )  [inline], [override], [virtual]
```

Translates the current graphics origin accounting for the current rotation.
Each translate() call must be matched by an untranslate() call. Successive translate() calls add up their effects.
Reimplemented from Fl_Widget_Surface.

13.93.2.16 untranslate()

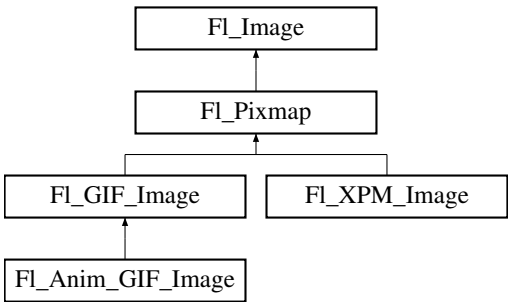
```
void Fl_PDF_File_Surface::untranslate (
            void  )  [inline], [override], [virtual]
```

Undoes the effect of a previous translate() call.
Reimplemented from Fl_Widget_Surface.
The documentation for this class was generated from the following files:

- Fl_PDF_File_Surface.H
- Fl_Device.cxx

13.94 Fl_Pixmap Class Reference

The Fl_Pixmap class supports caching and drawing of colormap (pixmap) images, including transparency.
`#include <Fl_Pixmap.H>`
Inheritance diagram for Fl_Pixmap:



Public Member Functions

- int **cache_h** () const
- int **cache_w** () const
- void color_average (Fl_Color c, float i) override

  The color_average() method averages the colors in the image with the provided FLTK color value.
- Fl_Image ∗ **copy** () const
- Fl_Image ∗ copy (int W, int H) const override

  Creates a resized copy of the image.
- void desaturate () override

  The desaturate() method converts an image to grayscale.
- void **draw** (int X, int Y)
- void draw (int X, int Y, int W, int H, int cx=0, int cy=0) override

  Draws the image to the current drawing surface with a bounding box.
- Fl_Pixmap (char ∗const ∗D)

  The constructors create a new pixmap from the specified XPM data.
- Fl_Pixmap (const char ∗const ∗D)

  The constructors create a new pixmap from the specified XPM data.
- Fl_Pixmap (const uchar ∗const ∗D)

The constructors create a new pixmap from the specified XPM data.

- Fl_Pixmap (uchar *const *D)

  The constructors create a new pixmap from the specified XPM data.
- void label (Fl_Menu_Item *m) override

  This method is an obsolete way to set the image attribute of a menu item.
- void label (Fl_Widget *w) override

  This method is an obsolete way to set the image attribute of a widget or menu item.
- void uncache () override

  If the image has been cached for display, delete the cache data.
- virtual ~**Fl_Pixmap** ()

  The destructor frees all memory and server resources that are used by the pixmap.

Public Member Functions inherited from Fl_Image

- virtual class Fl_Shared_Image * as_shared_image ()

  Returns whether an image is an Fl_Shared_Image or not.
- Fl_Image * copy () const

  Creates a copy of the image in the same size.
- int count () const

  Returns the number of data values associated with the image.
- int d () const

  Returns the image depth.
- const char *const * data () const

  Returns a pointer to the current image data array.
- int **data_h** () const

  Returns the height of the image data.
- int **data_w** () const

  Returns the width of the image data.
- void draw (int X, int Y)

  Draws the image to the current drawing surface.
- int fail () const

  Returns a value that is not 0 if there is currently no image available.
- Fl_Image (int W, int H, int D)

  The constructor creates an empty image with the specified width, height, and depth.
- int h () const

  Returns the current image drawing height in FLTK units.
- void inactive ()

  The inactive() method calls color_average(FL_BACKGROUND_COLOR, 0.33f) to produce an image that appears grayed out.
- int ld () const

  Returns the current line data size in bytes.
- virtual void release ()

  Releases an Fl_Image - the same as '`delete this`'.
- virtual void scale (int width, int height, int proportional=1, int can_expand=0)

  Sets the drawing size of the image.
- int w () const

  Returns the current image drawing width in FLTK units.
- virtual ~**Fl_Image** ()

  The destructor is a virtual method that frees all memory used by the image.

Public Attributes

- int **alloc_data**

Protected Member Functions

- void **measure** ()

Protected Member Functions inherited from Fl_Image

- void **d** (int D)

  Sets the current image depth.
- void data (const char ∗const ∗p, int c)

  Sets the current data pointer and count of pointers in the array.
- void draw_empty (int X, int Y)

  The protected method draw_empty() draws a box with an X in it.
- int draw_scaled (int X, int Y, int W, int H)

  Draw the image to the current drawing surface rescaled to a given width and height.
- void h (int H)

  Sets the height of the image data.
- void ld (int LD)

  Sets the current line data size in bytes.
- void w (int W)

  Sets the width of the image data.

Friends

- class **Fl_Graphics_Driver**

Additional Inherited Members

Static Public Member Functions inherited from Fl_Image

- static Fl_Labeltype **define_FL_IMAGE_LABEL** ()
- static Fl_RGB_Scaling **RGB_scaling** ()

  Returns the currently used RGB image scaling method.
- static void RGB_scaling (Fl_RGB_Scaling)

  Sets the RGB image scaling method used for copy(int, int).
- static Fl_RGB_Scaling **scaling_algorithm** ()

  Gets what algorithm is used when resizing a source image to draw it.
- static void scaling_algorithm (Fl_RGB_Scaling algorithm)

  Sets what algorithm is used when resizing a source image to draw it.

Static Public Attributes inherited from Fl_Image

- static const int **ERR_FILE_ACCESS** = -2
- static const int **ERR_FORMAT** = -3
- static const int **ERR_MEMORY_ACCESS** = -4
- static const int **ERR_NO_IMAGE** = -1
- static bool **register_images_done** = false

  True after fl_register_images() was called, false before.

Static Protected Member Functions inherited from Fl_Image

- static void **labeltype** (const Fl_Label ∗lo, int lx, int ly, int lw, int lh, Fl_Align la)
- static void **measure** (const Fl_Label ∗lo, int &lw, int &lh)

13.94.1 Detailed Description

The Fl_Pixmap class supports caching and drawing of colormap (pixmap) images, including transparency.

13.94.2 Constructor & Destructor Documentation

13.94.2.1 Fl_Pixmap() [1/4]

```
Fl_Pixmap::Fl_Pixmap (
            char *const * D )  [inline], [explicit]
```

The constructors create a new pixmap from the specified XPM data.

13.94.2.2 Fl_Pixmap() [2/4]

```
Fl_Pixmap::Fl_Pixmap (
            uchar *const * D )  [inline], [explicit]
```

The constructors create a new pixmap from the specified XPM data.

13.94.2.3 Fl_Pixmap() [3/4]

```
Fl_Pixmap::Fl_Pixmap (
            const char *const * D )  [inline], [explicit]
```

The constructors create a new pixmap from the specified XPM data.

13.94.2.4 Fl_Pixmap() [4/4]

```
Fl_Pixmap::Fl_Pixmap (
            const uchar *const * D )  [inline], [explicit]
```

The constructors create a new pixmap from the specified XPM data.

13.94.3 Member Function Documentation

13.94.3.1 color_average()

```
void Fl_Pixmap::color_average (
            Fl_Color c,
            float i )  [override], [virtual]
```

The color_average() method averages the colors in the image with the provided FLTK color value.
The first argument specifies the FLTK color to be used.
The second argument specifies the amount of the original image to combine with the color, so a value of 1.0 results in no color blend, and a value of 0.0 results in a constant image of the specified color.
An internal copy is made of the original image data before changes are applied, to avoid modifying the original image data in memory.
Reimplemented from Fl_Image.

13.94.3.2 copy()

```
Fl_Image * Fl_Pixmap::copy (
            int W,
            int H ) const  [override], [virtual]
```

Creates a resized copy of the image.
It is recommended not to call this member function to reduce the size of an image to the size of the area where this image will be drawn, and to use Fl_Image::scale() instead.
The new image should be released when you are done with it.
Note: since FLTK 1.4.0 you can use Fl_Image::release() for all types of images (i.e. all subclasses of Fl_Image) instead of operator *delete* for Fl_Image's and Fl_Image::release() for Fl_Shared_Image's.

The new image data will be converted to the requested size. RGB images are resized using the algorithm set by Fl_Image::RGB_scaling().
For the new image the following equations are true:

- w() == data_w() == W
- h() == data_h() == H

Parameters

in	*W,H*	Requested width and height of the new image

Note

The returned image can be safely cast to the same image type as that of the source image provided this type is one of Fl_RGB_Image, Fl_SVG_Image, Fl_Pixmap, Fl_Bitmap, Fl_Tiled_Image, Fl_Anim_GIF_Image and Fl_Shared_Image. Returned objects copied from images of other, derived, image classes belong to the parent class appearing in this list. For example, the copy of an Fl_GIF_Image is an object of class Fl_Pixmap.

Since FLTK 1.4.0 this method is 'const'. If you derive your own class from Fl_Image or any subclass your overridden methods of 'Fl_Image::copy() const' and 'Fl_Image::copy(int, int) const' **must** also be 'const' for inheritance to work properly. This is different than in FLTK 1.3.x and earlier where these methods have not been 'const'.

Reimplemented from Fl_Image.

13.94.3.3 desaturate()

```
void Fl_Pixmap::desaturate ( ) [override], [virtual]
```

The desaturate() method converts an image to grayscale.
If the image contains an alpha channel (depth = 4), the alpha channel is preserved.
An internal copy is made of the original image data before changes are applied, to avoid modifying the original image data in memory.
Reimplemented from Fl_Image.

13.94.3.4 draw()

```
void Fl_Pixmap::draw (
            int X,
            int Y,
            int W,
            int H,
            int cx = 0,
            int cy = 0 ) [override], [virtual]
```

Draws the image to the current drawing surface with a bounding box.
Arguments X,Y,W,H specify a bounding box for the image, with the origin (upper-left corner) of the image offset by the cx and cy arguments.
Another way to see what part of the image gets drawn and where, is to consider this alternative writing producing the same output:

```
fl_push_clip(X,Y,W,H);
this->draw(X - cx, Y - cy);
fl_pop_clip();
```

Repeated calls to this member function with the same image but varying W, H, cx, or cy arguments can be more efficiently processed using the above alternative writing.
Reimplemented from Fl_Image.

13.94.3.5 label() [1/2]

```
void Fl_Pixmap::label (
            Fl_Menu_Item * m )  [override], [virtual]
```

This method is an obsolete way to set the image attribute of a menu item.

Deprecated Please use Fl_Menu_Item::image() instead.

Reimplemented from Fl_Image.

13.94.3.6 label() [2/2]

```
void Fl_Pixmap::label (
            Fl_Widget * widget )  [override], [virtual]
```

This method is an obsolete way to set the image attribute of a widget or menu item.

Deprecated Please use Fl_Widget::image() or Fl_Widget::deimage() instead.

Reimplemented from Fl_Image.

13.94.3.7 uncache()

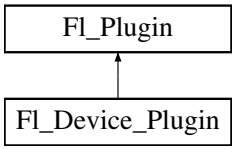
```
void Fl_Pixmap::uncache ( )  [override], [virtual]
```

If the image has been cached for display, delete the cache data.
This allows you to change the data used for the image and then redraw it without recreating an image object.
Reimplemented from Fl_Image.
The documentation for this class was generated from the following files:

- Fl_Pixmap.H
- Fl_Pixmap.cxx

13.95 Fl_Plugin Class Reference

Fl_Plugin allows link-time and run-time integration of binary modules.
`#include <Fl_Plugin.H>`
Inheritance diagram for Fl_Plugin:



Public Member Functions

- Fl_Plugin (const char *klass, const char *name)

  Create a plugin.
- virtual ~**Fl_Plugin** ()

  Clear the plugin and remove it from the database.

13.95.1 Detailed Description

Fl_Plugin allows link-time and run-time integration of binary modules.
Fl_Plugin and Fl_Plugin_Manager provide a small and simple solution for linking C++ classes at run-time, or optionally linking modules at compile time without the need to change the main application.
Fl_Plugin_Manager uses static initialization to create the plugin interface early during startup. Plugins are stored in a temporary database, organized in classes.
Plugins should derive a new class from Fl_Plugin as a base:

```
class My_Plugin : public Fl_Plugin {
```

```
public:
  My_Plugin() : Fl_Plugin("effects", "blur") { }
  void do_something(...);
};
My_Plugin blur_plugin();
```

Plugins can be put into modules and either linked before distribution, or loaded from dynamically linkable files. An Fl_Plugin_Manager is used to list and access all currently loaded plugins.

```
Fl_Plugin_Manager mgr("effects");
int i, n = mgr.plugins();
for (i=0; i<n; i++) {
  My_Plugin *pin = (My_Plugin*)mgr.plugin(i);
  pin->do_something();
}
```

13.95.2 Constructor & Destructor Documentation

13.95.2.1 Fl_Plugin()

```
Fl_Plugin::Fl_Plugin (
            const char * klass,
            const char * name )
```

Create a plugin.

Parameters

in	*klass*	plugins are grouped in classes
in	*name*	every plugin should have a unique name

The documentation for this class was generated from the following files:

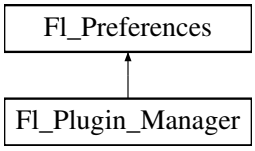
- Fl_Plugin.H
- Fl_Preferences.cxx

13.96 Fl_Plugin_Manager Class Reference

Fl_Plugin_Manager manages link-time and run-time plugin binaries.

`#include <Fl_Plugin.H>`

Inheritance diagram for Fl_Plugin_Manager:



Public Member Functions

- Fl_Preferences::ID addPlugin (const char *name, Fl_Plugin *plugin)

  This function adds a new plugin to the database.
- **Fl_Plugin_Manager** (const char *klass)

  Manage all plugins belonging to one class.
- Fl_Plugin * **plugin** (const char *name)

  Return the address of a plugin by name.
- Fl_Plugin * **plugin** (int index)

  Return the address of a plugin by index.
- int **plugins** ()

  Return the number of plugins in the klass.
- ~Fl_Plugin_Manager ()

  Remove the plugin manager.

Public Member Functions inherited from Fl_Preferences

- char **clear** ()

  Delete all groups and all entries.
- char **delete_all_entries** ()

  Delete all entries.
- char **delete_all_groups** ()

  Delete all groups.
- char delete_entry (const char *entry)

  Deletes a single name/value pair.
- char delete_group (const char *group)

  Deletes a group.
- int dirty ()

  Check if there were changes to the database that need to be written to disk.
- int entries ()

  Returns the number of entries (name/value pairs) in a group.
- const char * entry (int index)

  Returns the name of an entry.
- char entry_exists (const char *key)

  Returns non-zero if an entry with this name exists.
- Root filename (char *buffer, size_t buffer_size)

  Return the file name and path to the preference file.
- Fl_Preferences (const char *path, const char *vendor, const char *application)

  Deprecated: Use this constructor to create or read a preference file at an arbitrary position in the file system.
- Fl_Preferences (const char *path, const char *vendor, const char *application, Root flags)

  Use this constructor to create or read a preference file at an arbitrary position in the file system.
- **Fl_Preferences** (const Fl_Preferences &)

  Create another reference to a Preferences group.
- Fl_Preferences (Fl_Preferences &parent, const char *group)

  Generate or read a new group of entries within another group.
- Fl_Preferences (Fl_Preferences &parent, int groupIndex)

  Open a child group using a given index.
- Fl_Preferences (Fl_Preferences *parent, const char *group)

  Create or access a group of preferences using a name.
- Fl_Preferences (Fl_Preferences *parent, int groupIndex)
- Fl_Preferences (ID id)

  Create a new dataset access point using a dataset ID.
- Fl_Preferences (Root root, const char *vendor, const char *application)

  The constructor creates a group that manages key/value pairs and child groups.
- int flush ()

  Writes preferences to disk if they were modified.
- char get (const char *entry, char *&value, const char *defaultValue)

  Reads an entry from the group.
- char get (const char *entry, char *value, const char *defaultValue, int maxSize)

  Reads an entry from the group.
- char get (const char *entry, double &value, double defaultValue)

  Reads an entry from the group.
- char get (const char *entry, float &value, float defaultValue)

  Reads an entry from the group.
- char get (const char *entry, int &value, int defaultValue)

  Reads an entry from the group.

- char get (const char *entry, std::string &value, const std::string &defaultValue)

  Reads an entry from the group.
- char get (const char *entry, void *&value, const void *defaultValue, int defaultSize)

  Reads an entry from the group.
- char get (const char *entry, void *value, const void *defaultValue, int defaultSize, int *size)

  Reads a binary entry from the group, encoded in hexadecimal blocks.
- char get (const char *entry, void *value, const void *defaultValue, int defaultSize, int maxSize)

  Reads a binary entry from the group, encoded in hexadecimal blocks.
- char get_userdata_path (char *path, int pathlen)

  Creates a path that is related to the preference file and that is usable for additional application data.
- char **get_userdata_path** (std::string &path)
- const char * group (int num_group)

  Returns the name of the Nth (`num_group`) group.
- char group_exists (const char *key)

  Returns non-zero if a group with this name exists.
- int groups ()

  Returns the number of groups that are contained within a group.
- ID **id** () const

  Return an ID that can later be reused to open more references to this dataset.
- const char * **name** () const

  Return the name of this entry.
- const char * **path** () const

  Return the full path to this entry.
- char set (const char *entry, const char *value)

  Sets an entry (name/value pair).
- char set (const char *entry, const std::string &value)

  Sets an entry (name/value pair).
- char set (const char *entry, const void *value, int size)

  Sets an entry (name/value pair).
- char set (const char *entry, double value)

  Sets an entry (name/value pair).
- char set (const char *entry, double value, int precision)

  Sets an entry (name/value pair).
- char set (const char *entry, float value)

  Sets an entry (name/value pair).
- char set (const char *entry, float value, int precision)

  Sets an entry (name/value pair).
- char set (const char *entry, int value)

  Sets an entry (name/value pair).
- int size (const char *entry)

  Returns the size of the value part of an entry.
- virtual ~Fl_Preferences ()

  The destructor removes allocated resources.

Static Public Member Functions

- static std::vector< std::string > klass_list ()

  Return a list of all plugin klasses that have been registered so far.
- static int load (const char *filename)

  Load a module from disk.
- static int loadAll (const char *dirpath, const char *pattern=0)

  Use this function to load a whole directory full of modules.
- static void removePlugin (Fl_Preferences::ID id)

  Remove any plugin.

Static Public Member Functions inherited from Fl_Preferences

- static unsigned int file_access ()

  Return the current file access permissions for the FLTK preferences system.
- static void file_access (unsigned int flags)

  Tell the FLTK preferences system which files in the file system it may read, create, or write.
- static Root filename (char ∗buffer, size_t buffer_size, Root root, const char ∗vendor, const char ∗application)

  Determine the file name and path to preferences that would be opened with these parameters.
- static const char ∗ new_UUID ()

  Returns a UUID as generated by the system.
- static char **remove** (ID id_)

  Remove the group with this ID from a database.

Additional Inherited Members

Public Types inherited from Fl_Preferences

- typedef void ∗ ID

  Every Fl_Preferences-Group has a unique ID.
- enum Root {
  UNKNOWN_ROOT_TYPE = -1 , SYSTEM = 0 , USER , MEMORY ,
  ROOT_MASK = 0x00FF , CORE = 0x0100 , C_LOCALE = 0x1000 , CLEAR = 0x2000 ,
  SYSTEM_L = SYSTEM | C_LOCALE , USER_L = USER | C_LOCALE , CORE_SYSTEM_L = CORE | SYSTEM_L , CORE_USER_L = CORE | USER_L ,
  CORE_SYSTEM = CORE | SYSTEM , CORE_USER = CORE | USER }

  Define the scope of the preferences.

Static Public Attributes inherited from Fl_Preferences

- static const unsigned int **ALL** = ALL_READ_OK | ALL_WRITE_OK

  Set this to give FLTK and applications permission to read, write, and create preference files.
- static const unsigned int **ALL_READ_OK** = USER_READ_OK | SYSTEM_READ_OK | CORE_READ_OK

  Set this to allow FLTK and applications to read preference files.
- static const unsigned int **ALL_WRITE_OK** = USER_WRITE_OK | SYSTEM_WRITE_OK | CORE_WRITE_OK

  Set this to allow FLTK and applications to create and write preference files.
- static const unsigned int **APP_OK** = SYSTEM_OK | USER_OK

  Set this if it is OK for applications to read, create, and write any kind of preference files.
- static const unsigned int **CORE_OK** = CORE_READ_OK | CORE_WRITE_OK

  Set this if it is OK for FLTK to read, create, or write preference files.
- static const unsigned int CORE_READ_OK = 0x0010

  Set this if it is OK for FLTK to read preference files.
- static const unsigned int CORE_WRITE_OK = 0x0020

  Set this if it is OK for FLTK to create or write preference files.
- static const unsigned int NONE = 0x0000

  Set this if no call to Fl_Preferences shall access the file system.
- static const unsigned int **SYSTEM_OK** = SYSTEM_READ_OK | SYSTEM_WRITE_OK

  Set this if it is OK for applications to read, create, and write system wide preference files.
- static const unsigned int **SYSTEM_READ_OK** = 0x0004

  Set this if it is OK for applications to read system wide preference files.
- static const unsigned int **SYSTEM_WRITE_OK** = 0x0008

  Set this if it is OK for applications to create and write system wide preference files.
- static const unsigned int **USER_OK** = USER_READ_OK | USER_WRITE_OK

  Set this if it is OK for applications to read, create, and write user preference files.

- static const unsigned int **USER_READ_OK** = 0x0001

  Set this if it is OK for applications to read user preference files.
- static const unsigned int **USER_WRITE_OK** = 0x0002

  Set this if it is OK for applications to create and write user preference files.

Protected Attributes inherited from Fl_Preferences

- Node * **node**
- RootNode * **rootNode**

13.96.1 Detailed Description

Fl_Plugin_Manager manages link-time and run-time plugin binaries.

See also

Fl_Plugin

13.96.2 Constructor & Destructor Documentation

13.96.2.1 ~Fl_Plugin_Manager()

```
Fl_Plugin_Manager::~Fl_Plugin_Manager ( )
```

Remove the plugin manager.
Calling this does not remove the database itself or any plugins. It just removes the reference to the database.

13.96.3 Member Function Documentation

13.96.3.1 addPlugin()

```
Fl_Preferences::ID Fl_Plugin_Manager::addPlugin (
            const char * name,
            Fl_Plugin * plugin )
```

This function adds a new plugin to the database.
There is no need to call this function explicitly. Every Fl_Plugin constructor will call this function at initialization time.

13.96.3.2 klass_list()

```
std::vector< std::string > Fl_Plugin_Manager::klass_list ( )  [static]
```

Return a list of all plugin klasses that have been registered so far.
The returned strings can be used to crate a manager for the klass of plugins, which in turn can be used to list plugins for that klass.

```
auto kl = Fl_Plugin_Manager::klass_list();
for (auto &k: kl) {
  Fl_Plugin_Manager m { k.c_str() };
  std::cout « m.plugins() « "plugins have registered for klass" « k « std::endl;
}
```

Returns

a copy of a vector of strings

13.96.3.3 load()

```
int Fl_Plugin_Manager::load (
            const char * filename )  [static]
```

Load a module from disk.
A module must be a dynamically linkable file for the given operating system. When loading a module, its +init function will be called which in turn calls the constructor of all statically initialized Fl_Plugin classes and adds them to the database.

13.96.3.4 loadAll()

```
int Fl_Plugin_Manager::loadAll (
            const char * dirpath,
            const char * pattern = 0 )  [static]
```

Use this function to load a whole directory full of modules.

Parameters

dirpath	Pathname of a directory. It **must** end with the platform's directory separator character (i.e., '\' under Windows, '/' otherwise).
pattern	A filename pattern to catch all modules of interest in the targeted directory (e.g., "{∗.so,∗.dll,∗.dylib}"), or NULL to catch all files in the directory.

13.96.3.5 removePlugin()

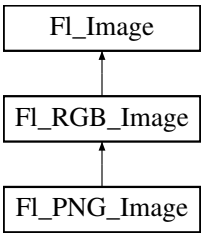
```
void Fl_Plugin_Manager::removePlugin (
            Fl_Preferences::ID id )  [static]
```

Remove any plugin.
There is no need to call this function explicitly. Every Fl_Plugin destructor will call this function at destruction time.
The documentation for this class was generated from the following files:

- Fl_Plugin.H
- Fl_Preferences.cxx

13.97 Fl_PNG_Image Class Reference

The Fl_PNG_Image class supports loading, caching, and drawing of Portable Network Graphics (PNG) image files.
`#include <Fl_PNG_Image.H>`
Inheritance diagram for Fl_PNG_Image:



Public Member Functions

- Fl_PNG_Image (const char ∗filename)

  The constructor loads the named PNG image from the given png filename.
- Fl_PNG_Image (const char ∗name_png, const unsigned char ∗buffer, int datasize)

  Constructor that reads a PNG image from memory.

Public Member Functions inherited from Fl_RGB_Image

- virtual Fl_SVG_Image ∗ as_svg_image ()

  Returns whether an RGB image is an Fl_SVG_Image or not.
- virtual const Fl_SVG_Image ∗ as_svg_image () const

  Returns whether an RGB image is an Fl_SVG_Image or not.
- int **cache_h** () const
- int **cache_w** () const
- void color_average (Fl_Color c, float i) override

The color_average() method averages the colors in the image with the provided FLTK color value.

- Fl_Image ∗ **copy** () const
- Fl_Image ∗ copy (int W, int H) const override

  Creates a resized copy of the image.
- void desaturate () override

  The desaturate() method converts an image to grayscale.
- void **draw** (int X, int Y)
- void draw (int X, int Y, int W, int H, int cx=0, int cy=0) override

  Draws the image to the current drawing surface with a bounding box.
- Fl_RGB_Image (const Fl_Pixmap ∗pxm, Fl_Color bg=FL_GRAY)

  The constructor creates a new RGBA image from the specified Fl_Pixmap.
- Fl_RGB_Image (const uchar ∗bits, int bits_length, int W, int H, int D, int LD)

  The constructor creates a new image from the specified data.
- Fl_RGB_Image (const uchar ∗bits, int W, int H, int D=3, int LD=0)

  The constructor creates a new image from the specified data.
- void label (Fl_Menu_Item ∗m) override

  This method is an obsolete way to set the image attribute of a menu item.
- void label (Fl_Widget ∗w) override

  This method is an obsolete way to set the image attribute of a widget or menu item.
- virtual void normalize ()

  Makes sure the object is fully initialized.
- void uncache () override

  If the image has been cached for display, delete the cache data.
- ∼**Fl_RGB_Image** () override

  The destructor frees all memory and server resources that are used by the image.

Public Member Functions inherited from Fl_Image

- virtual class Fl_Shared_Image ∗ as_shared_image ()

  Returns whether an image is an Fl_Shared_Image or not.
- Fl_Image ∗ copy () const

  Creates a copy of the image in the same size.
- int count () const

  Returns the number of data values associated with the image.
- int d () const

  Returns the image depth.
- const char ∗const ∗ data () const

  Returns a pointer to the current image data array.
- int **data_h** () const

  Returns the height of the image data.
- int **data_w** () const

  Returns the width of the image data.
- void draw (int X, int Y)

  Draws the image to the current drawing surface.
- int fail () const

  Returns a value that is not 0 if there is currently no image available.
- Fl_Image (int W, int H, int D)

  The constructor creates an empty image with the specified width, height, and depth.
- int h () const

  Returns the current image drawing height in FLTK units.
- void inactive ()

The inactive() method calls color_average(FL_BACKGROUND_COLOR, 0.33f) to produce an image that appears grayed out.

- int ld () const

  Returns the current line data size in bytes.
- virtual void release ()

  Releases an Fl_Image - the same as '`delete this`'.
- virtual void scale (int width, int height, int proportional=1, int can_expand=0)

  Sets the drawing size of the image.
- int w () const

  Returns the current image drawing width in FLTK units.
- virtual ~**Fl_Image** ()

  The destructor is a virtual method that frees all memory used by the image.

Friends

- class **Fl_ICO_Image**

Additional Inherited Members

Static Public Member Functions inherited from Fl_RGB_Image

- static size_t max_size ()

  Returns the maximum allowed image size in bytes when creating an Fl_RGB_Image object.
- static void max_size (size_t size)

  Sets the maximum allowed image size in bytes when creating an Fl_RGB_Image object.

Static Public Member Functions inherited from Fl_Image

- static Fl_Labeltype **define_FL_IMAGE_LABEL** ()
- static Fl_RGB_Scaling **RGB_scaling** ()

  Returns the currently used RGB image scaling method.
- static void RGB_scaling (Fl_RGB_Scaling)

  Sets the RGB image scaling method used for copy(int, int).
- static Fl_RGB_Scaling **scaling_algorithm** ()

  Gets what algorithm is used when resizing a source image to draw it.
- static void scaling_algorithm (Fl_RGB_Scaling algorithm)

  Sets what algorithm is used when resizing a source image to draw it.

Public Attributes inherited from Fl_RGB_Image

- int **alloc_array**

  If non-zero, the object's data array is delete[]'d when deleting the object.
- const uchar * array

  Points to the start of the object's data array.

Static Public Attributes inherited from Fl_Image

- static const int **ERR_FILE_ACCESS** = -2
- static const int **ERR_FORMAT** = -3
- static const int **ERR_MEMORY_ACCESS** = -4
- static const int **ERR_NO_IMAGE** = -1
- static bool **register_images_done** = false

  True after fl_register_images() was called, false before.

Protected Member Functions inherited from Fl_Image

- void **d** (int D)

  Sets the current image depth.
- void data (const char ∗const ∗p, int c)

  Sets the current data pointer and count of pointers in the array.
- void draw_empty (int X, int Y)

  The protected method draw_empty() draws a box with an X in it.
- int draw_scaled (int X, int Y, int W, int H)

  Draw the image to the current drawing surface rescaled to a given width and height.
- void h (int H)

  Sets the height of the image data.
- void ld (int LD)

  Sets the current line data size in bytes.
- void w (int W)

  Sets the width of the image data.

Static Protected Member Functions inherited from Fl_Image

- static void **labeltype** (const Fl_Label ∗lo, int lx, int ly, int lw, int lh, Fl_Align la)
- static void **measure** (const Fl_Label ∗lo, int &lw, int &lh)

13.97.1 Detailed Description

The Fl_PNG_Image class supports loading, caching, and drawing of Portable Network Graphics (PNG) image files. The class loads color-mapped and full-color images and handles color- and alpha-based transparency.

13.97.2 Constructor & Destructor Documentation

13.97.2.1 Fl_PNG_Image() [1/2]

```
Fl_PNG_Image::Fl_PNG_Image (
            const char * filename )
```

The constructor loads the named PNG image from the given png filename.
The destructor frees all memory and server resources that are used by the image.
Use Fl_Image::fail() to check if Fl_PNG_Image failed to load. fail() returns ERR_FILE_ACCESS if the file could not be opened or read, ERR_FORMAT if the PNG format could not be decoded, and ERR_NO_IMAGE if the image could not be loaded for another reason.

Parameters

in	*filename*	Name of PNG file to read

13.97.2.2 Fl_PNG_Image() [2/2]

```
Fl_PNG_Image::Fl_PNG_Image (
            const char * name_png,
            const unsigned char * buffer,
            int maxsize )
```

Constructor that reads a PNG image from memory.
Construct an image from a block of memory inside the application. Fluid offers "binary Data" chunks as a great way to add image data into the C++ source code. name_png can be NULL. If a name is given, the image is added to the list of shared images (see: Fl_Shared_Image) and will be available by that name.

Parameters

name_png	A name given to this image or NULL

Parameters

buffer	Pointer to the start of the PNG image in memory
maxsize	Size in bytes of the memory buffer containing the PNG image

The documentation for this class was generated from the following files:

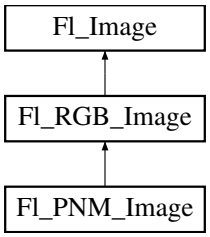
- Fl_PNG_Image.H
- Fl_PNG_Image.cxx

13.98 Fl_PNM_Image Class Reference

The Fl_PNM_Image class supports loading, caching, and drawing of Portable Anymap (PNM, PBM, PGM, PPM) image files.

`#include <Fl_PNM_Image.H>`

Inheritance diagram for Fl_PNM_Image:



Public Member Functions

- Fl_PNM_Image (const char *filename)

  The constructor loads the named PNM image.

Public Member Functions inherited from Fl_RGB_Image

- virtual Fl_SVG_Image * as_svg_image ()

  Returns whether an RGB image is an Fl_SVG_Image or not.
- virtual const Fl_SVG_Image * as_svg_image () const

  Returns whether an RGB image is an Fl_SVG_Image or not.
- int **cache_h** () const
- int **cache_w** () const
- void color_average (Fl_Color c, float i) override

  The color_average() method averages the colors in the image with the provided FLTK color value.
- Fl_Image * **copy** () const
- Fl_Image * copy (int W, int H) const override

  Creates a resized copy of the image.
- void desaturate () override

  The desaturate() method converts an image to grayscale.
- void **draw** (int X, int Y)
- void draw (int X, int Y, int W, int H, int cx=0, int cy=0) override

  Draws the image to the current drawing surface with a bounding box.
- Fl_RGB_Image (const Fl_Pixmap *pxm, Fl_Color bg=FL_GRAY)

  The constructor creates a new RGBA image from the specified Fl_Pixmap.
- Fl_RGB_Image (const uchar *bits, int bits_length, int W, int H, int D, int LD)

  The constructor creates a new image from the specified data.
- Fl_RGB_Image (const uchar *bits, int W, int H, int D=3, int LD=0)

The constructor creates a new image from the specified data.

- void label (Fl_Menu_Item *m) override

  This method is an obsolete way to set the image attribute of a menu item.
- void label (Fl_Widget *w) override

  This method is an obsolete way to set the image attribute of a widget or menu item.
- virtual void normalize ()

  Makes sure the object is fully initialized.
- void uncache () override

  If the image has been cached for display, delete the cache data.
- ~**Fl_RGB_Image** () override

  The destructor frees all memory and server resources that are used by the image.

Public Member Functions inherited from Fl_Image

- virtual class Fl_Shared_Image * as_shared_image ()

  Returns whether an image is an Fl_Shared_Image or not.
- Fl_Image * copy () const

  Creates a copy of the image in the same size.
- int count () const

  Returns the number of data values associated with the image.
- int d () const

  Returns the image depth.
- const char *const * data () const

  Returns a pointer to the current image data array.
- int **data_h** () const

  Returns the height of the image data.
- int **data_w** () const

  Returns the width of the image data.
- void draw (int X, int Y)

  Draws the image to the current drawing surface.
- int fail () const

  Returns a value that is not 0 if there is currently no image available.
- Fl_Image (int W, int H, int D)

  The constructor creates an empty image with the specified width, height, and depth.
- int h () const

  Returns the current image drawing height in FLTK units.
- void inactive ()

  The inactive() method calls color_average(FL_BACKGROUND_COLOR, 0.33f) to produce an image that appears grayed out.
- int ld () const

  Returns the current line data size in bytes.
- virtual void release ()

  Releases an Fl_Image - the same as '`delete this`'.
- virtual void scale (int width, int height, int proportional=1, int can_expand=0)

  Sets the drawing size of the image.
- int w () const

  Returns the current image drawing width in FLTK units.
- virtual ~**Fl_Image** ()

  The destructor is a virtual method that frees all memory used by the image.

Additional Inherited Members

Static Public Member Functions inherited from Fl_RGB_Image

- static size_t max_size ()

  Returns the maximum allowed image size in bytes when creating an Fl_RGB_Image object.
- static void max_size (size_t size)

  Sets the maximum allowed image size in bytes when creating an Fl_RGB_Image object.

Static Public Member Functions inherited from Fl_Image

- static Fl_Labeltype **define_FL_IMAGE_LABEL** ()
- static Fl_RGB_Scaling **RGB_scaling** ()

  Returns the currently used RGB image scaling method.
- static void RGB_scaling (Fl_RGB_Scaling)

  Sets the RGB image scaling method used for copy(int, int).
- static Fl_RGB_Scaling **scaling_algorithm** ()

  Gets what algorithm is used when resizing a source image to draw it.
- static void scaling_algorithm (Fl_RGB_Scaling algorithm)

  Sets what algorithm is used when resizing a source image to draw it.

Public Attributes inherited from Fl_RGB_Image

- int **alloc_array**

  If non-zero, the object's data array is delete[]'d when deleting the object.
- const uchar ∗ array

  Points to the start of the object's data array.

Static Public Attributes inherited from Fl_Image

- static const int **ERR_FILE_ACCESS** = -2
- static const int **ERR_FORMAT** = -3
- static const int **ERR_MEMORY_ACCESS** = -4
- static const int **ERR_NO_IMAGE** = -1
- static bool **register_images_done** = false

  True after fl_register_images() was called, false before.

Protected Member Functions inherited from Fl_Image

- void **d** (int D)

  Sets the current image depth.
- void data (const char ∗const ∗p, int c)

  Sets the current data pointer and count of pointers in the array.
- void draw_empty (int X, int Y)

  The protected method draw_empty() draws a box with an X in it.
- int draw_scaled (int X, int Y, int W, int H)

  Draw the image to the current drawing surface rescaled to a given width and height.
- void h (int H)

  Sets the height of the image data.
- void ld (int LD)

  Sets the current line data size in bytes.
- void w (int W)

  Sets the width of the image data.

Static Protected Member Functions inherited from Fl_Image

- static void **labeltype** (const Fl_Label *lo, int lx, int ly, int lw, int lh, Fl_Align la)
- static void **measure** (const Fl_Label *lo, int &lw, int &lh)

13.98.1 Detailed Description

The Fl_PNM_Image class supports loading, caching, and drawing of Portable Anymap (PNM, PBM, PGM, PPM) image files.
The class loads bitmap, grayscale, and full-color images in both ASCII and binary formats.

13.98.2 Constructor & Destructor Documentation

13.98.2.1 Fl_PNM_Image()

```
Fl_PNM_Image::Fl_PNM_Image (
            const char * filename )
```

The constructor loads the named PNM image.
The destructor frees all memory and server resources that are used by the image.
Use Fl_Image::fail() to check if Fl_PNM_Image failed to load. fail() returns ERR_FILE_ACCESS if the file could not be opened or read, ERR_FORMAT if the PNM format could not be decoded, and ERR_NO_IMAGE if the image could not be loaded for another reason.

Parameters

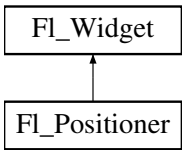
in	*filename*	a full path and name pointing to a valid jpeg file.

The documentation for this class was generated from the following files:

- Fl_PNM_Image.H
- Fl_PNM_Image.cxx

13.99 Fl_Positioner Class Reference

This class is provided for Forms compatibility.
`#include <Fl_Positioner.H>`
Inheritance diagram for Fl_Positioner:



Public Member Functions

- Fl_Positioner (int x, int y, int w, int h, const char *l=0)

  Creates a new Fl_Positioner widget using the given position, size, and label string.
- int handle (int) override

  Handles the specified event.
- int **value** (double, double)

  Returns the current position in x and y.
- void **xbounds** (double, double)

  Sets the X axis bounds.
- double **xmaximum** () const

  Gets the X axis maximum.

- void **xmaximum** (double a)

  Same as xbounds(xminimum(), a)
- double **xminimum** () const

  Gets the X axis minimum.
- void **xminimum** (double a)

  Same as xbounds(a, xmaximum())
- void **xstep** (double a)

  Sets the stepping value for the X axis.
- double **xvalue** () const

  Gets the X axis coordinate.
- int **xvalue** (double)

  Sets the X axis coordinate.
- void **ybounds** (double, double)

  Sets the Y axis bounds.
- double **ymaximum** () const

  Gets the Y axis maximum.
- void **ymaximum** (double a)

  Same as ybounds(ymininimum(), a)
- double **yminimum** () const

  Gets the Y axis minimum.
- void **yminimum** (double a)

  Same as ybounds(a, ymaximum())
- void **ystep** (double a)

  Sets the stepping value for the Y axis.
- double **yvalue** () const

  Gets the Y axis coordinate.
- int **yvalue** (double)

  Sets the Y axis coordinate.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

Returns an Fl_Group pointer if this widget is an Fl_Group.

- virtual Fl_Group const ∗ **as_group** () const
- virtual Fl_Window ∗ as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ **as_window** () const
- void bind_deimage (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗cb, void ∗p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget ∗w) const

  Checks if w is a child of this widget.

- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

Checks if this widget is a child of `wgt`.

- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().

- const char * label () const

  Gets the current label text.

- void label (const char *text)

  Sets the current label pointer.

- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.

- int label_image_spacing ()

  Return the gap size between the label and the image.

- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.

- Fl_Color labelcolor () const

  Gets the label color.

- void labelcolor (Fl_Color c)

  Sets the label color.

- Fl_Font labelfont () const

  Gets the font to use.

- void labelfont (Fl_Font f)

  Sets the font to use.

- Fl_Fontsize labelsize () const

  Gets the font size in pixels.

- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.

- Fl_Labeltype labeltype () const

  Gets the label type.

- void labeltype (Fl_Labeltype a)

  Sets the label type.

- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.

- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.

- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.

- unsigned int output () const

  Returns if a widget is used for output only.

- Fl_Group * parent () const

  Returns a pointer to the parent widget.

- void parent (Fl_Group *p)

  Internal use only - "for hacks only".

- void position (int X, int Y)

  Repositions the window or widget.

- void redraw ()

  Schedules the drawing of the widget.

- void redraw_label ()

  Schedules the drawing of the label.

- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.

- Fl_Color selection_color () const

  Gets the selection color.

- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

Set the spacing between the label and the vertical edge of the widget.

- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void draw () override

  Draws the widget.
- void **draw** (int, int, int, int)
- int **handle** (int, int, int, int, int)

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

Draws the label in an arbitrary bounding box.

- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.99.1 Detailed Description

This class is provided for Forms compatibility.
It provides 2D input. It would be useful if this could be put atop another widget so that the crosshairs are on top, but this is not implemented. The color of the crosshairs is selection_color().

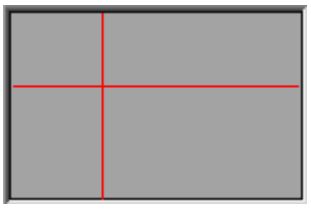


Figure 13.35 Fl_Positioner

13.99.2 Constructor & Destructor Documentation

13.99.2.1 Fl_Positioner()

```
Fl_Positioner::Fl_Positioner (
            int X,
            int Y,
            int W,
            int H,
            const char * l = 0 )
```

Creates a new Fl_Positioner widget using the given position, size, and label string.
The default boxtype is FL_NO_BOX.

13.99.3 Member Function Documentation

13.99.3.1 draw()

```
void Fl_Positioner::draw ( )  [override], [protected], [virtual]
```

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();                 // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

13.99.3.2 handle()

```
int Fl_Positioner::handle (
            int event )  [override], [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to *override* the behavior of the base class. This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to *filter* event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you *consumed* the event by returning 1 even if you didn't *do* anything with it.

Parameters

in	*event*	the kind of event received

Return values

0	if the event was not used or understood
1	if the event was used and can be deleted

See also

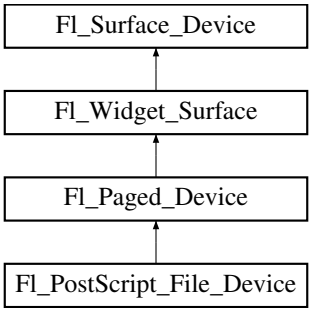
Fl_Event

Reimplemented from Fl_Widget.
The documentation for this class was generated from the following files:

- Fl_Positioner.H
- Fl_Positioner.cxx

13.100 Fl_PostScript_File_Device Class Reference

To send graphical output to a PostScript file.
`#include <Fl_PostScript.H>`
Inheritance diagram for Fl_PostScript_File_Device:



Public Member Functions

- int begin_job (FILE ∗ps_output, int pagecount=0, enum Fl_Paged_Device::Page_Format format=Fl_Paged_Device::A4, enum Fl_Paged_Device::Page_Layout layout=Fl_Paged_Device::PORTRAIT)

  Begins the session where all graphics requests will go to FILE pointer.
- int begin_job (int pagecount, int ∗from, int ∗to, char ∗∗perr_message) override

  Don't use with this class.
- int begin_job (int pagecount=0, enum Fl_Paged_Device::Page_Format format=Fl_Paged_Device::A4, enum Fl_Paged_Device::Page_Layout layout=Fl_Paged_Device::PORTRAIT)

  Begins the session where all graphics requests will go to a local PostScript file.
- int begin_page (void) override

  Begins a new printed page.
- void **close_command** (Fl_PostScript_Close_Command cmd)

  Sets the function end_job() calls to close the file()
- void end_current () override

  FLTK calls this each time a surface ceases to be the current drawing surface.
- void end_job (void) override

  Completes all PostScript output.
- int end_page (void) override

  To be called at the end of each page.
- FILE ∗ **file** ()

  Returns the underlying FILE∗ receiving all PostScript data.
- **Fl_PostScript_File_Device** ()

The constructor.

- void margins (int ∗left, int ∗top, int ∗right, int ∗bottom) override

  Computes the dimensions of margins that lie between the printable page area and the full page.
- void origin (int ∗x, int ∗y) override

  Computes the coordinates of the current origin of graphics functions.
- void origin (int x, int y) override

  Sets the position of the origin of graphics in the drawable part of the drawing surface.
- int printable_rect (int ∗w, int ∗h) override

  Computes the width and height of the drawable area of the drawing surface.
- void rotate (float angle) override

  Rotates the graphics operations relatively to paper.
- void scale (float scale_x, float scale_y=0.) override

  Changes the scaling of page coordinates.
- void set_current () override

  Make this surface the current drawing surface.
- int start_job (FILE ∗ps_output, int pagecount=0, enum Fl_Paged_Device::Page_Format format=Fl_Paged_Device::A4, enum Fl_Paged_Device::Page_Layout layout=Fl_Paged_Device::PORTRAIT)

  Synonym of begin_job().
- int start_job (int pagecount=0, enum Fl_Paged_Device::Page_Format format=Fl_Paged_Device::A4, enum Fl_Paged_Device::Page_Layout layout=Fl_Paged_Device::PORTRAIT)

  Synonym of begin_job().
- void translate (int x, int y) override

  Translates the current graphics origin accounting for the current rotation.
- void untranslate (void) override

  Undoes the effect of a previous translate() call.
- ∼**Fl_PostScript_File_Device** ()

  The destructor.

Public Member Functions inherited from Fl_Paged_Device

- void **print_widget** (Fl_Widget ∗widget, int delta_x=0, int delta_y=0)

  Synonym of draw(Fl_Widget∗, int, int)
- void **print_window** (Fl_Window ∗win, int x_off=0, int y_off=0)

  Synonym of draw_decorated_window(Fl_Window∗, int, int)
- int start_job (int pagecount=0, int ∗frompage=NULL, int ∗topage=NULL, char ∗∗perr_message=NULL)

  Synonym of begin_job(int pagecount, int ∗frompage, int ∗topage, char ∗∗perr_message).
- int start_page ()

  Synonym of begin_page().
- virtual ∼**Fl_Paged_Device** ()

  The destructor.

Public Member Functions inherited from Fl_Widget_Surface

- void draw (Fl_Widget ∗widget, int delta_x=0, int delta_y=0)

  Draws the widget on the drawing surface.
- void draw_decorated_window (Fl_Window ∗win, int x_offset=0, int y_offset=0)

  Draws a window with its title bar and frame if any.
- void print_window_part (Fl_Window ∗win, int x, int y, int w, int h, int delta_x=0, int delta_y=0)

  Draws a rectangular part of an on-screen window.

Public Member Functions inherited from Fl_Surface_Device

- Fl_Graphics_Driver * **driver** ()

  Returns the graphics driver of this drawing surface.
- virtual bool is_current ()

  Is this surface the current drawing surface?
- virtual ~**Fl_Surface_Device** ()

  The destructor.

Static Public Attributes

- static const char * **file_chooser_title**

  Label of the PostScript file chooser window.

Static Public Attributes inherited from Fl_Paged_Device

- static const page_format **page_formats** [NO_PAGE_FORMATS]

  width, height and name of all elements of the enum Page_Format.

Protected Member Functions

- Fl_PostScript_Graphics_Driver * **driver** ()

  Returns the PostScript driver of this drawing surface.

Protected Member Functions inherited from Fl_Paged_Device

- **Fl_Paged_Device** ()

  The constructor.

Protected Member Functions inherited from Fl_Widget_Surface

- Fl_Widget_Surface (Fl_Graphics_Driver *d)

  The constructor.

Protected Member Functions inherited from Fl_Surface_Device

- void **driver** (Fl_Graphics_Driver *graphics_driver)

  Sets the graphics driver of this drawing surface.
- **Fl_Surface_Device** (Fl_Graphics_Driver *graphics_driver)

  Constructor that sets the graphics driver to use for the created surface.

Additional Inherited Members

Public Types inherited from Fl_Paged_Device

- enum Page_Format {
  A0 = 0 , A1 , A2 , A3 ,
  A4 , A5 , A6 , A7 ,
  A8 , A9 , B0 , B1 ,
  B2 , B3 , B4 , B5 ,
  B6 , B7 , B8 , B9 ,
  B10 , **C5E** , **DLE** , EXECUTIVE ,
  FOLIO , LEDGER , LEGAL , LETTER ,
  TABLOID , **ENVELOPE** , **MEDIA** = 0x1000 }

  Possible page formats.
- enum Page_Layout { PORTRAIT = 0 , LANDSCAPE = 0x100 , REVERSED = 0x200 , ORIENTATION = 0x300 }

  Possible page layouts.

Static Public Member Functions inherited from Fl_Surface_Device

- static Fl_Surface_Device * pop_current ()

  Removes the top element from the current drawing surface stack, and makes the new top element current.
- static void push_current (Fl_Surface_Device *new_current)

  Pushes `new_current` on top of the stack of current drawing surfaces, and makes it current.
- static Fl_Surface_Device * surface ()

  The current drawing surface.

Protected Attributes inherited from Fl_Widget_Surface

- int **x_offset**

  horizontal offset to the origin of graphics coordinates
- int **y_offset**

  vertical offset to the origin of graphics coordinates

13.100.1 Detailed Description

To send graphical output to a PostScript file.
This class is used exactly as the Fl_Printer class except for the begin_job() call, two variants of which are usable and allow to specify what page format and layout are desired.

Processing of text: Text uses vectorial fonts under the X11 + Pango and the Wayland platforms. With other platforms, only text restricted to the Latin alphabet (and a few other characters listed in the table below) and to FLTK standard fonts is vectorized. All other unicode characters or all other fonts (FL_FREE_FONT and above) are output as a bitmap. FLTK standard fonts are output using the corresponding PostScript standard fonts. The latin alphabet means all unicode characters between U+0020 and U+017F, or, in other words, the ASCII, Latin-1 Supplement and Latin Extended-A charts.

Char	Codepoint	Name	Char	Codepoint	Name	Char	Codepoint	Name
ƒ	U+0192	florin	‚	U+201A	quotesinglbase	™	U+2122	trademark
ˆ	U+02C6	circumflex	“	U+201C	quotedblleft	∂	U+2202	partialdiff
ˇ	U+02C7	caron	”	U+201D	quotedblright	Δ	U+2206	Delta
˘	U+02D8	breve	„	U+201E	quotedblbase	∑	U+2211	summation
˙	U+02D9	dotaccent	†	U+2020	dagger	√	U+221A	radical
˚	U+02DA	ring	‡	U+2021	daggerdbl	∞	U+221E	infinity
˛	U+02DB	ogonek	•	U+2022	bullet	≠	U+2260	notequal
˜	U+02DC	tilde	…	U+2026	ellipsis	≤	U+2264	lessequal
˝	U+02DD	hungarumlaut	‰	U+2030	perthousand	≥	U+2265	greaterequal
–	U+2013	endash	‹	U+2039	guilsinglleft	◊	U+25CA	lozenge
—	U+2014	emdash	›	U+203A	guilsinglright	fi	U+FB01	fi
‘	U+2018	quoteleft	/	U+2044	fraction	fl	U+FB02	fl
’	U+2019	quoteright	€	U+20AC	Euro	﴿	U+F8FF	apple (macOS only)

Figure 13.36 Extra characters supported by standard PostScript fonts

Processing of transparent Fl_RGB_Image objects: Under the X11 + Pango and the Wayland platforms, these objects are output with their exact transparency. With other platforms, these objects are drawn blended to white color. Class Fl_EPS_File_Surface 's constructor allows to set another background color for blending.

13.100.2 Member Function Documentation

13.100.2.1 begin_job() [1/3]

```
int Fl_PostScript_File_Device::begin_job (
            FILE * ps_output,
            int pagecount = 0,
            enum Fl_Paged_Device::Page_Format format = Fl_Paged_Device::A4,
            enum Fl_Paged_Device::Page_Layout layout = Fl_Paged_Device::PORTRAIT )
```

Begins the session where all graphics requests will go to FILE pointer.
This member function prevents end_job() from closing `ps_output`, so the user can check with `ferror(ps_output)` for output errors.

Parameters

ps_output	A writable FILE pointer that will receive PostScript output and that should not be closed until after end_job() has been called.
pagecount	The total number of pages to be created. Use 0 if this number is unknown when this function is called.
format	Desired page format.
layout	Desired page layout.

Returns

always 0.

13.100.2.2 begin_job() [2/3]

```
int Fl_PostScript_File_Device::begin_job (
            int pagecount,
            int * from,
            int * to,
            char ** perr_message )  [override], [virtual]
```

Don't use with this class.
Reimplemented from Fl_Paged_Device.

13.100.2.3 begin_job() [3/3]

```
int Fl_PostScript_File_Device::begin_job (
            int pagecount = 0,
            enum Fl_Paged_Device::Page_Format format = Fl_Paged_Device::A4,
            enum Fl_Paged_Device::Page_Layout layout = Fl_Paged_Device::PORTRAIT )
```

Begins the session where all graphics requests will go to a local PostScript file.
Opens a file dialog to select an output PostScript file. This member function makes end_job() close the resulting PostScript file and display an alert message with fl_alert() in case of any output error.

Parameters

pagecount	The total number of pages to be created. Use 0 if this number is unknown when this function is called.
format	Desired page format.
layout	Desired page layout.

Returns

0 if OK, 1 if user cancelled the file dialog, 2 if fopen failed on user-selected output file.

13.100.2.4 begin_page()

```
int Fl_PostScript_File_Device::begin_page (
            void ) [override], [virtual]
```

Begins a new printed page.
The page coordinates are initially in points, i.e., 1/72 inch, and with origin at the top left of the printable page area.
This function also makes this surface the current drawing surface with Fl_Surface_Device::push_current().

Note

begin_page() calls Fl_Surface_Device::push_current() and leaves this device as the active surface. If any calls between begin_page() and end_page() open dialog boxes or will otherwise draw into FLTK windows, those calls must be put between a call to Fl_Surface_Device::pop_current() and a call to Fl_Surface_Device::push_current(), or the content of the dialog box will be rendered to the printer instead of the screen.

Returns

0 if OK, non-zero if any error

Reimplemented from Fl_Paged_Device.

13.100.2.5 end_current()

```
void Fl_PostScript_File_Device::end_current ( ) [override], [virtual]
```

FLTK calls this each time a surface ceases to be the current drawing surface.
This member function is mostly of interest to developers of new Fl_Surface_Device derived classes. It allows to perform surface-specific operations necessary when this surface ceases to be current. Each implementation should end with a call to Fl_Surface_Device::end_current().
Reimplemented from Fl_Surface_Device.

13.100.2.6 end_job()

```
void Fl_PostScript_File_Device::end_job (
            void ) [override], [virtual]
```

Completes all PostScript output.
This also closes with `fclose()` the underlying file() unless close_command() was used to set another function.
Reimplemented from Fl_Paged_Device.

13.100.2.7 end_page()

```
int Fl_PostScript_File_Device::end_page (
            void ) [override], [virtual]
```

To be called at the end of each page.
This function also stops this surface from being the current drawing surface with Fl_Surface_Device::pop_current().

Note

end_page() calls Fl_Surface_Device::pop_current(). If any calls between begin_page() and end_page() open dialog boxes or will otherwise draw into FLTK windows, those calls must be put between a call to Fl_Surface_Device::pop_current() and a call to Fl_Surface_Device::push_current().

Returns

0 if OK, non-zero if any error.

Reimplemented from Fl_Paged_Device.

13.100.2.8 margins()

```
void Fl_PostScript_File_Device::margins (
            int * left,
            int * top,
            int * right,
            int * bottom )  [override], [virtual]
```

Computes the dimensions of margins that lie between the printable page area and the full page.
Values are in the same unit as that used by FLTK drawing functions. They are changed by scale() calls.

Parameters

out	*left*	If non-null, *left is set to the left margin size.
out	*top*	If non-null, *top is set to the top margin size.
out	*right*	If non-null, *right is set to the right margin size.
out	*bottom*	If non-null, *bottom is set to the bottom margin size.

Reimplemented from Fl_Paged_Device.

13.100.2.9 origin() [1/2]

```
void Fl_PostScript_File_Device::origin (
            int * x,
            int * y )  [override], [virtual]
```

Computes the coordinates of the current origin of graphics functions.

Parameters

out	*x,y*	If non-null, *x and *y are set to the horizontal and vertical coordinates of the graphics origin.

Reimplemented from Fl_Widget_Surface.

13.100.2.10 origin() [2/2]

```
void Fl_PostScript_File_Device::origin (
            int x,
            int y )  [override], [virtual]
```

Sets the position of the origin of graphics in the drawable part of the drawing surface.
Arguments should be expressed relatively to the result of a previous printable_rect() call. That is, `printable_rect(&w, &h); origin(w/2, 0);` sets the graphics origin at the top center of the drawable area. Successive origin() calls don't combine their effects. Origin() calls are not affected by rotate() calls (for classes derived from Fl_Paged_Device).

Parameters

in	*x,y*	Horizontal and vertical positions in the drawing surface of the desired origin of graphics.

Reimplemented from Fl_Widget_Surface.

13.100.2.11 printable_rect()

```
int Fl_PostScript_File_Device::printable_rect (
            int * w,
            int * h )  [override], [virtual]
```

Computes the width and height of the drawable area of the drawing surface.
Values are in the same unit as that used by FLTK drawing functions and are unchanged by calls to origin(). If the object is derived from class Fl_Paged_Device, values account for the user-selected paper type and print orientation and are changed by scale() calls.

Returns

0 if OK, non-zero if any error

Reimplemented from Fl_Widget_Surface.

13.100.2.12 rotate()

```
void Fl_PostScript_File_Device::rotate (
            float angle )  [override], [virtual]
```

Rotates the graphics operations relatively to paper.
The rotation is centered on the current graphics origin. Successive rotate() calls don't combine their effects.

Parameters

angle	Rotation angle in counter-clockwise degrees.

Reimplemented from Fl_Paged_Device.

13.100.2.13 scale()

```
void Fl_PostScript_File_Device::scale (
            float scale_x,
            float scale_y = 0.  )  [override], [virtual]
```

Changes the scaling of page coordinates.
This function also resets the origin of graphics functions at top left of printable page area. After a scale() call, do a printable_rect() call to get the new dimensions of the printable page area. Successive scale() calls don't combine their effects.

Parameters

scale↩ _x	Horizontal dimensions of plot are multiplied by this quantity.
scale↩ _y	Same as above, vertically. The value 0. is equivalent to setting `scale_y` = `scale_x`. Thus, scale(factor); is equivalent to scale(factor, factor);

Reimplemented from Fl_Paged_Device.

13.100.2.14 set_current()

```
void Fl_PostScript_File_Device::set_current (
            void  )  [override], [virtual]
```

Make this surface the current drawing surface.
This surface will receive all future graphics requests.
Since FLTK 1.4.0 the preferred API to change the current drawing surface is Fl_Surface_Device::push_current( ) / Fl_Surface_Device::pop_current().

Note

It is recommended to use this function only as follows :

- The current drawing surface is the display;
- make current another surface, e.g., an Fl_Printer or an Fl_Image_Surface object, calling set_current() on this object;
- draw to that surface;
- make the display current again with Fl_Display_Device::display_device()->set_current();
  don't do any other call to set_current() before this one.

Other scenarios of drawing surface changes should be performed via Fl_Surface_Device::push_current() and Fl_Surface_Device::pop_current().
Reimplemented from Fl_Surface_Device.

13.100.2.15 start_job() [1/2]

```
int Fl_PostScript_File_Device::start_job (
            FILE * ps_output,
            int pagecount = 0,
            enum Fl_Paged_Device::Page_Format format = Fl_Paged_Device::A4,
            enum Fl_Paged_Device::Page_Layout layout = Fl_Paged_Device::PORTRAIT )  [inline]
```

Synonym of begin_job().
For API compatibility with FLTK 1.3.x

13.100.2.16 start_job() [2/2]

```
int Fl_PostScript_File_Device::start_job (
            int pagecount = 0,
            enum Fl_Paged_Device::Page_Format format = Fl_Paged_Device::A4,
            enum Fl_Paged_Device::Page_Layout layout = Fl_Paged_Device::PORTRAIT )  [inline]
```

Synonym of begin_job().
For API compatibility with FLTK 1.3.x

13.100.2.17 translate()

```
void Fl_PostScript_File_Device::translate (
            int x,
            int y )  [override], [virtual]
```

Translates the current graphics origin accounting for the current rotation.
Each translate() call must be matched by an untranslate() call. Successive translate() calls add up their effects.
Reimplemented from Fl_Widget_Surface.

13.100.2.18 untranslate()

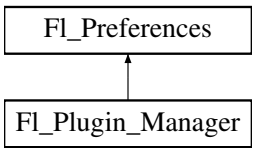
```
void Fl_PostScript_File_Device::untranslate (
            void )  [override], [virtual]
```

Undoes the effect of a previous translate() call.
Reimplemented from Fl_Widget_Surface.
The documentation for this class was generated from the following file:

- Fl_PostScript.H

13.101 Fl_Preferences Class Reference

Fl_Preferences store user settings between application starts.
`#include <Fl_Preferences.H>`
Inheritance diagram for Fl_Preferences:



Classes

- struct Entry
- class Name

  'Name' provides a simple method to create numerical or more complex procedural names for entries and groups on the fly.
- class Node
- class RootNode

Public Types

- typedef void * ID

  Every Fl_Preferences-Group has a unique ID.
- enum Root {
  UNKNOWN_ROOT_TYPE = -1 , SYSTEM = 0 , USER , MEMORY ,
  ROOT_MASK = 0x00FF , CORE = 0x0100 , C_LOCALE = 0x1000 , CLEAR = 0x2000 ,
  SYSTEM_L = SYSTEM | C_LOCALE , USER_L = USER | C_LOCALE , CORE_SYSTEM_L = CORE | SYSTEM_L , CORE_USER_L = CORE | USER_L ,
  CORE_SYSTEM = CORE | SYSTEM , CORE_USER = CORE | USER }

  Define the scope of the preferences.

Public Member Functions

- char **clear** ()

  Delete all groups and all entries.
- char **delete_all_entries** ()

  Delete all entries.
- char **delete_all_groups** ()

  Delete all groups.
- char delete_entry (const char *entry)

  Deletes a single name/value pair.
- char delete_group (const char *group)

  Deletes a group.
- int dirty ()

  Check if there were changes to the database that need to be written to disk.
- int entries ()

  Returns the number of entries (name/value pairs) in a group.
- const char * entry (int index)

  Returns the name of an entry.
- char entry_exists (const char *key)

  Returns non-zero if an entry with this name exists.
- Root filename (char *buffer, size_t buffer_size)

  Return the file name and path to the preference file.
- Fl_Preferences (const char *path, const char *vendor, const char *application)

  Deprecated: Use this constructor to create or read a preference file at an arbitrary position in the file system.
- Fl_Preferences (const char *path, const char *vendor, const char *application, Root flags)

  Use this constructor to create or read a preference file at an arbitrary position in the file system.
- **Fl_Preferences** (const Fl_Preferences &)

  Create another reference to a Preferences group.
- Fl_Preferences (Fl_Preferences &parent, const char *group)

  Generate or read a new group of entries within another group.
- Fl_Preferences (Fl_Preferences &parent, int groupIndex)

  Open a child group using a given index.
- Fl_Preferences (Fl_Preferences *parent, const char *group)

  Create or access a group of preferences using a name.
- Fl_Preferences (Fl_Preferences *parent, int groupIndex)
- Fl_Preferences (ID id)

  Create a new dataset access point using a dataset ID.
- Fl_Preferences (Root root, const char *vendor, const char *application)

  The constructor creates a group that manages key/value pairs and child groups.
- int flush ()

Writes preferences to disk if they were modified.

- char get (const char *entry, char *&value, const char *defaultValue)

  Reads an entry from the group.
- char get (const char *entry, char *value, const char *defaultValue, int maxSize)

  Reads an entry from the group.
- char get (const char *entry, double &value, double defaultValue)

  Reads an entry from the group.
- char get (const char *entry, float &value, float defaultValue)

  Reads an entry from the group.
- char get (const char *entry, int &value, int defaultValue)

  Reads an entry from the group.
- char get (const char *entry, std::string &value, const std::string &defaultValue)

  Reads an entry from the group.
- char get (const char *entry, void *&value, const void *defaultValue, int defaultSize)

  Reads an entry from the group.
- char get (const char *entry, void *value, const void *defaultValue, int defaultSize, int *size)

  Reads a binary entry from the group, encoded in hexadecimal blocks.
- char get (const char *entry, void *value, const void *defaultValue, int defaultSize, int maxSize)

  Reads a binary entry from the group, encoded in hexadecimal blocks.
- char get_userdata_path (char *path, int pathlen)

  Creates a path that is related to the preference file and that is usable for additional application data.
- char **get_userdata_path** (std::string &path)
- const char * group (int num_group)

  Returns the name of the Nth (`num_group`) group.
- char group_exists (const char *key)

  Returns non-zero if a group with this name exists.
- int groups ()

  Returns the number of groups that are contained within a group.
- ID **id** () const

  Return an ID that can later be reused to open more references to this dataset.
- const char * **name** () const

  Return the name of this entry.
- const char * **path** () const

  Return the full path to this entry.
- char set (const char *entry, const char *value)

  Sets an entry (name/value pair).
- char set (const char *entry, const std::string &value)

  Sets an entry (name/value pair).
- char set (const char *entry, const void *value, int size)

  Sets an entry (name/value pair).
- char set (const char *entry, double value)

  Sets an entry (name/value pair).
- char set (const char *entry, double value, int precision)

  Sets an entry (name/value pair).
- char set (const char *entry, float value)

  Sets an entry (name/value pair).
- char set (const char *entry, float value, int precision)

  Sets an entry (name/value pair).
- char set (const char *entry, int value)

  Sets an entry (name/value pair).
- int size (const char *entry)

  Returns the size of the value part of an entry.
- virtual ~Fl_Preferences ()

  The destructor removes allocated resources.

Static Public Member Functions

- static unsigned int file_access ()

  Return the current file access permissions for the FLTK preferences system.
- static void file_access (unsigned int flags)

  Tell the FLTK preferences system which files in the file system it may read, create, or write.
- static Root filename (char ∗buffer, size_t buffer_size, Root root, const char ∗vendor, const char ∗application)

  Determine the file name and path to preferences that would be opened with these parameters.
- static const char ∗ new_UUID ()

  Returns a UUID as generated by the system.
- static char **remove** (ID id_)

  Remove the group with this ID from a database.

Static Public Attributes

- static const unsigned int **ALL** = ALL_READ_OK | ALL_WRITE_OK

  Set this to give FLTK and applications permission to read, write, and create preference files.
- static const unsigned int **ALL_READ_OK** = USER_READ_OK | SYSTEM_READ_OK | CORE_READ_OK

  Set this to allow FLTK and applications to read preference files.
- static const unsigned int **ALL_WRITE_OK** = USER_WRITE_OK | SYSTEM_WRITE_OK | CORE_WRITE_OK

  Set this to allow FLTK and applications to create and write preference files.
- static const unsigned int **APP_OK** = SYSTEM_OK | USER_OK

  Set this if it is OK for applications to read, create, and write any kind of preference files.
- static const unsigned int **CORE_OK** = CORE_READ_OK | CORE_WRITE_OK

  Set this if it is OK for FLTK to read, create, or write preference files.
- static const unsigned int CORE_READ_OK = 0x0010

  Set this if it is OK for FLTK to read preference files.
- static const unsigned int CORE_WRITE_OK = 0x0020

  Set this if it is OK for FLTK to create or write preference files.
- static const unsigned int NONE = 0x0000

  Set this if no call to Fl_Preferences shall access the file system.
- static const unsigned int **SYSTEM_OK** = SYSTEM_READ_OK | SYSTEM_WRITE_OK

  Set this if it is OK for applications to read, create, and write system wide preference files.
- static const unsigned int **SYSTEM_READ_OK** = 0x0004

  Set this if it is OK for applications to read system wide preference files.
- static const unsigned int **SYSTEM_WRITE_OK** = 0x0008

  Set this if it is OK for applications to create and write system wide preference files.
- static const unsigned int **USER_OK** = USER_READ_OK | USER_WRITE_OK

  Set this if it is OK for applications to read, create, and write user preference files.
- static const unsigned int **USER_READ_OK** = 0x0001

  Set this if it is OK for applications to read user preference files.
- static const unsigned int **USER_WRITE_OK** = 0x0002

  Set this if it is OK for applications to create and write user preference files.

Protected Attributes

- Node ∗ **node**
- RootNode ∗ **rootNode**

Friends

- class **Node**
- class **RootNode**

13.101.1 Detailed Description

Fl_Preferences store user settings between application starts.
FLTK Preferences are similar to the Registry on Windows and Preferences on MacOS, providing a simple method to store customizable user settings between application launches. A typical use is storing the last window position or a history of previously used documents.
Preferences are organized in a hierarchy of groups. Every group can contain more groups and any number of key/value pairs. Keys can be text strings containing ASCII letters, digits, periods, and underscores. Forward slashes in a key name are treated as subgroups, i.e. the key 'window/width' would actually refer to the key 'width' inside the group 'window'.
Keys have a unique name within their group. A value can be any string including control characters 0x00 to 0x1f, 0x7f, and UTF-8 octets.
Several methods allow setting and getting numerical values and binary data.
Preferences files are the same across platforms. User comments in preference files are preserved. Filenames are unique for each application by using a vendor/application naming scheme. The developer app must provide default values for all entries to ensure proper operation should preferences be corrupted or not yet exist.

Note

The format of preferences files is not part of the FLTK specification and intentionally undocumented. The only valid way to read or write prefs files is via the API from your app. The fact that the current implementation looks like human-readable text is purely coincidental and may change at any time. Preferences files are not meant to be created or edited "by hand."

FLTK preferences are not meant to replace a fully featured database. No merging of data takes place. If several instances of an app access the same database at the same time, only the most recent changes will persist.
Preferences should not be used to store document data. The .prefs file should be kept small for performance reasons. One application can have multiple preferences files. Extensive binary data however should be stored in separate files: see *Fl_Preferences::get_userdata_path()* .
Fl_Preferences are not thread-safe. They can temporarily change the locale on some platforms during read and write access, which also changes it temporarily in other threads of the same app.
Typically a preferences database is read at startup, and then reopened and written at app shutdown:

```
int appWindowWidth, appWindowHeight;

void launch() {
  Fl_Preferences app(Fl_Preferences::USER_L, "matthiasm.com", "hello");
  // 'app' constructor will be called, reading data from .prefs file
  Fl_Preferences window(app, "window");
  window.get("width", appWindowWidth, 800);
  window.get("height", appWindowHeight, 600);
  // 'app' destructor will be called. This will write data to the
  // .prefs file if any preferences were changed or added
}

void quit() {
  Fl_Preferences app(Fl_Preferences::USER_L, "matthiasm.com", "hello");
  Fl_Preferences window(app, "window");
  window.set("width", appWindowWidth);
  window.set("height", appWindowHeight);
}
```

See also

Fl_Preferences::Fl_Preferences(Root root, const char *vendor, const char *application)

As a special case, Fl_Preferences can be memory mapped and not be associated with a file on disk. See Fl_Preferences::Fl_Preferences(Fl_Preferences *parent, const char *group) and Fl_Preferences::MEMORY for more details on memory mapped preferences.

Note

Starting with FLTK 1.3, preferences databases are expected to be in UTF-8 encoding. Previous databases were stored in the current character set or code page which renders them incompatible for text entries using international characters.

Starting with FLTK 1.4, searching a valid path to store the preference files has changed slightly. Please see Fl_Preferences::Fl_Preferences(Root, const char*, const char*) for details.

Starting with FLTK 1.4, preference files should be created with `SYSTEM_L` or `USER_L` to be interchangeable between computers with differing locale settings. The legacy modes, `LOCAL` and `SYSTEM`, will read and

write floating point values using the decimal point of the current locale. As a result, a fp-value would be written '3,1415' on a German machine, and would be read back as '3.0' on a US machine because the comma would not be recognized as an alternative decimal point.

13.101.2 Member Typedef Documentation

13.101.2.1 ID

```
typedef void* Fl_Preferences::ID
```

Every Fl_Preferences-Group has a unique ID.
ID's can be retrieved from an Fl_Preferences-Group and can then be used to create more Fl_Preference references to the same data set, as long as the database remains open.

13.101.3 Member Enumeration Documentation

13.101.3.1 Root

```
enum Fl_Preferences::Root
```

Define the scope of the preferences.

Enumerator

UNKNOWN_ROOT_TYPE	Returned if storage could not be determined.
SYSTEM	Preferences are used system-wide. Deprecated, see SYSTEM_L.
USER	Preferences apply only to the current user. Deprecated, see USER_L.
MEMORY	Returned if querying memory mapped preferences.
ROOT_MASK	Mask for the values above.
CORE	OR'd by FLTK to read and write core library preferences and options.
C_LOCALE	This flag should always be set to ensure that floating point values are written and read correctly independently of the current locale.
CLEAR	Don't read a possibly existing database. Instead, start with an empty set of preferences.
SYSTEM_L	Preferences are used system-wide, locale independent.
USER_L	Preferences apply only to the current user, locale independent.
CORE_SYSTEM_L	Same as CORE | SYSTEM | C_LOCALE.
CORE_USER_L	Same as CORE | USER | C_LOCALE.
CORE_SYSTEM	Deprecated, same as CORE | SYSTEM. Use CORE_SYSTEM_L instead.
CORE_USER	Deprecated, same as CORE | USER. Use CORE_USER_L instead.

13.101.4 Constructor & Destructor Documentation

13.101.4.1 Fl_Preferences() [1/8]

```
Fl_Preferences::Fl_Preferences (
            Root root,
            const char * vendor,
            const char * application )
```

The constructor creates a group that manages key/value pairs and child groups.
Preferences can be stored per user using the root type `Fl_Preferences::USER_L`, or stored system-wide using `Fl_Preferences::SYSTEM_L`.
Groups and key/value pairs can be read and written randomly. Reading undefined values will return the default value. Writing undefined values will create all required groups and key/vlaue pairs.
This constructor creates the *base* instance for all following entries and reads the database from disk into memory if it exists. The vendor argument is a unique text string identifying the development team or vendor of an application. A domain name or an EMail address (replacing the '@' with a '.') are great unique names, e.g. "research.matthiasm.↩

com" or "fluid.fltk.org". The application argument can be the working title or final name of your application. Both vendor and application must be valid UNIX path segments as they become parts of the preference file path and may contain forward slashes to create deeper file structures.

Note

On **Windows**, the directory is constructed by querying the *Common AppData* or *AppData* key of the `Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders` registry entry. The filename and path is then constructed as `$(query)/$(vendor)/$(application).prefs` . If the query call fails, data will be stored in RAM only. It will be lost when the app exits.

In FLTK versions before 1.4.0, if querying the registry failed,

preferences would be written to `C:\FLTK\$(vendor)\$(application).prefs` .

Note

On **Linux**, the `USER` directory is constructed by reading `$HOME` . If `$HOME` is not set or not pointing to an existing directory, FLTK will check the path member of the passwd struct returned by `getpwuid(getuid())` . If all attempts fail, data will be stored in RAM only and be lost when the app exits.

The `SYSTEM` preference filename is hardcoded as `/etc/fltk/$(vendor)/$(application).prefs` .
For backward compatibility, the old `USER.prefs` file naming scheme `$(directory)/.fltk/$(vendor)/$(application` is checked first. If that file does not exist, the environment variable `$XDG_CONFIG_HOME` is read as a base directory. If `$XDG_CONFIG_HOME` is not set, the base directory defaults to `$HOME/.config/`.
The user preferences will be stored in `$(directory)/$(vendor)/$(application).prefs`. The user data path will be `$(directory)/$(vendor)/$(application)/`.
In FLTK versions before 1.4.0, if `$HOME` was not set, the `USER` path would be empty, generating `$(vendor)/$(application).prefs`, which was used relative to the current working directory.

Note

On **macOS**, the `USER` directory is constructed by reading `$HOME` . If `$HOME` is not set or not pointing to an existing directory, we check the path returned by `NSHomeDirectory()` , and finally checking the path member of the passwd struct returned by `getpwuid(getuid())` . If all attempts fail, data will be stored in RAM only and be lost when the app exits. The filename and path is then constructed as `$(directory)/Library/Preferences/$(vendor)/$(application).prefs` . The `SYSTEM` directory is hardcoded as `/Library/Preferences/$(vendor)/$(application).prefs` .

In FLTK versions before 1.4.0, if `$HOME` was not set, the `USER` path

would be `NULL` , generating `<null>/Library/Preferences/$(vendor)/$(application).prefs`, which would silently fail to create a preference file.

Parameters

in	*root*	can be `USER_L` or `SYSTEM_L` for user specific or system wide preferences, add the `CLEAR` flag to start with a clean set of preferences instead of reading them from a preexisting database
in	*vendor*	unique text describing the company or author of this file, must be a valid filepath segment
in	*application*	unique text describing the application, must be a valid filepath segment

See also

Fl_Preferences(Fl_Preferences *parent, const char *group) with parent set to NULL

13.101.4.2 Fl_Preferences() [2/8]

```
Fl_Preferences::Fl_Preferences (
```

```
const char * path,
const char * vendor,
const char * application,
Root flags )
```

Use this constructor to create or read a preference file at an arbitrary position in the file system.
The file name is generated in the form `$(path)/$(application).prefs`. If `application` is `NULL`, `path` is taken literally as the file path and name.

```
// Example: read from an existing database and write modifications when flushed
// or destructor is called
Fl_Preferences database("/user/matt/test.prefs", "org.fltk.test", NULL,
                        Fl_Preferences::C_LOCALE);

// Example: create a new preferences file with an empty data set
Fl_Preferences database("/user/matt/test.prefs", "org.fltk.test", NULL,
     (Fl_Preferences::Root)(Fl_Preferences::C_LOCALE|Fl_Preferences::CLEAR));
```

Note

the C_LOCALE flag is is not set by default for backward compatibility, but it is highly recommended to set it when opening a database.

Parameters

in	*path*	path to the directory that contains the preference file
in	*vendor*	unique text describing the company or author of this file, must be a valid file path segment
in	*application*	unique text describing the application, must be a valid filename or NULL
in	*flags*	C_LOCALE to make the preferences file independent of the current locale, add the CLEAR flag to start with a clean set of preferences instead of reading from the database

13.101.4.3 Fl_Preferences() [3/8]

```
Fl_Preferences::Fl_Preferences (
            Fl_Preferences & parent,
            const char * group )
```

Generate or read a new group of entries within another group.
Use the `group` argument to name the group that you would like to access. `Group` can also contain a path to a group further down the hierarchy by separating group names with a forward slash '/'.

Parameters

in	*parent*	reference object for the new group
in	*group*	name of the group to access (may contain '/'s)

13.101.4.4 Fl_Preferences() [4/8]

```
Fl_Preferences::Fl_Preferences (
            Fl_Preferences * parent,
            const char * group )
```

Create or access a group of preferences using a name.
Parent should point to a previously created parent preferences group to create a preferences hierarchy.
If `parent` is set to `NULL`, an unnamed database will be accessed that exists only in local memory and is not associated with a file on disk. The root type of this database is set to `Fl_Preferences::MEMORY`.

- the memory database is *not* shared among multiple instances of the same app
- memory databases are *not* thread safe
- all data will be lost when the app quits

```
void some_function() {
  Fl_Preferences guide( NULL, "Guide" );
  guide.set("answer", 42);
}
void other_function() {
  int x;
  Fl_Preferences guide( NULL, "Guide" );
  guide.get("answer", x, -1);
}
```

FLTK uses the memory database to manage plugins. See `Fl_Plugin`.

Parameters

in	*parent*	the parameter parent is a pointer to the parent group. If `parent` is `NULL`, the new preferences item refers to an application internal database ("runtime prefs") which exists only once, and remains in RAM only until the application quits. This database is used to manage plugins and other data indexes by strings. Runtime prefs are *not* thread-safe.
in	*group*	a group name that is used as a key into the database

See also

Fl_Preferences(Fl_Preferences&, const char *group)

13.101.4.5 Fl_Preferences() [5/8]

```
Fl_Preferences::Fl_Preferences (
            Fl_Preferences & parent,
            int groupIndex )
```

Open a child group using a given index.
Use the `groupIndex` argument to find the group that you would like to access. If the given index is invalid (negative or too high), a new group is created with a UUID as a name.
The index needs to be fixed. It is currently backward. Index 0 points to the last member in the 'list' of preferences.

Parameters

in	*parent*	reference object for the new group
in	*groupIndex*	zero based index into child groups

13.101.4.6 Fl_Preferences() [6/8]

```
Fl_Preferences::Fl_Preferences (
            Fl_Preferences * parent,
            int groupIndex )
```

See also

Fl_Preferences(Fl_Preferences&, int groupIndex)

13.101.4.7 Fl_Preferences() [7/8]

```
Fl_Preferences::Fl_Preferences (
            Fl_Preferences::ID id )
```

Create a new dataset access point using a dataset ID.
ID's are a great way to remember shortcuts to database entries that are deeply nested in a preferences database, as long as the database root is not deleted. An ID can be retrieved from any Fl_Preferences dataset, and can then be used to create multiple new references to the same dataset.
ID's can be very helpful when put into the `user_data()` field of widget callbacks.

13.101.4.8 ~Fl_Preferences()

```
Fl_Preferences::~Fl_Preferences ( ) [virtual]
```

The destructor removes allocated resources.
When used on the *base* preferences group, the destructor flushes all changes to the preference file and deletes all internal databases.
The destructor does not remove any data from the database. It merely deletes your reference to the database.

13.101.4.9 Fl_Preferences() [8/8]

```
Fl_Preferences::Fl_Preferences (
            const char * path,
            const char * vendor,
            const char * application )
```

Deprecated: Use this constructor to create or read a preference file at an arbitrary position in the file system.

Deprecated "since 1.4.0 - use Fl_Preferences(path, vendor, application, flags) instead"

This constructor should no longer be used because the generated database uses the current locale, making it impossible to exchange floating point settings between machines with different language settings.
Use `Fl_Preferences(path, vendor, application, Fl_Preferences::C_LOCALE)` in new projects and `Fl_Preferences(path, vendor, application, 0)` if you must keep backward compatibility.

See also

Fl_Preferences(const char *path, const char *vendor, const char *application, Root flags)

13.101.5 Member Function Documentation

13.101.5.1 delete_entry()

```
char Fl_Preferences::delete_entry (
            const char * key )
```

Deletes a single name/value pair.
This function removes the entry `key` from the database.

Parameters

in	*key*	name of entry to delete

Returns

0 if deleting the entry failed

13.101.5.2 delete_group()

```
char Fl_Preferences::delete_group (
            const char * group )
```

Deletes a group.
Removes a group and all keys and groups within that group from the database.

Parameters

in	*group*	name of the group to delete

Returns

0 if call failed

13.101.5.3 dirty()

```
int Fl_Preferences::dirty ( )
```

Check if there were changes to the database that need to be written to disk.

Returns

1 if the database will be written to disk by `flush` or destructor.

0 if the database is unchanged since the last write operation.

-1 f there is an internal database error.

13.101.5.4 entries()

```
int Fl_Preferences::entries ( )
```

Returns the number of entries (name/value pairs) in a group.

Returns

number of entries

13.101.5.5 entry()

```
const char * Fl_Preferences::entry (
            int index )
```

Returns the name of an entry.
There is no guaranteed order of entry names. The index must be within the range given by entries().

Parameters

in	*index*	number indexing the requested entry

Returns

pointer to value cstring

13.101.5.6 entry_exists()

```
char Fl_Preferences::entry_exists (
            const char * key )
```

Returns non-zero if an entry with this name exists.

Parameters

in	*key*	name of entry that is searched for

Returns

0 if entry was not found

13.101.5.7 file_access() [1/2]

```
unsigned int Fl_Preferences::file_access ( )  [static]
```

Return the current file access permissions for the FLTK preferences system.

See also

Fl_Preferences::file_access(unsigned int)

13.101.5.8 file_access() [2/2]

```
void Fl_Preferences::file_access (
            unsigned int flags )  [static]
```

Tell the FLTK preferences system which files in the file system it may read, create, or write.

The FLTK core library will try to read or even create or write preference files when calling Fl::option(), Fl_File_Chooser, the printing panel, and possibly some other internal functions. If your application wants to keep FLTK from touching the file system, call this function before making any other FLTK calls:

```
// neither FLTK nor the app may read, create, or write preference files
Fl_Preferences::file_access( Fl_Preferences::NONE );
```

or

```
// FLTK may not read, create, or write preference files, but the application may
Fl_Preferences::file_access( Fl_Preferences::APP_OK );
```

All flags can be combined using an OR operator. If flags are not set, that specific access to the file system will not be allowed. By default, all access is granted. To clear one or more flags from the default setting, use:

```
Fl_Preferences::file_access( Fl_Preferences::file_access()
                             &~ Fl_Preferences::SYSTEM_WRITE );
```

If preferences are created using a filename (instead of Fl_Preferences::USER or Fl_Preferences::SYSTEM), file access is handled as if the Fl_Preferences::USER flag was set.

See also

Fl_Preferences::NONE and others for a list of flags.

Fl_Preferences::file_access()

13.101.5.9 filename() [1/2]

```
Fl_Preferences::Root Fl_Preferences::filename (
            char * buffer,
            size_t buffer_size )
```

Return the file name and path to the preference file.

If the preferences have not changed or have not been flushed, the file or directory may not have been created yet.

Parameters

out	*buffer*	write the resulting path into this buffer
in	*buffer_size*	size of the `buffer` in bytes

Returns

the root type at creation type, or MEMORY for runtime prefs, it does not return CORE or LOCALE flags.

13.101.5.10 filename() [2/2]

```
Fl_Preferences::Root Fl_Preferences::filename (
            char * buffer,
            size_t buffer_size,
            Root root,
            const char * vendor,
            const char * application )  [static]
```

Determine the file name and path to preferences that would be opened with these parameters.

Find the possible location of a preference file on disk without touching any of the pathname components. This can be used to check if a preference file already exists.

Parameters

out	*buffer*	write the resulting path into this buffer
in	*buffer_size*	size of the `buffer` in bytes
in	*root*	can be `USER_L` or `SYSTEM_L` for user specific or system wide preferences
in	*vendor*	unique text describing the company or author of this file, must be a valid filepath segment
in	*application*	unique text describing the application, must be a valid filepath segment

Returns

the input root value, or Fl_Preferences::UNKNOWN_ROOT_TYPE if the path could not be determined.

See also

Fl_Preferences(Root root, const char *vendor, const char *application)

13.101.5.11 flush()

```
int Fl_Preferences::flush ( )
```

Writes preferences to disk if they were modified.
This method can be used to verify that writing a preference file went well. Deleting the base preferences object will also write the contents of the database to disk.

Returns

-1 if anything went wrong, i.e. file could not be opened, permissions blocked writing, etc.

0 if the file was written to disk. This does not check if the disk ran out of space and the file is truncated.

1 if no data was written to the database and no write attempt to disk was made.

13.101.5.12 get() [1/9]

```
char Fl_Preferences::get (
            const char * key,
            char *& text,
            const char * defaultValue )
```

Reads an entry from the group.
A default value must be supplied. The return value indicates if the value was available (non-zero) or the default was used (0). get() allocates memory of sufficient size to hold the value. The buffer must be free'd by the developer using 'free(value)'.

Parameters

in	*key*	name of entry
out	*text*	returned from preferences or default value if none was set
in	*defaultValue*	default value to be used if no preference was set

Returns

0 if the default value was used

13.101.5.13 get() [2/9]

```
char Fl_Preferences::get (
            const char * key,
            char * text,
```

```
            const char * defaultValue,
            int maxSize )
```

Reads an entry from the group.
A default value must be supplied. The return value indicates if the value was available (non-zero) or the default was used (0). 'maxSize' is the maximum length of text that will be read. The text buffer must allow for one additional byte for a trailing zero.

Parameters

in	*key*	name of entry
out	*text*	returned from preferences or default value if none was set
in	*defaultValue*	default value to be used if no preference was set
in	*maxSize*	maximum length of value plus one byte for a trailing zero

Returns

0 if the default value was used

13.101.5.14 get() [3/9]

```
char Fl_Preferences::get (
            const char * key,
            double & value,
            double defaultValue )
```

Reads an entry from the group.
A default value must be supplied. The return value indicates if the value was available (non-zero) or the default was used (0).

Parameters

in	*key*	name of entry
out	*value*	returned from preferences or default value if none was set
in	*defaultValue*	default value to be used if no preference was set

Returns

0 if the default value was used

13.101.5.15 get() [4/9]

```
char Fl_Preferences::get (
            const char * key,
            float & value,
            float defaultValue )
```

Reads an entry from the group.
A default value must be supplied. The return value indicates if the value was available (non-zero) or the default was used (0).

Parameters

in	*key*	name of entry
out	*value*	returned from preferences or default value if none was set
in	*defaultValue*	default value to be used if no preference was set

Returns

0 if the default value was used

13.101.5.16 get() [5/9]

```
char Fl_Preferences::get (
            const char * key,
            int & value,
            int defaultValue )
```

Reads an entry from the group.
A default value must be supplied. The return value indicates if the value was available (non-zero) or the default was used (0).

Parameters

in	*key*	name of entry
out	*value*	returned from preferences or default value if none was set
in	*defaultValue*	default value to be used if no preference was set

Returns

0 if the default value was used

13.101.5.17 get() [6/9]

```
char Fl_Preferences::get (
            const char * key,
            std::string & value,
            const std::string & defaultValue )
```

Reads an entry from the group.
A default value must be supplied. The return value indicates if the value was available (non-zero) or the default was used (0). NUL characters in std::string are treated as the end of the text.

Parameters

in	*key*	name of entry
out	*value*	returned from preferences or default value if none was set
in	*defaultValue*	default value to be used if no preference was set

Returns

0 if the default value was used

13.101.5.18 get() [7/9]

```
char Fl_Preferences::get (
            const char * key,
            void *& data,
            const void * defaultValue,
            int defaultSize )
```

Reads an entry from the group.
A default value must be supplied. The return value indicates if the value was available (non-zero) or the default was used (0). get() allocates memory of sufficient size to hold the value. The buffer must be free'd by the developer using 'free(value)'.

Parameters

in	*key*	name of entry
out	*data*	returned from preferences or default value if none was set
in	*defaultValue*	default value to be used if no preference was set
in	*defaultSize*	size of default value array

Returns

0 if the default value was used

13.101.5.19 get() [8/9]

```
char Fl_Preferences::get (
            const char * key,
            void * data,
            const void * defaultValue,
            int defaultSize,
            int * maxSize )
```

Reads a binary entry from the group, encoded in hexadecimal blocks.
A binary (not hex) default value can be supplied. The return value indicates if the value was available (non-zero) or the default was used (0). `maxSize` is the maximum length of text that will be read and returns the actual number of bytes read.

Parameters

in	*key*	name of entry
out	*data*	value returned from preferences or default value if none was set
in	*defaultValue*	default value to be used if no preference was set
in	*defaultSize*	size of default value array
in,out	*maxSize*	maximum length of value and actual number of bytes set

Returns

0 if the default value was used

13.101.5.20 get() [9/9]

```
char Fl_Preferences::get (
            const char * key,
            void * data,
            const void * defaultValue,
            int defaultSize,
            int maxSize )
```

Reads a binary entry from the group, encoded in hexadecimal blocks.

Parameters

in	*key*	name of entry
out	*data*	value returned from preferences or default value if none was set
in	*defaultValue*	default value
in	*defaultSize*	size of default value array
in	*maxSize*	maximum length of value, to receive the number of bytes read, use the function below instead.

Returns

0 if the default value was used

See also

Fl_Preferences::get(const char *key, void *data, const void *defaultValue, int defaultSize, int *maxSize)

13.101.5.21 get_userdata_path()

```
char Fl_Preferences::get_userdata_path (
            char * path,
            int pathlen )
```

Creates a path that is related to the preference file and that is usable for additional application data.
This function creates a directory that is named after the preferences database without the .prefs extension and located in the same directory. It then fills the given buffer with the complete path name.
There is no way to verify that the path name fit into the buffer. If the name is too long, it will be clipped.
This function can be used with direct paths that don't end in .prefs . *get_userdata_path()* will remove any extension and end the path with a `/` . If the file name has no extension, *get_userdata_path()* will append .data/ to the path name.
Example:

```
Fl_Preferences prefs( USER, "matthiasm.com", "test" );
char path[FL_PATH_MAX];
prefs.get_userdata_path( path, FL_PATH_MAX );
```

creates the preferences database in the directory (User 'matt' on Linux):

```
/Users/matt/.fltk/matthiasm.com/test.prefs
```

..and returns the userdata path:

```
/Users/matt/.fltk/matthiasm.com/test/
```

Parameters

out	*path*	buffer for user data path
in	*pathlen*	size of path buffer (should be at least `FL_PATH_MAX`)

Returns

1 if there is no filename (*path* will be unmodified)

1 if *pathlen* is 0 (*path* will be unmodified)

1 if a path was created successfully, *path* will contain the path name ending in a '/'

0 if path was not created for some reason; *path* will contain the pathname that could not be created

See also

Fl_Preferences::Fl_Preferences(Root, const char*, const char*)

13.101.5.22 group()

```
const char * Fl_Preferences::group (
            int num_group )
```

Returns the name of the Nth (`num_group`) group.
There is no guaranteed order of group names. The index must be within the range given by groups().

Parameters

in	*num_group*	number indexing the requested group

Returns

'C' string pointer to the group name

13.101.5.23 group_exists()

```
char Fl_Preferences::group_exists (
            const char * key )
```

Returns non-zero if a group with this name exists.
Group names are relative to the Fl_Preferences node and can contain a path. "." describes the current node, "./" describes the topmost node. By preceding a groupname with a "./" its path becomes relative to the topmost node.

Parameters

in	*key*	name of group that is searched for

Returns

0 if no group by that name was found

13.101.5.24 groups()

```
int Fl_Preferences::groups ( )
```

Returns the number of groups that are contained within a group.

Returns

0 for no groups at all

13.101.5.25 new_UUID()

```
const char * Fl_Preferences::new_UUID ( )  [static]
```

Returns a UUID as generated by the system.
A UUID is a "universally unique identifier" which is commonly used in configuration files to create identities. A UUID in ASCII looks like this: `937C4900-51AA-4C11-8DD3-7AB59944F03E`. It has always 36 bytes plus a trailing zero.

Returns

a pointer to a static buffer containing the new UUID in ASCII format. The buffer is overwritten during every call to this function!

13.101.5.26 set() [1/8]

```
char Fl_Preferences::set (
            const char * key,
            const char * text )
```

Sets an entry (name/value pair).
The return value indicates if there was a problem storing the data in memory. However it does not reflect if the value was actually stored in the preference file.

Parameters

in	*key*	name of entry
in	*text*	set this entry to `value`

Returns

0 if setting the value failed

13.101.5.27 set() [2/8]

```
char Fl_Preferences::set (
            const char * entry,
            const std::string & value )
```

Sets an entry (name/value pair).
The return value indicates if there was a problem storing the data in memory. However it does not reflect if the value was actually stored in the preference file. NUL characters in std::string are treated as the end of the text.

Parameters

in	*entry*	name of entry
in	*value*	set this entry to value (stops at the first nul character).

Returns

0 if setting the value failed

13.101.5.28 set() [3/8]

```
char Fl_Preferences::set (
            const char * key,
            const void * data,
            int dsize )
```

Sets an entry (name/value pair).
The return value indicates if there was a problem storing the data in memory. However it does not reflect if the value was actually stored in the preference file.

Parameters

in	*key*	name of entry
in	*data*	set this entry to `value`
in	*dsize*	size of data array

Returns

0 if setting the value failed

13.101.5.29 set() [4/8]

```
char Fl_Preferences::set (
            const char * key,
            double value )
```

Sets an entry (name/value pair).
The return value indicates if there was a problem storing the data in memory. However it does not reflect if the value was actually stored in the preference file.

Parameters

in	*key*	name of entry
in	*value*	set this entry to `value`

Returns

0 if setting the value failed

13.101.5.30 set() [5/8]

```
char Fl_Preferences::set (
            const char * key,
            double value,
            int precision )
```

Sets an entry (name/value pair).
The return value indicates if there was a problem storing the data in memory. However it does not reflect if the value was actually stored in the preference file.

Parameters

in	*key*	name of entry
in	*value*	set this entry to `value`
in	*precision*	number of decimal digits to represent value

Returns

0 if setting the value failed

13.101.5.31 set() [6/8]

```
char Fl_Preferences::set (
            const char * key,
            float value )
```

Sets an entry (name/value pair).
The return value indicates if there was a problem storing the data in memory. However it does not reflect if the value was actually stored in the preference file.

Parameters

in	*key*	name of entry
in	*value*	set this entry to `value`

Returns

0 if setting the value failed

13.101.5.32 set() [7/8]

```
char Fl_Preferences::set (
            const char * key,
            float value,
            int precision )
```

Sets an entry (name/value pair).
The return value indicates if there was a problem storing the data in memory. However it does not reflect if the value was actually stored in the preference file.

Parameters

in	*key*	name of entry
in	*value*	set this entry to `value`
in	*precision*	number of decimal digits to represent value

Returns

0 if setting the value failed

13.101.5.33 set() [8/8]

```
char Fl_Preferences::set (
            const char * key,
            int value )
```

Sets an entry (name/value pair).
The return value indicates if there was a problem storing the data in memory. However it does not reflect if the value was actually stored in the preference file.

Parameters

in	*key*	name of entry
in	*value*	set this entry to `value`

Returns

0 if setting the value failed

13.101.5.34 size()

```
int Fl_Preferences::size (
            const char * key )
```

Returns the size of the value part of an entry.

Parameters

in	*key*	name of entry

Returns

size of value

13.101.6 Member Data Documentation

13.101.6.1 CORE_READ_OK

```
const unsigned int Fl_Preferences::CORE_READ_OK = 0x0010  [static]
```

Set this if it is OK for FLTK to read preference files.
USER_READ_OK and/or SYSTEM_READ_OK must also be set.

13.101.6.2 CORE_WRITE_OK

```
const unsigned int Fl_Preferences::CORE_WRITE_OK = 0x0020  [static]
```

Set this if it is OK for FLTK to create or write preference files.
USER_WRITE_OK and/or SYSTEM_WRITE_OK must also be set.

13.101.6.3 NONE

```
const unsigned int Fl_Preferences::NONE = 0x0000  [static]
```

Set this if no call to Fl_Preferences shall access the file system.

See also

Fl_Preferences::file_access(unsigned int)

Fl_Preferences::file_access()

The documentation for this class was generated from the following files:

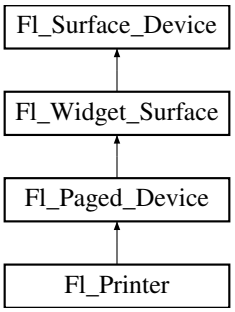
- Fl_Preferences.H
- Fl_Preferences.cxx

13.102 Fl_Printer Class Reference

OS-independent print support.

`#include <Fl_Printer.H>`

Inheritance diagram for Fl_Printer:



Public Member Functions

- int begin_job (int pagecount=0, int ∗frompage=NULL, int ∗topage=NULL, char ∗∗perr_message=NULL) override

  Begins a print job.
- int begin_page (void) override

  Begins a new printed page.
- void end_job (void) override

  To be called at the end of a print job.
- int end_page (void) override

  To be called at the end of each page.
- **Fl_Printer** (void)

  The constructor.
- bool is_current () override

  Is this surface the current drawing surface?
- void margins (int ∗left, int ∗top, int ∗right, int ∗bottom) override

  Computes the dimensions of margins that lie between the printable page area and the full page.
- void origin (int ∗x, int ∗y) override

  Computes the coordinates of the current origin of graphics functions.
- void origin (int x, int y) override

  Sets the position of the origin of graphics in the drawable part of the drawing surface.
- int printable_rect (int ∗w, int ∗h) override

  Computes the width and height of the drawable area of the drawing surface.
- void rotate (float angle) override

  Rotates the graphics operations relatively to paper.
- void scale (float scale_x, float scale_y=0.) override

  Changes the scaling of page coordinates.

- void set_current (void) override

  Make this surface the current drawing surface.
- void translate (int x, int y) override

  Translates the current graphics origin accounting for the current rotation.
- void untranslate (void) override

  Undoes the effect of a previous translate() call.
- ~**Fl_Printer** (void)

  The destructor.

Public Member Functions inherited from Fl_Paged_Device

- void **print_widget** (Fl_Widget *widget, int delta_x=0, int delta_y=0)

  Synonym of draw(Fl_Widget, int, int)*
- void **print_window** (Fl_Window *win, int x_off=0, int y_off=0)

  Synonym of draw_decorated_window(Fl_Window, int, int)*
- int start_job (int pagecount=0, int *frompage=NULL, int *topage=NULL, char **perr_message=NULL)

  *Synonym of begin_job(int pagecount, int *frompage, int *topage, char **perr_message).*
- int start_page ()

  Synonym of begin_page().
- virtual ~**Fl_Paged_Device** ()

  The destructor.

Public Member Functions inherited from Fl_Widget_Surface

- void draw (Fl_Widget *widget, int delta_x=0, int delta_y=0)

  Draws the widget on the drawing surface.
- void draw_decorated_window (Fl_Window *win, int x_offset=0, int y_offset=0)

  Draws a window with its title bar and frame if any.
- void print_window_part (Fl_Window *win, int x, int y, int w, int h, int delta_x=0, int delta_y=0)

  Draws a rectangular part of an on-screen window.

Public Member Functions inherited from Fl_Surface_Device

- Fl_Graphics_Driver * **driver** ()

  Returns the graphics driver of this drawing surface.
- virtual ~**Fl_Surface_Device** ()

  The destructor.

Static Public Attributes

These attributes are useful for the Linux/Unix platform only.

- static const char * **dialog_title** = "Print"

  [this text may be customized at run-time]
- static const char * **dialog_printer** = "Printer:"

  [this text may be customized at run-time]
- static const char * **dialog_range** = "Print Range"

  [this text may be customized at run-time]
- static const char * **dialog_copies** = "Copies"

  [this text may be customized at run-time]
- static const char * **dialog_all** = "All"

  [this text may be customized at run-time]
- static const char * **dialog_pages** = "Pages"

  [this text may be customized at run-time]
- static const char * **dialog_from** = "From:"

[this text may be customized at run-time]

- static const char * **dialog_to** = "To:"

  [this text may be customized at run-time]
- static const char * **dialog_properties** = "Properties..."

  [this text may be customized at run-time]
- static const char * **dialog_copyNo** = "# Copies:"

  [this text may be customized at run-time]
- static const char * **dialog_print_button** = "Print"

  [this text may be customized at run-time]
- static const char * **dialog_cancel_button** = "Cancel"

  [this text may be customized at run-time]
- static const char * **dialog_print_to_file** = "Print To File"

  [this text may be customized at run-time]
- static const char * **property_title** = "Printer Properties"

  [this text may be customized at run-time]
- static const char * **property_pagesize** = "Page Size:"

  [this text may be customized at run-time]
- static const char * **property_mode** = "Output Mode:"

  [this text may be customized at run-time]
- static const char * **property_use** = "Use"

  [this text may be customized at run-time]
- static const char * **property_save** = "Save"

  [this text may be customized at run-time]
- static const char * **property_cancel** = "Cancel"

  [this text may be customized at run-time]

Static Public Attributes inherited from Fl_Paged_Device

- static const page_format **page_formats** [NO_PAGE_FORMATS]

  width, height and name of all elements of the enum Page_Format.

Additional Inherited Members

Public Types inherited from Fl_Paged_Device

- enum Page_Format {
  A0 = 0 , A1 , A2 , A3 ,
  A4 , A5 , A6 , A7 ,
  A8 , A9 , B0 , B1 ,
  B2 , B3 , B4 , B5 ,
  B6 , B7 , B8 , B9 ,
  B10 , **C5E** , **DLE** , EXECUTIVE ,
  FOLIO , LEDGER , LEGAL , LETTER ,
  TABLOID , **ENVELOPE** , **MEDIA** = 0x1000 }

  Possible page formats.
- enum Page_Layout { PORTRAIT = 0 , LANDSCAPE = 0x100 , REVERSED = 0x200 , ORIENTATION = 0x300 }

  Possible page layouts.

Static Public Member Functions inherited from Fl_Surface_Device

- static Fl_Surface_Device * pop_current ()

  Removes the top element from the current drawing surface stack, and makes the new top element current.
- static void push_current (Fl_Surface_Device *new_current)

  Pushes `new_current` on top of the stack of current drawing surfaces, and makes it current.
- static Fl_Surface_Device * surface ()

  The current drawing surface.

Protected Member Functions inherited from Fl_Paged_Device

- **Fl_Paged_Device** ()

  The constructor.

Protected Member Functions inherited from Fl_Widget_Surface

- Fl_Widget_Surface (Fl_Graphics_Driver *d)

  The constructor.

Protected Member Functions inherited from Fl_Surface_Device

- void **driver** (Fl_Graphics_Driver *graphics_driver)

  Sets the graphics driver of this drawing surface.
- virtual void end_current ()

  FLTK calls this each time a surface ceases to be the current drawing surface.
- **Fl_Surface_Device** (Fl_Graphics_Driver *graphics_driver)

  Constructor that sets the graphics driver to use for the created surface.

Protected Attributes inherited from Fl_Widget_Surface

- int **x_offset**

  horizontal offset to the origin of graphics coordinates
- int **y_offset**

  vertical offset to the origin of graphics coordinates

13.102.1 Detailed Description

OS-independent print support.
Fl_Printer allows to use all drawing, color, text, image, and clip FLTK functions, and to have them operate on printed page(s). There are two main, non exclusive, ways to use it.

- Print any widget (standard, custom, Fl_Window, Fl_Gl_Window) as it appears on screen, with optional translation, scaling and rotation. This is done by calling print_widget(), print_window() or print_window_part().
- Use a series of FLTK graphics commands (e.g., font, text, lines, colors, clip, image) to compose a page appropriately shaped for printing.

In both cases, begin by begin_job(), begin_page(), printable_rect() and origin() calls and finish by end_page() and end_job() calls.
Example of use: print a widget centered in a page

```
#include <FL/Fl_Printer.H>
#include <FL/fl_draw.H>
int width, height;
Fl_Widget *widget = ...  // a widget we want printed
Fl_Printer *printer = new Fl_Printer();
if (printer->begin_job(1) == 0) {
 printer->begin_page();
 printer->printable_rect(&width, &height);
 fl_color(FL_BLACK);
 fl_line_style(FL_SOLID, 2);
 fl_rect(0, 0, width, height);
 fl_font(FL_COURIER, 12);
 time_t now; time(&now); fl_draw(ctime(&now), 0, fl_height());
 printer->origin(width/2, height/2);
 printer->print_widget(widget, -widget->w()/2, -widget->h()/2);
 printer->end_page();
 printer->end_job();
}
delete printer;
```

Recommended method to refresh GUI while printing :

```
printer->begin_job(0);
......
Fl_Surface_Device::push_current(Fl_Display_Device::display_device());
Fl::check(); // or any operation that draws to display
Fl_Surface_Device::pop_current();
```

```
.....
printer->end_job();
```

Platform specifics

- X11 and Wayland platforms:
  - FLTK expresses all graphics data using (Level 2) PostScript and sends that to the selected printer. See class Fl_PostScript_File_Device for a description of how text and transparent images appear in print.
  - If the GTK library is available at run-time, class Fl_Printer runs GTK's printer dialog which allows to set printer, paper size and orientation.
  - If the GTK library is not available, or if Fl::option(Fl::OPTION_PRINTER_USES_GTK) has been turned off, class Fl_Printer runs FLTK's print dialog.
    * Unless it has been previously changed, the default paper size is A4. To change that, press the "Properties" button of the "Print" dialog window opened by an Fl_Printer::begin_job() call. This opens a "Printer Properties" window where it's possible to select the adequate paper size. Finally press the "Save" button therein to assign the chosen paper size to the chosen printer for this and all further print operations.
    * Use the static public attributes of this class to set the print dialog to other languages than English. For example, the "Printer:" dialog item Fl_Printer::dialog_printer can be set to French with:
      `Fl_Printer::dialog_printer = "Imprimante:";`
      before creation of the Fl_Printer object.
    * Use Fl_PostScript_File_Device::file_chooser_title to customize the title of the file chooser dialog that opens when using the "Print To File" option of the print dialog.
- Windows platform: Transparent Fl_RGB_Image 's don't print with exact transparency on most printers (a workaround is to use print_window_part()). Fl_RGB_Image 's don't rotate() well.
- Mac OS X platform: all graphics requests print as on display and accept rotation and scaling.

13.102.2 Member Function Documentation

13.102.2.1 begin_job()

```
int Fl_Printer::begin_job (
            int pagecount = 0,
            int * frompage = NULL,
            int * topage = NULL,
            char ** perr_message = NULL )  [override], [virtual]
```

Begins a print job.

Parameters

in	*pagecount*	the total number of pages of the job (or 0 if you don't know the number of pages)
out	*frompage*	if non-null, ∗frompage is set to the first page the user wants printed
out	*topage*	if non-null, ∗topage is set to the last page the user wants printed
out	*perr_message*	if non-null and if the returned value is 2, ∗perr_message is set to a string describing the error. That string can be delete[]'d after use.

Returns

0 if OK, 1 if user cancelled the job, 2 if any error.

Reimplemented from Fl_Paged_Device.

13.102.2.2 begin_page()

```
int Fl_Printer::begin_page (
            void )  [override], [virtual]
```

Begins a new printed page.

The page coordinates are initially in points, i.e., 1/72 inch, and with origin at the top left of the printable page area. This function also makes this surface the current drawing surface with Fl_Surface_Device::push_current().

Note

begin_page() calls Fl_Surface_Device::push_current() and leaves this device as the active surface. If any calls between begin_page() and end_page() open dialog boxes or will otherwise draw into FLTK windows, those calls must be put between a call to Fl_Surface_Device::pop_current() and a call to Fl_Surface_Device::push_current(), or the content of the dialog box will be rendered to the printer instead of the screen.

Returns

0 if OK, non-zero if any error

Reimplemented from Fl_Paged_Device.

13.102.2.3 end_job()

```
void Fl_Printer::end_job (
            void ) [override], [virtual]
```

To be called at the end of a print job.
Reimplemented from Fl_Paged_Device.

13.102.2.4 end_page()

```
int Fl_Printer::end_page (
            void ) [override], [virtual]
```

To be called at the end of each page.
This function also stops this surface from being the current drawing surface with Fl_Surface_Device::pop_current().

Note

end_page() calls Fl_Surface_Device::pop_current(). If any calls between begin_page() and end_page() open dialog boxes or will otherwise draw into FLTK windows, those calls must be put between a call to Fl_Surface_Device::pop_current() and a call to Fl_Surface_Device::push_current().

Returns

0 if OK, non-zero if any error.

Reimplemented from Fl_Paged_Device.

13.102.2.5 is_current()

```
bool Fl_Printer::is_current ( ) [override], [virtual]
```

Is this surface the current drawing surface?
Reimplemented from Fl_Surface_Device.

13.102.2.6 margins()

```
void Fl_Printer::margins (
            int * left,
            int * top,
            int * right,
            int * bottom ) [override], [virtual]
```

Computes the dimensions of margins that lie between the printable page area and the full page.
Values are in the same unit as that used by FLTK drawing functions. They are changed by scale() calls.

Parameters

out	*left*	If non-null, ∗left is set to the left margin size.
out	*top*	If non-null, ∗top is set to the top margin size.
out	*right*	If non-null, ∗right is set to the right margin size.
out	*bottom*	If non-null, ∗bottom is set to the bottom margin size.

Reimplemented from Fl_Paged_Device.

13.102.2.7 origin() [1/2]

```
void Fl_Printer::origin (
            int * x,
            int * y )  [override], [virtual]
```

Computes the coordinates of the current origin of graphics functions.

Parameters

out	*x,y*	If non-null, *x and *y are set to the horizontal and vertical coordinates of the graphics origin.

Reimplemented from Fl_Widget_Surface.

13.102.2.8 origin() [2/2]

```
void Fl_Printer::origin (
            int x,
            int y )  [override], [virtual]
```

Sets the position of the origin of graphics in the drawable part of the drawing surface.
Arguments should be expressed relatively to the result of a previous printable_rect() call. That is, `printable_rect(&w, &h); origin(w/2, 0);` sets the graphics origin at the top center of the drawable area. Successive origin() calls don't combine their effects. Origin() calls are not affected by rotate() calls (for classes derived from Fl_Paged_Device).

Parameters

in	*x,y*	Horizontal and vertical positions in the drawing surface of the desired origin of graphics.

Reimplemented from Fl_Widget_Surface.

13.102.2.9 printable_rect()

```
int Fl_Printer::printable_rect (
            int * w,
            int * h )  [override], [virtual]
```

Computes the width and height of the drawable area of the drawing surface.
Values are in the same unit as that used by FLTK drawing functions and are unchanged by calls to origin(). If the object is derived from class Fl_Paged_Device, values account for the user-selected paper type and print orientation and are changed by scale() calls.

Returns

0 if OK, non-zero if any error

Reimplemented from Fl_Widget_Surface.

13.102.2.10 rotate()

```
void Fl_Printer::rotate (
            float angle )  [override], [virtual]
```

Rotates the graphics operations relatively to paper.
The rotation is centered on the current graphics origin. Successive rotate() calls don't combine their effects.

Parameters

angle	Rotation angle in counter-clockwise degrees.

Reimplemented from Fl_Paged_Device.

13.102.2.11 scale()

```
void Fl_Printer::scale (
            float scale_x,
            float scale_y = 0.  )  [override], [virtual]
```

Changes the scaling of page coordinates.
This function also resets the origin of graphics functions at top left of printable page area. After a scale() call, do a printable_rect() call to get the new dimensions of the printable page area. Successive scale() calls don't combine their effects.

Parameters

scale↩ _x	Horizontal dimensions of plot are multiplied by this quantity.
scale↩ _y	Same as above, vertically. The value 0. is equivalent to setting `scale_y` = `scale_x`. Thus, scale(factor); is equivalent to scale(factor, factor);

Reimplemented from Fl_Paged_Device.

13.102.2.12 set_current()

```
void Fl_Printer::set_current (
            void  )  [override], [virtual]
```

Make this surface the current drawing surface.
This surface will receive all future graphics requests.
Since FLTK 1.4.0 the preferred API to change the current drawing surface is Fl_Surface_Device::push_current( ) / Fl_Surface_Device::pop_current().

Note

It is recommended to use this function only as follows :

- The current drawing surface is the display;
- make current another surface, e.g., an Fl_Printer or an Fl_Image_Surface object, calling set_current() on this object;
- draw to that surface;
- make the display current again with Fl_Display_Device::display_device()->set_current();
  don't do any other call to set_current() before this one.

Other scenarios of drawing surface changes should be performed via Fl_Surface_Device::push_current() and Fl_Surface_Device::pop_current().
Reimplemented from Fl_Surface_Device.

13.102.2.13 translate()

```
void Fl_Printer::translate (
            int x,
            int y )  [override], [virtual]
```

Translates the current graphics origin accounting for the current rotation.
Each translate() call must be matched by an untranslate() call. Successive translate() calls add up their effects.
Reimplemented from Fl_Widget_Surface.

13.102.2.14 untranslate()

```
void Fl_Printer::untranslate (
            void  )  [override], [virtual]
```

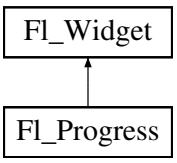
Undoes the effect of a previous translate() call.

Reimplemented from Fl_Widget_Surface.
The documentation for this class was generated from the following files:

- Fl_Printer.H
- Fl_Printer.cxx

13.103 Fl_Progress Class Reference

Displays a progress bar for the user.
`#include <Fl_Progress.H>`
Inheritance diagram for Fl_Progress:



Public Member Functions

- Fl_Progress (int x, int y, int w, int h, const char *l=0)

  The constructor creates the progress bar using the position, size, and label.
- float maximum () const

  Gets the maximum value in the progress widget.
- void maximum (float v)

  Sets the maximum value in the progress widget.
- float minimum () const

  Gets the minimum value in the progress widget.
- void minimum (float v)

  Sets the minimum value in the progress widget.
- float value () const

  Gets the current value in the progress widget.
- void value (float v)

  Sets the current value in the progress widget.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.

- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.

- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual int handle (int event)

  Handles the specified event.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

Sets the image to use as part of the widget label when in the active state.

- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.

- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

Sets the user data for this widget.

- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void draw () override

  Draws the progress bar.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

Draws a focus box for the widget at the given position and size.

- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.103.1 Detailed Description

Displays a progress bar for the user.

13.103.2 Constructor & Destructor Documentation

13.103.2.1 Fl_Progress()

```
Fl_Progress::Fl_Progress (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

The constructor creates the progress bar using the position, size, and label.
You can set the background color with color() and the progress bar color with selection_color(), or you can set both colors together with color(unsigned bg, unsigned sel).
The default colors are FL_BACKGROUND2_COLOR and FL_YELLOW, resp.

13.103.3 Member Function Documentation

13.103.3.1 draw()

```
void Fl_Progress::draw (
            void ) [override], [protected], [virtual]
```

Draws the progress bar.
Implements Fl_Widget.

13.103.3.2 maximum() [1/2]

```
float Fl_Progress::maximum ( ) const [inline]
```

Gets the maximum value in the progress widget.

13.103.3.3 maximum() [2/2]

```
void Fl_Progress::maximum (
            float v ) [inline]
```

Sets the maximum value in the progress widget.

13.103.3.4 minimum() [1/2]

```
float Fl_Progress::minimum ( ) const [inline]
```

Gets the minimum value in the progress widget.

13.103.3.5 minimum() [2/2]

```
void Fl_Progress::minimum (
            float v ) [inline]
```

Sets the minimum value in the progress widget.

13.103.3.6 value() [1/2]

```
float Fl_Progress::value ( ) const [inline]
```

Gets the current value in the progress widget.

13.103.3.7 value() [2/2]

```
void Fl_Progress::value (
            float v ) [inline]
```

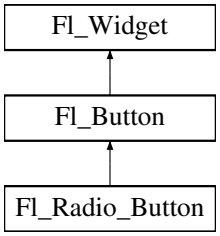
Sets the current value in the progress widget.

The documentation for this class was generated from the following files:

- Fl_Progress.H
- Fl_Progress.cxx

13.104 Fl_Radio_Button Class Reference

Inheritance diagram for Fl_Radio_Button:



Public Member Functions

- Fl_Radio_Button (int X, int Y, int W, int H, const char ∗L=0)

  The constructor creates the button using the given position, size, and label.

Public Member Functions inherited from Fl_Button

- int clear ()

  Same as `value(0)`.
- uchar compact () const

  Return true if buttons are rendered as compact buttons.
- void compact (uchar v)

  Decide if buttons should be rendered in compact mode.
- Fl_Boxtype down_box () const

  Returns the current down box type, which is drawn when value() is non-zero.
- void down_box (Fl_Boxtype b)

  Sets the down box type.
- Fl_Color **down_color** () const

  (for backwards compatibility)
- void **down_color** (unsigned c)

  (for backwards compatibility)
- Fl_Button (int X, int Y, int W, int H, const char ∗L=0)

  The constructor creates the button using the given position, size, and label.
- int handle (int) override

  Handles the specified event.
- int set ()

  Same as `value(1)`.
- void **setonly** ()

  Turns on this button and turns off all other radio buttons in the group (calling `value(1)` or `set()` does not do this).
- int shortcut () const

Returns the current shortcut key for the button.

- void **shortcut** (const char *s)

  (for backwards compatibility)
- void shortcut (int s)

  Sets the shortcut key to s.
- char **value** () const

  Returns the current value of the button (0 or 1).
- int value (int v)

  Sets the current value of the button.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.

- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

Sets the image to use as part of the widget label when in the inactive state.

- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.

- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

Makes a widget visible.

- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.

- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Button

- void draw () override

  Draws the widget.
- void **simulate_key_action** ()

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

Draws a focus rectangle around the widget.

- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Static Protected Member Functions inherited from Fl_Button

- static void **key_release_timeout** (void *)

Static Protected Attributes inherited from Fl_Button

- static Fl_Widget_Tracker * **key_release_tracker** = 0

13.104.1 Constructor & Destructor Documentation

13.104.1.1 Fl_Radio_Button()

```
Fl_Radio_Button::Fl_Radio_Button (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

The constructor creates the button using the given position, size, and label.
The Button type() is set to FL_RADIO_BUTTON.

Parameters

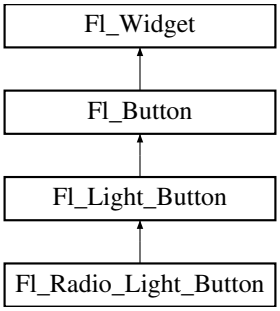
in	*X,Y,W,H*	position and size of the widget
in	*L*	widget label, default is no label

The documentation for this class was generated from the following files:

- Fl_Radio_Button.H
- Fl_Button.cxx

13.105 Fl_Radio_Light_Button Class Reference

Inheritance diagram for Fl_Radio_Light_Button:



Public Member Functions

- **Fl_Radio_Light_Button** (int X, int Y, int W, int H, const char ∗l=0)

Public Member Functions inherited from Fl_Light_Button

- Fl_Light_Button (int x, int y, int w, int h, const char ∗l=0)

  Creates a new Fl_Light_Button widget using the given position, size, and label string.
- int handle (int) override

  Handles the specified event.

Public Member Functions inherited from Fl_Button

- int clear ()

  Same as `value(0)`.
- uchar compact () const

  Return true if buttons are rendered as compact buttons.
- void compact (uchar v)

  Decide if buttons should be rendered in compact mode.
- Fl_Boxtype down_box () const

  Returns the current down box type, which is drawn when value() is non-zero.
- void down_box (Fl_Boxtype b)

  Sets the down box type.
- Fl_Color **down_color** () const

  (for backwards compatibility)
- void **down_color** (unsigned c)

  (for backwards compatibility)
- Fl_Button (int X, int Y, int W, int H, const char ∗L=0)

  The constructor creates the button using the given position, size, and label.
- int set ()

  Same as `value(1)`.
- void **setonly** ()

  Turns on this button and turns off all other radio buttons in the group (calling `value(1)` *or* `set()` *does not do this).*
- int shortcut () const

  Returns the current shortcut key for the button.
- void **shortcut** (const char ∗s)

  (for backwards compatibility)
- void shortcut (int s)

Sets the shortcut key to s.

- char **value** () const

  Returns the current value of the button (0 or 1).
- int value (int v)

  Sets the current value of the button.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.

- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

Returns whether the inactive image is managed by the widget.

- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.

- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

Gives the widget the keyboard focus.

- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Light_Button

- void draw () override

  Draws the widget.

Protected Member Functions inherited from Fl_Button

- void **simulate_key_action** ()

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

Draws a focus box for the widget at the given position and size.

- void draw_label () const

  Draws the widget's label at the defined label position.

- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.

- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.

- unsigned int **flags** () const

  Gets the widget flags mask.

- void h (int v)

  Internal use only.

- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.

- void w (int v)

  Internal use only.

- void x (int v)

  Internal use only.

- void y (int v)

  Internal use only.

Static Protected Member Functions inherited from Fl_Button

- static void **key_release_timeout** (void *)

Static Protected Attributes inherited from Fl_Button

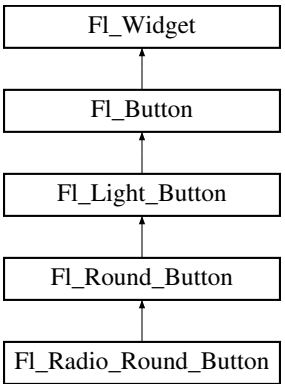
- static Fl_Widget_Tracker * **key_release_tracker** = 0

The documentation for this class was generated from the following files:

- Fl_Radio_Light_Button.H
- Fl_Light_Button.cxx

13.106 Fl_Radio_Round_Button Class Reference

Inheritance diagram for Fl_Radio_Round_Button:



Public Member Functions

- Fl_Radio_Round_Button (int X, int Y, int W, int H, const char *L=0)

  Creates a new Fl_Radio_Button widget using the given position, size, and label string.

Public Member Functions inherited from Fl_Round_Button

- Fl_Round_Button (int x, int y, int w, int h, const char *l=0)

  Creates a new Fl_Round_Button widget using the given position, size, and label string.

Public Member Functions inherited from Fl_Light_Button

- Fl_Light_Button (int x, int y, int w, int h, const char *l=0)

  Creates a new Fl_Light_Button widget using the given position, size, and label string.
- int handle (int) override

  Handles the specified event.

Public Member Functions inherited from Fl_Button

- int clear ()

  Same as `value(0)`.
- uchar compact () const

  Return true if buttons are rendered as compact buttons.
- void compact (uchar v)

  Decide if buttons should be rendered in compact mode.
- Fl_Boxtype down_box () const

  Returns the current down box type, which is drawn when value() is non-zero.
- void down_box (Fl_Boxtype b)

  Sets the down box type.
- Fl_Color **down_color** () const

  (for backwards compatibility)
- void **down_color** (unsigned c)

  (for backwards compatibility)
- Fl_Button (int X, int Y, int W, int H, const char *L=0)

  The constructor creates the button using the given position, size, and label.
- int set ()

  Same as `value(1)`.
- void **setonly** ()

  Turns on this button and turns off all other radio buttons in the group (calling `value(1)` or `set()` does not do this).
- int shortcut () const

  Returns the current shortcut key for the button.
- void **shortcut** (const char *s)

  (for backwards compatibility)
- void shortcut (int s)

  Sets the shortcut key to `s`.
- char **value** () const

  Returns the current value of the button (0 or 1).
- int value (int v)

  Sets the current value of the button.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.

- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.

- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

Get the spacing between the label and the horizontal edge of the widget.

- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.

- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

Gets the widget type.

- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Light_Button

- void draw () override

  Draws the widget.

Protected Member Functions inherited from Fl_Button

- void **simulate_key_action** ()

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

Internal use only.

- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Static Protected Member Functions inherited from Fl_Button

- static void **key_release_timeout** (void ∗)

Static Protected Attributes inherited from Fl_Button

- static Fl_Widget_Tracker ∗ **key_release_tracker** = 0

13.106.1 Constructor & Destructor Documentation

13.106.1.1 Fl_Radio_Round_Button()

```
Fl_Radio_Round_Button::Fl_Radio_Round_Button (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

Creates a new Fl_Radio_Button widget using the given position, size, and label string.
The button type() is set to FL_RADIO_BUTTON.

Parameters

in	*X,Y,W,H*	position and size of the widget
in	*L*	widget label, default is no label

The documentation for this class was generated from the following files:

- Fl_Radio_Round_Button.H
- Fl_Round_Button.cxx

13.107 Fl_Rect Class Reference

Rectangle with standard FLTK coordinates (X, Y, W, H).

```
#include <Fl_Rect.H>
```

Public Member Functions

- int b () const

  gets the bottom edge (y + h).
- void **b** (int B)

  sets the height based on B and y
- bool **contains** (int X, int Y) const
- **Fl_Rect** ()

  The default constructor creates an empty rectangle (x = y = w = h = 0).
- **Fl_Rect** (const Fl_Widget &widget)

This constructor creates a rectangle based on a widget's position and size.

- **Fl_Rect** (const Fl_Widget *const widget)

  This constructor creates a rectangle based on a widget's position and size.
- **Fl_Rect** (int W, int H)

  This constructor creates a rectangle with x = y = 0 and the given width and height.
- **Fl_Rect** (int X, int Y, int W, int H)

  This constructor creates a rectangle with the given x,y coordinates and the given width and height.
- Fl_Rect (int X, int Y, int W, int H, Fl_Boxtype bt)

  This constructor creates a rectangle with the given x,y coordinates and the given width and height reduced by the box frame size.
- int **h** () const

  gets the height
- void **h** (int H)

  sets the height
- void inset (Fl_Boxtype bt)

  Move all edges in by the frame size of box type `bt`.
- void inset (int d)

  Move all edges in by `d`.
- void inset (int left, int top, int right, int bottom)

  Move all edges in by `left`, `top`, `right`, `bottom`.
- int r () const

  gets the right edge (x + w).
- void **r** (int R)

  sets the width based on R and x
- int **w** () const

  gets the width
- void **w** (int W)

  sets the width
- int **x** () const

  gets the x coordinate (left edge)
- void **x** (int X)

  sets the x coordinate (left edge)
- int **y** () const

  gets the y coordinate (top edge)
- void **y** (int Y)

  sets the y coordinate (top edge)

Friends

- bool **operator!=** (const Fl_Rect &lhs, const Fl_Rect &rhs)
- bool **operator==** (const Fl_Rect &lhs, const Fl_Rect &rhs)

13.107.1 Detailed Description

Rectangle with standard FLTK coordinates (X, Y, W, H).
This may be used internally, for overloaded widget constructors and other overloaded methods like fl_measure(), fl_text_extents(), fl_rect(), fl_rectf(), and maybe more.

13.107.2 Constructor & Destructor Documentation

13.107.2.1 Fl_Rect()

```
Fl_Rect::Fl_Rect (
            int X,
            int Y,
            int W,
            int H,
            Fl_Boxtype bt )  [inline]
```

This constructor creates a rectangle with the given x,y coordinates and the given width and height reduced by the box frame size.
This is the same as using the constructor w/o `bt` and subsequently calling inset(`bt`).

13.107.3 Member Function Documentation

13.107.3.1 b()

```
int Fl_Rect::b ( ) const  [inline]
```

gets the bottom edge (y + h).

Note

r() and b() are coordinates **outside** the area of the rectangle.

13.107.3.2 inset() [1/3]

```
void Fl_Rect::inset (
            Fl_Boxtype bt )  [inline]
```

Move all edges in by the frame size of box type `bt`.
Shrinks the rectangle at all sides by the frame width or height of the given box type `bt`.
This method uses the frame sizes given by the box type `bt` using

- Fl::box_dx(bt)
- Fl::box_dy(bt)
- Fl::box_dw(bt)
- Fl::box_dh(bt)

If the rectangle is smaller than the frame sizes the result is undefined, i.e. an invalid or empty rectangle.

13.107.3.3 inset() [2/3]

```
void Fl_Rect::inset (
            int d )  [inline]
```

Move all edges in by `d`.
Shrinks the rectangle by `d` at all sides keeping the center of the rectangle at the same spot.
If `d` is negative, the rectangle is enlarged.
If `d` >= w() or h() the result is undefined, i.e. an invalid or empty rectangle.

13.107.3.4 inset() [3/3]

```
void Fl_Rect::inset (
            int left,
            int top,
            int right,
            int bottom )  [inline]
```

Move all edges in by `left`, `top`, `right`, `bottom`.
Shrinks the rectangle on all sides keeping the center of the rectangle at the same spot.
If any value is negative, the rectangle is enlarged.
Values are not range checked; it is possible to create an invalid or empty rectangle.

13.107.3.5 r()

```
int Fl_Rect::r ( ) const  [inline]
```

gets the right edge (x + w).

Note

r() and b() are coordinates **outside** the area of the rectangle.

The documentation for this class was generated from the following file:

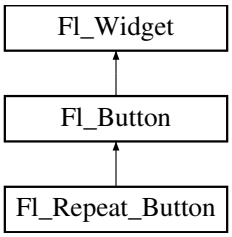
- Fl_Rect.H

13.108 Fl_Repeat_Button Class Reference

The Fl_Repeat_Button is a subclass of Fl_Button that generates a callback when it is pressed and then repeatedly generates callbacks as long as it is held down.

`#include <Fl_Repeat_Button.H>`

Inheritance diagram for Fl_Repeat_Button:



Public Member Functions

- void **deactivate** ()
- Fl_Repeat_Button (int X, int Y, int W, int H, const char *l=0)

  Creates a new Fl_Repeat_Button widget using the given position, size, and label string.
- int handle (int) override

  Handles the specified event.

Public Member Functions inherited from Fl_Button

- int clear ()

  Same as `value(0)`.
- uchar compact () const

  Return true if buttons are rendered as compact buttons.
- void compact (uchar v)

  Decide if buttons should be rendered in compact mode.
- Fl_Boxtype down_box () const

  Returns the current down box type, which is drawn when value() is non-zero.
- void down_box (Fl_Boxtype b)

  Sets the down box type.
- Fl_Color **down_color** () const

  (for backwards compatibility)
- void **down_color** (unsigned c)

  (for backwards compatibility)
- Fl_Button (int X, int Y, int W, int H, const char *L=0)

  The constructor creates the button using the given position, size, and label.
- int set ()

Same as `value(1)`.

- void **setonly** ()

  Turns on this button and turns off all other radio buttons in the group (calling `value(1)` or `set()` does not do this).
- int shortcut () const

  Returns the current shortcut key for the button.
- void **shortcut** (const char *s)

  (for backwards compatibility)
- void shortcut (int s)

  Sets the shortcut key to `s`.
- char **value** () const

  Returns the current value of the button (0 or 1).
- int value (int v)

  Sets the current value of the button.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.

- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

Gets the image that is used as part of the widget label when in the inactive state.

- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.

- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

Returns whether the widget's label uses '&' to indicate shortcuts.

- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.

- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget ∗widget, void ∗data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char ∗t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char ∗, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Button

- void draw () override

  Draws the widget.
- void **simulate_key_action** ()

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

Draws a box of type t, of color c at the position X,Y and size W,H.

- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Static Protected Member Functions inherited from Fl_Button

- static void **key_release_timeout** (void *)

Static Protected Attributes inherited from Fl_Button

- static Fl_Widget_Tracker * **key_release_tracker** = 0

13.108.1 Detailed Description

The Fl_Repeat_Button is a subclass of Fl_Button that generates a callback when it is pressed and then repeatedly generates callbacks as long as it is held down.
The speed of the repeat is fixed and depends on the implementation.

13.108.2 Constructor & Destructor Documentation

13.108.2.1 Fl_Repeat_Button()

```
Fl_Repeat_Button::Fl_Repeat_Button (
            int X,
            int Y,
            int W,
            int H,
            const char * l = 0 )
```

Creates a new Fl_Repeat_Button widget using the given position, size, and label string.
The default boxtype is FL_UP_BOX. Deletes the button.

13.108.3 Member Function Documentation

13.108.3.1 handle()

```
int Fl_Repeat_Button::handle (
            int event )  [override], [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to *override* the behavior of the base class. This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to *filter* event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you *consumed* the event by returning 1 even if you didn't *do* anything with it.

Parameters

in	*event*	the kind of event received

Return values

0	if the event was not used or understood
1	if the event was used and can be deleted

See also

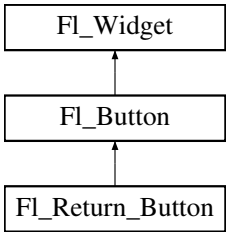
Fl_Event

Reimplemented from Fl_Button.
The documentation for this class was generated from the following files:

- Fl_Repeat_Button.H
- Fl_Repeat_Button.cxx

13.109 Fl_Return_Button Class Reference

The Fl_Return_Button is a subclass of Fl_Button that generates a callback when it is pressed or when the user presses the Enter key.
`#include <Fl_Return_Button.H>`
Inheritance diagram for Fl_Return_Button:



Public Member Functions

- Fl_Return_Button (int X, int Y, int W, int H, const char *l=0)

Creates a new Fl_Return_Button widget using the given position, size, and label string.

- int handle (int) override

  Handles the specified event.

Public Member Functions inherited from Fl_Button

- int clear ()

  Same as `value(0)`.
- uchar compact () const

  Return true if buttons are rendered as compact buttons.
- void compact (uchar v)

  Decide if buttons should be rendered in compact mode.
- Fl_Boxtype down_box () const

  Returns the current down box type, which is drawn when value() is non-zero.
- void down_box (Fl_Boxtype b)

  Sets the down box type.
- Fl_Color **down_color** () const

  (for backwards compatibility)
- void **down_color** (unsigned c)

  (for backwards compatibility)
- Fl_Button (int X, int Y, int W, int H, const char *L=0)

  The constructor creates the button using the given position, size, and label.
- int set ()

  Same as `value(1)`.
- void **setonly** ()

  Turns on this button and turns off all other radio buttons in the group (calling `value(1)` or `set()` does not do this).
- int shortcut () const

  Returns the current shortcut key for the button.
- void **shortcut** (const char *s)

  (for backwards compatibility)
- void shortcut (int s)

  Sets the shortcut key to `s`.
- char **value** () const

  Returns the current value of the button (0 or 1).
- int value (int v)

  Sets the current value of the button.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

Gets the current user data (long) argument that is passed to the callback function.

- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ **as_gl_window** () const
- virtual Fl_Group ∗ as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const ∗ **as_group** () const
- virtual Fl_Window ∗ as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ **as_window** () const
- void bind_deimage (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗cb, void ∗p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

Gets the background color of the widget.

- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.

- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

Schedules the drawing of the widget.

- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.

- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void draw () override

  Draws the widget.

Protected Member Functions inherited from Fl_Button

- void **simulate_key_action** ()

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.

- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Static Protected Member Functions inherited from Fl_Button

- static void **key_release_timeout** (void ∗)

Static Protected Attributes inherited from Fl_Button

- static Fl_Widget_Tracker ∗ **key_release_tracker** = 0

13.109.1 Detailed Description

The Fl_Return_Button is a subclass of Fl_Button that generates a callback when it is pressed or when the user presses the Enter key.
A carriage-return symbol is drawn next to the button label.

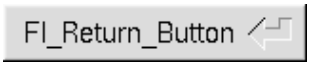


Figure 13.37 Fl_Return_Button

13.109.2 Constructor & Destructor Documentation

13.109.2.1 Fl_Return_Button()

```
Fl_Return_Button::Fl_Return_Button (
            int X,
            int Y,
            int W,
            int H,
            const char * l = 0 )
```

Creates a new Fl_Return_Button widget using the given position, size, and label string.
The default boxtype is FL_UP_BOX.
The inherited destructor deletes the button.

13.109.3 Member Function Documentation

13.109.3.1 draw()

```
void Fl_Return_Button::draw ( )  [override], [protected], [virtual]
```

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();                 // calls Fl_Scrollbar::draw()
```

Reimplemented from Fl_Button.

13.109.3.2 handle()

```
int Fl_Return_Button::handle (
            int event )  [override], [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to *override* the behavior of the base class. This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to *filter* event processing, for instance if you want to dismiss non-numeric characters (keypresses)

in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you *consumed* the event by returning 1 even if you didn't *do* anything with it.

Parameters

in	*event*	the kind of event received

Return values

0	if the event was not used or understood
1	if the event was used and can be deleted

See also

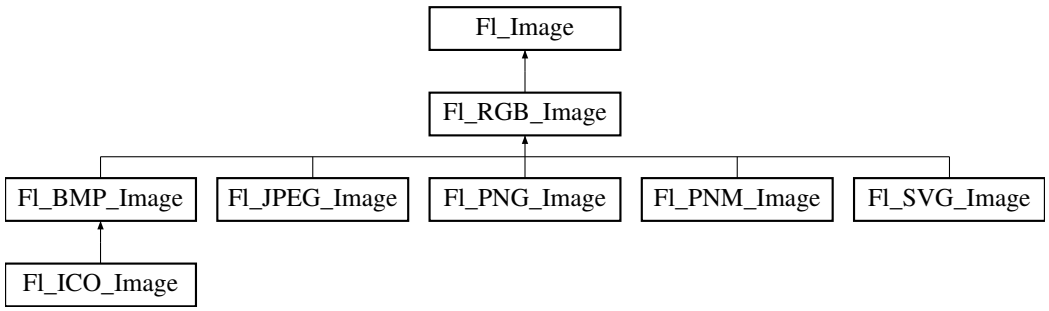
Fl_Event

Reimplemented from Fl_Button.
The documentation for this class was generated from the following files:

- Fl_Return_Button.H
- Fl_Return_Button.cxx

13.110 Fl_RGB_Image Class Reference

The Fl_RGB_Image class supports caching and drawing of full-color images with 1 to 4 channels of color information.
`#include <Fl_Image.H>`
Inheritance diagram for Fl_RGB_Image:



Public Member Functions

- virtual Fl_SVG_Image ∗ as_svg_image ()

  Returns whether an RGB image is an Fl_SVG_Image or not.
- virtual const Fl_SVG_Image ∗ as_svg_image () const

  Returns whether an RGB image is an Fl_SVG_Image or not.
- int **cache_h** () const
- int **cache_w** () const
- void color_average (Fl_Color c, float i) override

  The color_average() method averages the colors in the image with the provided FLTK color value.
- Fl_Image ∗ **copy** () const
- Fl_Image ∗ copy (int W, int H) const override

  Creates a resized copy of the image.
- void desaturate () override

The desaturate() method converts an image to grayscale.

- void **draw** (int X, int Y)
- void draw (int X, int Y, int W, int H, int cx=0, int cy=0) override

  Draws the image to the current drawing surface with a bounding box.
- Fl_RGB_Image (const Fl_Pixmap *pxm, Fl_Color bg=FL_GRAY)

  The constructor creates a new RGBA image from the specified Fl_Pixmap.
- Fl_RGB_Image (const uchar *bits, int bits_length, int W, int H, int D, int LD)

  The constructor creates a new image from the specified data.
- Fl_RGB_Image (const uchar *bits, int W, int H, int D=3, int LD=0)

  The constructor creates a new image from the specified data.
- void label (Fl_Menu_Item *m) override

  This method is an obsolete way to set the image attribute of a menu item.
- void label (Fl_Widget *w) override

  This method is an obsolete way to set the image attribute of a widget or menu item.
- virtual void normalize ()

  Makes sure the object is fully initialized.
- void uncache () override

  If the image has been cached for display, delete the cache data.
- ~**Fl_RGB_Image** () override

  The destructor frees all memory and server resources that are used by the image.

Public Member Functions inherited from Fl_Image

- virtual class Fl_Shared_Image * as_shared_image ()

  Returns whether an image is an Fl_Shared_Image or not.
- Fl_Image * copy () const

  Creates a copy of the image in the same size.
- int count () const

  Returns the number of data values associated with the image.
- int d () const

  Returns the image depth.
- const char *const * data () const

  Returns a pointer to the current image data array.
- int **data_h** () const

  Returns the height of the image data.
- int **data_w** () const

  Returns the width of the image data.
- void draw (int X, int Y)

  Draws the image to the current drawing surface.
- int fail () const

  Returns a value that is not 0 if there is currently no image available.
- Fl_Image (int W, int H, int D)

  The constructor creates an empty image with the specified width, height, and depth.
- int h () const

  Returns the current image drawing height in FLTK units.
- void inactive ()

  The inactive() method calls color_average(FL_BACKGROUND_COLOR, 0.33f) to produce an image that appears grayed out.
- int ld () const

  Returns the current line data size in bytes.
- virtual void release ()

Releases an Fl_Image - the same as '`delete this`'.

- virtual void scale (int width, int height, int proportional=1, int can_expand=0)

  Sets the drawing size of the image.
- int w () const

  Returns the current image drawing width in FLTK units.
- virtual ~**Fl_Image** ()

  The destructor is a virtual method that frees all memory used by the image.

Static Public Member Functions

- static size_t max_size ()

  Returns the maximum allowed image size in bytes when creating an Fl_RGB_Image object.
- static void max_size (size_t size)

  Sets the maximum allowed image size in bytes when creating an Fl_RGB_Image object.

Static Public Member Functions inherited from Fl_Image

- static Fl_Labeltype **define_FL_IMAGE_LABEL** ()
- static Fl_RGB_Scaling **RGB_scaling** ()

  Returns the currently used RGB image scaling method.
- static void RGB_scaling (Fl_RGB_Scaling)

  Sets the RGB image scaling method used for copy(int, int).
- static Fl_RGB_Scaling **scaling_algorithm** ()

  Gets what algorithm is used when resizing a source image to draw it.
- static void scaling_algorithm (Fl_RGB_Scaling algorithm)

  Sets what algorithm is used when resizing a source image to draw it.

Public Attributes

- int **alloc_array**

  If non-zero, the object's data array is delete[]'d when deleting the object.
- const uchar * array

  Points to the start of the object's data array.

Friends

- class **Fl_Graphics_Driver**

Additional Inherited Members

Static Public Attributes inherited from Fl_Image

- static const int **ERR_FILE_ACCESS** = -2
- static const int **ERR_FORMAT** = -3
- static const int **ERR_MEMORY_ACCESS** = -4
- static const int **ERR_NO_IMAGE** = -1
- static bool **register_images_done** = false

  True after fl_register_images() was called, false before.

Protected Member Functions inherited from Fl_Image

- void **d** (int D)

  Sets the current image depth.
- void data (const char ∗const ∗p, int c)

  Sets the current data pointer and count of pointers in the array.
- void draw_empty (int X, int Y)

  The protected method draw_empty() draws a box with an X in it.
- int draw_scaled (int X, int Y, int W, int H)

  Draw the image to the current drawing surface rescaled to a given width and height.
- void h (int H)

  Sets the height of the image data.
- void ld (int LD)

  Sets the current line data size in bytes.
- void w (int W)

  Sets the width of the image data.

Static Protected Member Functions inherited from Fl_Image

- static void **labeltype** (const Fl_Label ∗lo, int lx, int ly, int lw, int lh, Fl_Align la)
- static void **measure** (const Fl_Label ∗lo, int &lw, int &lh)

13.110.1 Detailed Description

The Fl_RGB_Image class supports caching and drawing of full-color images with 1 to 4 channels of color information.
Images with an even number of channels are assumed to contain alpha information, which is used to blend the image with the contents of the screen. Functions fl_write_png(const char ∗filename, Fl_RGB_Image ∗img) and fl_write_jpeg(const char ∗filename, Fl_RGB_Image ∗img) allow to save any Fl_RGB_Image object to a PNG or JPEG file, respectively. Both require linking to the `fltk_images` library.
Fl_RGB_Image is defined in <FL/Fl_Image.H>, however for compatibility reasons <FL/Fl_RGB_Image.H> should be included.

13.110.2 Constructor & Destructor Documentation

13.110.2.1 Fl_RGB_Image() [1/3]

```
Fl_RGB_Image::Fl_RGB_Image (
            const uchar * bits,
            int W,
            int H,
            int D = 3,
            int LD = 0 )
```

The constructor creates a new image from the specified data.
The data array `bits` must contain sufficient data to provide `W ∗ H ∗ D` image bytes and optional line padding, see `LD`.
`W` and `H` are the width and height of the image in pixels, resp.
`D` is the image depth and can be:

- D=1: each uchar in `bits`[] is a grayscale pixel value
- D=2: each uchar pair in `bits`[] is a grayscale + alpha pixel value
- D=3: each uchar triplet in `bits`[] is an R/G/B pixel value
- D=4: each uchar quad in `bits`[] is an R/G/B/A pixel value

LD specifies the line data size of the array, see Fl_Image::ld(int). If LD is zero, then W * D is assumed, otherwise LD must be greater than or equal to W * D to account for (unused) extra data per line (padding).
The caller is responsible that the image data array bits persists as long as the image is used.
This constructor sets Fl_RGB_Image::alloc_array to 0. To have the image object control the deallocation of the data array bits, set alloc_array to non-zero after construction.

Parameters

in	*bits*	The image data array.
in	*W*	The width of the image in pixels.
in	*H*	The height of the image in pixels.
in	*D*	The image depth, or 'number of channels' (default=3).
in	*LD*	Line data size (default=0).

See also

Fl_Image::data(), Fl_Image::w(), Fl_Image::h(), Fl_Image::d(), Fl_Image::ld(int)

13.110.2.2 Fl_RGB_Image() [2/3]

```
Fl_RGB_Image::Fl_RGB_Image (
            const uchar * bits,
            int bits_length,
            int W,
            int H,
            int D,
            int LD )
```

The constructor creates a new image from the specified data.
If the provided array is too small to contain all the image data, the constructor will not generate the image to avoid illegal memory read access and instead set data to NULL and ld to ERR_MEMORY_ACCESS.

Parameters

bits	image data
bits_length	length of the bits array in bytes
W	image width in pixels
H	image height in pixels
D	image depth in bytes, 1 for gray scale, 2 for gray with alpha, 3 for RGB, and 4 for RGB plus alpha
LD	line length in bytes, or 0 to use W∗D.

See also

Fl_RGB_Image(const uchar ∗bits, int W, int H, int D, int LD)

13.110.2.3 Fl_RGB_Image() [3/3]

```
Fl_RGB_Image::Fl_RGB_Image (
            const Fl_Pixmap * pxm,
            Fl_Color bg = FL_GRAY )
```

The constructor creates a new RGBA image from the specified Fl_Pixmap.
The RGBA image is built fully opaque except for the transparent area of the pixmap that is assigned the bg color with full transparency.
This constructor creates a new internal data array and sets Fl_RGB_Image::alloc_array to 1 so the data array is deleted when the image is destroyed.

13.110.3 Member Function Documentation

13.110.3.1 as_svg_image() [1/2]

```
virtual Fl_SVG_Image * Fl_RGB_Image::as_svg_image ( ) [inline], [virtual]
```

Returns whether an RGB image is an Fl_SVG_Image or not.
This virtual method returns a pointer to the Fl_SVG_Image if this object is an instance of Fl_SVG_Image or NULL if not.
Reimplemented in Fl_SVG_Image.

13.110.3.2 as_svg_image() [2/2]

```
virtual const Fl_SVG_Image * Fl_RGB_Image::as_svg_image ( ) const [inline], [virtual]
```

Returns whether an RGB image is an Fl_SVG_Image or not.
This virtual method returns a pointer to the Fl_SVG_Image if this object is an instance of const Fl_SVG_Image or NULL if not.
Reimplemented in Fl_SVG_Image.

13.110.3.3 color_average()

```
void Fl_RGB_Image::color_average (
            Fl_Color c,
            float i ) [override], [virtual]
```

The color_average() method averages the colors in the image with the provided FLTK color value.
The first argument specifies the FLTK color to be used.
The second argument specifies the amount of the original image to combine with the color, so a value of 1.0 results in no color blend, and a value of 0.0 results in a constant image of the specified color.
An internal copy is made of the original image data before changes are applied, to avoid modifying the original image data in memory.
Reimplemented from Fl_Image.
Reimplemented in Fl_SVG_Image.

13.110.3.4 copy()

```
Fl_Image * Fl_RGB_Image::copy (
            int W,
            int H ) const [override], [virtual]
```

Creates a resized copy of the image.
It is recommended not to call this member function to reduce the size of an image to the size of the area where this image will be drawn, and to use Fl_Image::scale() instead.
The new image should be released when you are done with it.
Note: since FLTK 1.4.0 you can use Fl_Image::release() for all types of images (i.e. all subclasses of Fl_Image) instead of operator *delete* for Fl_Image's and Fl_Image::release() for Fl_Shared_Image's.
The new image data will be converted to the requested size. RGB images are resized using the algorithm set by Fl_Image::RGB_scaling().
For the new image the following equations are true:

- w() == data_w() == W
- h() == data_h() == H

Parameters

in	*W,H*	Requested width and height of the new image

Note

The returned image can be safely cast to the same image type as that of the source image provided this type is one of Fl_RGB_Image, Fl_SVG_Image, Fl_Pixmap, Fl_Bitmap, Fl_Tiled_Image, Fl_Anim_GIF_Image and Fl_Shared_Image. Returned objects copied from images of other, derived, image classes belong to the parent class appearing in this list. For example, the copy of an Fl_GIF_Image is an object of class Fl_Pixmap.

Since FLTK 1.4.0 this method is 'const'. If you derive your own class from Fl_Image or any subclass your overridden methods of 'Fl_Image::copy() const' and 'Fl_Image::copy(int, int) const' **must** also be 'const' for inheritance to work properly. This is different than in FLTK 1.3.x and earlier where these methods have not been 'const'.

Reimplemented from Fl_Image.
Reimplemented in Fl_SVG_Image.

13.110.3.5 desaturate()

```
void Fl_RGB_Image::desaturate ( ) [override], [virtual]
```

The desaturate() method converts an image to grayscale.
If the image contains an alpha channel (depth = 4), the alpha channel is preserved.
An internal copy is made of the original image data before changes are applied, to avoid modifying the original image data in memory.
Reimplemented from Fl_Image.
Reimplemented in Fl_SVG_Image.

13.110.3.6 draw()

```
void Fl_RGB_Image::draw (
            int X,
            int Y,
            int W,
            int H,
            int cx = 0,
            int cy = 0 ) [override], [virtual]
```

Draws the image to the current drawing surface with a bounding box.
Arguments `X,Y,W,H` specify a bounding box for the image, with the origin (upper-left corner) of the image offset by the `cx` and `cy` arguments.
Another way to see what part of the image gets drawn and where, is to consider this alternative writing producing the same output:

```
fl_push_clip(X,Y,W,H);
this->draw(X - cx, Y - cy);
fl_pop_clip();
```

Repeated calls to this member function with the same image but varying `W, H, cx`, or `cy` arguments can be more efficiently processed using the above alternative writing.
Reimplemented from Fl_Image.
Reimplemented in Fl_SVG_Image.

13.110.3.7 label() [1/2]

```
void Fl_RGB_Image::label (
            Fl_Menu_Item * m ) [override], [virtual]
```

This method is an obsolete way to set the image attribute of a menu item.

Deprecated Please use Fl_Menu_Item::image() instead.

Reimplemented from Fl_Image.

13.110.3.8 label() [2/2]

```
void Fl_RGB_Image::label (
            Fl_Widget * widget ) [override], [virtual]
```

This method is an obsolete way to set the image attribute of a widget or menu item.

Deprecated Please use Fl_Widget::image() or Fl_Widget::deimage() instead.

Reimplemented from Fl_Image.

13.110.3.9 max_size() [1/2]

```
static size_t Fl_RGB_Image::max_size ( ) [inline], [static]
```

Returns the maximum allowed image size in bytes when creating an Fl_RGB_Image object.

See also

void Fl_RGB_Image::max_size(size_t)

13.110.3.10 max_size() [2/2]

```
static void Fl_RGB_Image::max_size (
            size_t size ) [inline], [static]
```

Sets the maximum allowed image size in bytes when creating an Fl_RGB_Image object.
The image size in bytes of an Fl_RGB_Image object is the value of the product w() ∗ h() ∗ d(). If this product exceeds size, the created object of a derived class of Fl_RGB_Image won't be loaded with the image data. This does not apply to direct RGB image creation with Fl_RGB_Image::Fl_RGB_Image(const uchar ∗bits, int W, int H, int D, int LD).
The default max_size() value is essentially infinite.

13.110.3.11 normalize()

```
virtual void Fl_RGB_Image::normalize ( ) [inline], [virtual]
```

Makes sure the object is fully initialized.
In particular, makes sure member variable array is non-null.
Reimplemented in Fl_SVG_Image.

13.110.3.12 uncache()

```
void Fl_RGB_Image::uncache ( ) [override], [virtual]
```

If the image has been cached for display, delete the cache data.
This allows you to change the data used for the image and then redraw it without recreating an image object.
Reimplemented from Fl_Image.

13.110.4 Member Data Documentation

13.110.4.1 array

```
const uchar* Fl_RGB_Image::array
```

Points to the start of the object's data array.

See also

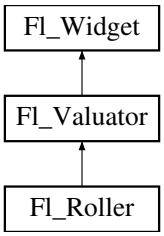
class Fl_SVG_Image which delays initialization of this member variable.

The documentation for this class was generated from the following files:

- Fl_Image.H
- Fl_Image.cxx

13.111 Fl_Roller Class Reference

The Fl_Roller widget is a "dolly" control commonly used to move 3D objects.
`#include <Fl_Roller.H>`
Inheritance diagram for Fl_Roller:



Public Member Functions

- Fl_Roller (int X, int Y, int W, int H, const char ∗L=0)

  Creates a new Fl_Roller widget using the given position, size, and label string.
- int handle (int) override

  Handles the specified event.

Public Member Functions inherited from Fl_Valuator

- void **bounds** (double a, double b)

  Sets the minimum (a) and maximum (b) values for the valuator widget.
- double **clamp** (double)

  Clamps the passed value to the valuator range.
- virtual int format (char ∗)

  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.
- virtual std::string format_str ()

  C++11 API for Fl_Valuator::format(char∗ buffer).
- double increment (double, int)

  Adds n times the step value to the passed value.
- double maximum () const

  Gets the maximum value for the valuator.
- void maximum (double a)

  Sets the maximum value for the valuator.
- double minimum () const

  Gets the minimum value for the valuator.
- void minimum (double a)

  Sets the minimum value for the valuator.
- void precision (int digits)

  Sets the step value to 1.0 / 10^{digits}.
- void range (double a, double b)

  Sets the minimum and maximum values for the valuator.
- double round (double)

  Round the passed value to the nearest step increment.
- double step () const

  Gets or sets the step value.
- void **step** (double a, int b)

  See double Fl_Valuator::step() const
- void **step** (double s)

  See double Fl_Valuator::step() const.
- void **step** (int a)

  See double Fl_Valuator::step() const
- double value () const

Gets the floating point(double) value.

- int value (double)

  Sets the current value.

- ~**Fl_Valuator** () override

  Destructor is accessible despite protected constructor.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.

- unsigned int active () const

  Returns whether the widget is active.

- int active_r () const

  Returns whether the widget and all of its parents are active.

- Fl_Align align () const

  Gets the label alignment.

- void align (Fl_Align alignment)

  Sets the label alignment.

- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.

- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.

- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.

- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.

- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.

- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.

- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.

- Fl_Boxtype box () const

  Gets the box type of the widget.

- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.

- Fl_Callback_p callback () const

  Gets the current callback function for the widget.

- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.

- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.

- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

Returns whether the inactive image is managed by the widget.

- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.

- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

Gives the widget the keyboard focus.

- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void draw () override

  Draws the widget.

Protected Member Functions inherited from Fl_Valuator

- Fl_Valuator (int X, int Y, int W, int H, const char ∗L)

  Creates a new Fl_Valuator widget using the given position, size, and label string.
- void **handle_drag** (double newvalue)

  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.
- void **handle_push** ()

  Stores the current value in the previous value.
- void **handle_release** ()

  Called after an FL_WHEN_RELEASE event is received and before the callback.
- int **horizontal** () const

  Tells if the valuator is an FL_HORIZONTAL one.
- double **previous_value** () const

  Gets the previous floating point value before an event changed it.
- void **set_value** (double v)

  Sets the current floating point value.
- double **softclamp** (double)

  Clamps the value, but accepts v if the previous value is not already out of range.
- virtual void value_damage ()

  Asks for partial redraw.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.

- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.111.1 Detailed Description

The Fl_Roller widget is a "dolly" control commonly used to move 3D objects.
The roller can be controlled by clicking and dragging the mouse, by the corresponding arrow keys when the roller has the keyboard focus, or by the mouse wheels when the mouse pointer is positioned over the roller widget.

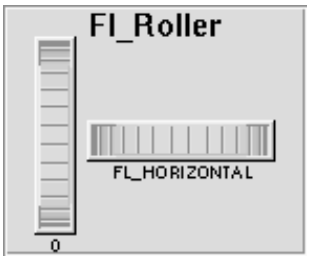


Figure 13.38 Fl_Roller

13.111.2 Constructor & Destructor Documentation

13.111.2.1 Fl_Roller()

```
Fl_Roller::Fl_Roller (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

Creates a new Fl_Roller widget using the given position, size, and label string.
The default boxtype is FL_NO_BOX.
Inherited destructor destroys the valuator.

13.111.3 Member Function Documentation

13.111.3.1 draw()

```
void Fl_Roller::draw ( )  [override], [protected], [virtual]
```

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();                 // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

13.111.3.2 handle()

```
int Fl_Roller::handle (
            int event )  [override], [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to *override* the behavior of the base class. This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to *filter* event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you *consumed* the event by returning 1 even if you didn't *do* anything with it.

Parameters

in	*event*	the kind of event received

Return values

0	if the event was not used or understood
1	if the event was used and can be deleted

See also

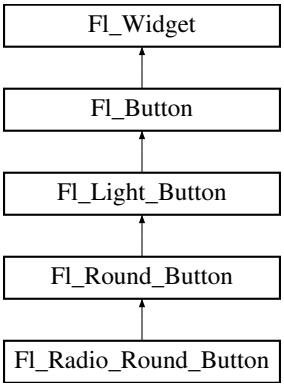
Fl_Event

Reimplemented from Fl_Widget.
The documentation for this class was generated from the following files:

- Fl_Roller.H
- Fl_Roller.cxx

13.112 Fl_Round_Button Class Reference

Buttons generate callbacks when they are clicked by the user.
`#include <Fl_Round_Button.H>`
Inheritance diagram for Fl_Round_Button:



Public Member Functions

- Fl_Round_Button (int x, int y, int w, int h, const char *l=0)

  Creates a new Fl_Round_Button widget using the given position, size, and label string.

Public Member Functions inherited from Fl_Light_Button

- Fl_Light_Button (int x, int y, int w, int h, const char *l=0)

  Creates a new Fl_Light_Button widget using the given position, size, and label string.
- int handle (int) override

  Handles the specified event.

Public Member Functions inherited from Fl_Button

- int clear ()

  Same as `value(0)`.
- uchar compact () const

  Return true if buttons are rendered as compact buttons.
- void compact (uchar v)

  Decide if buttons should be rendered in compact mode.
- Fl_Boxtype down_box () const

  Returns the current down box type, which is drawn when value() is non-zero.
- void down_box (Fl_Boxtype b)

  Sets the down box type.
- Fl_Color **down_color** () const

  (for backwards compatibility)

- void **down_color** (unsigned c)

  (for backwards compatibility)
- Fl_Button (int X, int Y, int W, int H, const char *L=0)

  The constructor creates the button using the given position, size, and label.
- int set ()

  Same as `value(1)`.
- void **setonly** ()

  Turns on this button and turns off all other radio buttons in the group (calling `value(1)` *or* `set()` *does not do this).*
- int shortcut () const

  Returns the current shortcut key for the button.
- void **shortcut** (const char *s)

  (for backwards compatibility)
- void shortcut (int s)

  Sets the shortcut key to `s`.
- char **value** () const

  Returns the current value of the button (0 or 1).
- int value (int v)

  Sets the current value of the button.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

Sets the image to use as part of the widget label when in the active state.

- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.

- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

Return the gap size between the label and the image.

- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.

- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

Gets the widget width.

- Fl_When when () const

  Returns the conditions under which the callback is called.

- void when (uchar i)

  Sets the flags used to decide when a callback is called.

- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.

- int x () const

  Gets the widget position in its window.

- int y () const

  Gets the widget position in its window.

- virtual ~Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.

- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.

- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Light_Button

- void draw () override

  Draws the widget.

Protected Member Functions inherited from Fl_Button

- void **simulate_key_action** ()

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Static Protected Member Functions inherited from Fl_Button

- static void **key_release_timeout** (void *)

Static Protected Attributes inherited from Fl_Button

- static Fl_Widget_Tracker * **key_release_tracker** = 0

13.112.1 Detailed Description

Buttons generate callbacks when they are clicked by the user.
You control exactly when and how by changing the values for type() and when().

Figure 13.39 Fl_Round_Button

The Fl_Round_Button subclass displays the "on" state by turning on a light, rather than drawing pushed in. The shape of the "light" is initially set to FL_ROUND_DOWN_BOX. The color of the light when on is controlled with selection_color(), which defaults to FL_FOREGROUND_COLOR.

13.112.2 Constructor & Destructor Documentation

13.112.2.1 Fl_Round_Button()

```
Fl_Round_Button::Fl_Round_Button (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

Creates a new Fl_Round_Button widget using the given position, size, and label string.

Figure 13.40 Fl_Round_Button

The Fl_Round_Button subclass displays the "ON" state by turning on a light, rather than drawing pushed in.
The default box type is FL_NO_BOX, which draws the label w/o a box right of the checkmark.
The shape of the "light" is set with down_box() and its default value is FL_ROUND_DOWN_BOX.
The color of the light when on is controlled with selection_color(), which defaults to FL_FOREGROUND_COLOR (usually black).

Parameters

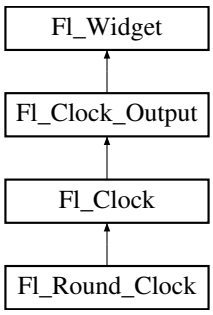
in	*X,Y,W,H*	position and size of the widget
in	*L*	widget label, default is no label

The documentation for this class was generated from the following files:

- Fl_Round_Button.H
- Fl_Round_Button.cxx

13.113 Fl_Round_Clock Class Reference

A clock widget of type FL_ROUND_CLOCK.
`#include <Fl_Round_Clock.H>`
Inheritance diagram for Fl_Round_Clock:



Public Member Functions

- Fl_Round_Clock (int X, int Y, int W, int H, const char *L=0)

  Creates the clock widget, setting his type and box.

Public Member Functions inherited from Fl_Clock

- Fl_Clock (int X, int Y, int W, int H, const char ∗L=0)

  Create an Fl_Clock widget using the given position, size, and label string.
- Fl_Clock (uchar t, int X, int Y, int W, int H, const char ∗L)

  Create an Fl_Clock widget using the given clock type `t`, position, size, and label string.
- int handle (int) override

  Handles the specified event.
- ∼**Fl_Clock** ()

  The destructor removes the clock.

Public Member Functions inherited from Fl_Clock_Output

- Fl_Clock_Output (int X, int Y, int W, int H, const char ∗L=0)

  Create a new Fl_Clock_Output widget with the given position, size and label.
- int hour () const

  Returns the displayed hour (0 to 23).
- int minute () const

  Returns the displayed minute (0 to 59).
- int second () const

  Returns the displayed second (0 to 60, 60=leap second).
- int shadow () const

  Returns the shadow drawing mode of the hands.
- void shadow (int mode)

  Sets the shadow drawing mode of the hands.
- ulong value () const

  Returns the displayed time.
- void value (int H, int m, int s)

  Set the displayed time.
- void value (ulong v)

  Set the displayed time.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

- virtual class Fl_Gl_Window const ∗ **as_gl_window** () const
- virtual Fl_Group ∗ as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const ∗ **as_group** () const
- virtual Fl_Window ∗ as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ **as_window** () const
- void bind_deimage (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗cb, void ∗p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.

- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

Returns whether the image is managed by the widget.

- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.

- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

Get the spacing between the label and the vertical edge of the widget.

- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Clock_Output

- void draw () override

  Draw clock with current position and size.
- void draw (int X, int Y, int W, int H)

  Draw clock with the given position and size.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

13.113.1 Detailed Description

A clock widget of type FL_ROUND_CLOCK.
Has no box.

13.113.2 Constructor & Destructor Documentation

13.113.2.1 Fl_Round_Clock()

```
Fl_Round_Clock::Fl_Round_Clock (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

Creates the clock widget, setting his type and box.
Create an Fl_Round_Clock widget using the given position, size, and label string.
The clock type is `FL_ROUND_CLOCK` and the boxtype is `FL_NO_BOX`.
This construcktor is the same as Fl_Clock(FL_ROUND_CLOCK, X, Y, W, H, L).

See also

Fl_Clock(uchar, int, int, int, int, const char *)

Parameters

in	*X,Y,W,H*	position and size of the widget
in	*L*	widget label, default is no label

The documentation for this class was generated from the following files:

- Fl_Round_Clock.H
- Fl_Clock.cxx

13.114 Fl_Scheme Class Reference

Base class with static methods for future Fl_Scheme classes.
`#include <Fl_Scheme.H>`

Static Public Member Functions

- static int add_scheme_name (const char *name)

  Add a scheme name to the list of known schemes.
- static const char ** names ()

  Return a list of all known scheme names.
- static int num_schemes ()

  Return the number of currently registered schemes.
- static void plastic_color_average (int av)

  Set the color average value of the 'plastic' scheme in percent.

13.114.1 Detailed Description

Base class with static methods for future Fl_Scheme classes.
This class is intentionally not fully documented and is subject to change in future FLTK versions.

Note

Do not rely on details of this class.

Since

1.4.0

13.114.2 Member Function Documentation

13.114.2.1 add_scheme_name()

```
int Fl_Scheme::add_scheme_name (
            const char * name )  [static]
```

Add a scheme name to the list of known schemes.

This method is public since FLTK 1.4.0 because derived classes of Fl_Scheme are not yet implemented. Thus, users implementing their own schemes can use this method to add the scheme name to the list of known schemes which is for instance used in Fl_Scheme::names().

Note

Attention! In a future version, when subclasses of Fl_Scheme will be implemented, this method will either be replaced by another `protected` method or it will no longer do anything (kept only for backwards compatibility).

The new scheme name must consist of valid ASCII characters as described below:

- lowercase letters `'a'-'z'`
- numbers `'0'-'9'`
- any character in `"$+_."` (w/o the quotes).

The name must not be longer than 12 ASCII characters (bytes). The new scheme name is added to the **end** of the **unordered** list.

Note

Call this method only once for each scheme name. If the returned value is <= 0 you should check the scheme name.

The given scheme `name` is copied and may be freed directly after the call to add_scheme_name().

Parameters

in	*name*	New scheme name

Returns

The new number of schemes if the name was successfully added. This is the same as the index of the scheme + 1.

Return values

0	Scheme `name` already exists
-1	Invalid character(s) in `name`
-2	The `name` is too long

Since

1.4.0

13.114.2.2 names()

```
const char ** Fl_Scheme::names ( )  [static]
```

Return a list of all known scheme names.

This list is only valid until a new scheme is added or one is removed. It is possible that scheme names are appended to the list during the runtime of the program but schemes can't be removed.

Getting the list of known schemes can be useful to populate a menu of scheme choices to let the user select a scheme. You should process the names immediately and you should never store a pointer to the list or any individual name for later reference because the location of the list can be changed (reallocated) when schemes are added.
The list of scheme names is nul-terminated.

Note

Currently (since FLTK 1.4.0) schemes can only be added and can't be removed from the list. This may change in a later version.

Returns

List of currently known scheme names.

13.114.2.3 num_schemes()

```
static int Fl_Scheme::num_schemes ( ) [inline], [static]
```

Return the number of currently registered schemes.

Returns

Number of registered schemes.

13.114.2.4 plastic_color_average()

```
void Fl_Scheme::plastic_color_average (
            int av ) [static]
```

Set the color average value of the 'plastic' scheme in percent.
Legal values are in the range [10 ... 100], other values are clamped. The default value is 75 which is backwards compatible with FLTK 1.4 and earlier.
Higher values make the colors in boxes etc. appear "more gray" whereas lower values make colors appear more like the original color. The recommended value is about 40 to 60 but this is left to the user.
If this method is not called then the environment variable `FLTK_PLASTIC_AVERAGE` can be used to set the color average. The environment variable must be a pure numeric (integer) value in the given range, otherwise the behavior is undefined.
However, calling **this** method takes precedence over the environment variable. This method can be called at any time (e.g. to view dynamic changes). This will permanently change the appearance for all later box / background drawings.
For details see GitHub Issue 464: "RFE: plastic scheme with faithful colors".

Note

Program developers are supposed to use this method to apply "better" values than the default (1.↩ 4) look and feel. End users are supposed to use the environment variable for programs that **don't** use Fl_Scheme::plastic_color_average().

Include the following header:
`#include <FL/Fl_Scheme.H>`

Parameters

in	*av*	color average value in the documented range.

Since

1.5.0

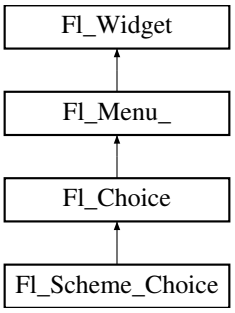
The documentation for this class was generated from the following files:

- Fl_Scheme.H

- fl_plastic.cxx
- Fl_Scheme.cxx

13.115 Fl_Scheme_Choice Class Reference

Inheritance diagram for Fl_Scheme_Choice:



Public Member Functions

- Fl_Scheme_Choice (int X, int Y, int W, int H, const char ∗L=0)

  The constructor initializes the Fl_Scheme_Choice object with all known schemes.
- int handle (int event) override

  Handle FLTK events.
- virtual void init_value ()

  Public method to initialize the value of the Fl_Scheme_Choice widget.

Public Member Functions inherited from Fl_Choice

- Fl_Choice (int X, int Y, int W, int H, const char ∗L=0)

  Create a new Fl_Choice widget using the given position, size and label string.
- int value () const

  Gets the index of the last item chosen by the user.
- int value (const Fl_Menu_Item ∗v)

  Sets the currently selected value using a pointer to menu item.
- int value (int v)

  Sets the currently selected value using the index into the menu item array.

Public Member Functions inherited from Fl_Menu_

- int add (const char ∗)

  This is a Forms (and SGI GL library) compatible add function, it adds many menu items, with '|' separating the menu items, and tab separating the menu item names from an optional shortcut string.
- int add (const char ∗, int shortcut, Fl_Callback ∗, void ∗=0, int=0)

  Adds a new menu item.
- int **add** (const char ∗a, const char ∗b, Fl_Callback ∗c, void ∗d=0, int e=0)

  See int Fl_Menu_::add(const char∗ label, int shortcut, Fl_Callback∗, void ∗user_data=0, int flags=0)
- void clear ()

  Same as menu(NULL), set the array pointer to null, indicating a zero-length menu.
- int clear_submenu (int index)

  Clears the specified submenu pointed to by `index` of all menu items.
- void copy (const Fl_Menu_Item ∗m, void ∗user_data=0)

  Sets the menu array pointer with a copy of m that will be automatically deleted.

- Fl_Boxtype **down_box** () const

  This box type is used to surround the currently-selected items in the menus.
- void down_box (Fl_Boxtype b)

  Sets the box type used to surround the currently-selected items in the menus.
- Fl_Color **down_color** () const

  For back compatibility, same as selection_color()
- void **down_color** (unsigned c)

  For back compatibility, same as selection_color()
- int find_index (const char *name) const

  Find the menu item index for a given menu `pathname`, such as "Edit/Copy".
- int find_index (const Fl_Menu_Item *item) const

  Find the index into the menu array for a given `item`.
- int find_index (Fl_Callback *cb) const

  Find the index into the menu array for a given callback `cb`.
- const Fl_Menu_Item * find_item (const char *name)

  Find the menu item for a given menu `pathname`, such as "Edit/Copy".
- const Fl_Menu_Item * find_item (Fl_Callback *)

  Find the menu item for the given callback `cb`.
- const Fl_Menu_Item * find_item_with_argument (long)

  Find the menu item for the given user argument `v`.
- const Fl_Menu_Item * find_item_with_user_data (void *)

  Find the menu item for the given user data `v`.
- Fl_Menu_ (int, int, int, int, const char *=0)

  Creates a new Fl_Menu_ widget using the given position, size, and label string.
- void global ()

  Make the shortcuts for this menu work no matter what window has the focus when you type it.
- int insert (int index, const char *, int shortcut, Fl_Callback *, void *=0, int=0)

  Inserts a new menu item at the specified `index` position.
- int **insert** (int index, const char *a, const char *b, Fl_Callback *c, void *d=0, int e=0)

  See int Fl_Menu_::insert(const char label, int shortcut, Fl_Callback*, void *user_data=0, int flags=0)*
- int item_pathname (char *name, int namelen, const Fl_Menu_Item *finditem=0) const

  Get the menu 'pathname' for the specified menuitem.
- const Fl_Menu_Item * menu () const

  Returns a pointer to the array of Fl_Menu_Items.
- void menu (const Fl_Menu_Item *m)

  Sets the menu array pointer directly.
- Fl_Boxtype menu_box () const

  Get the box type for the menu popup windows.
- void menu_box (Fl_Boxtype b)

  Set the box type for the menu popup windows.
- const Fl_Menu_Item * menu_end ()

  Finishes menu modifications and returns menu().
- int mode (int i) const

  Get the flags of item i.
- void mode (int i, int fl)

  Set the flags of item i.
- const Fl_Menu_Item * mvalue () const

  Return a pointer to the last menu item that was picked.
- const Fl_Menu_Item * picked (const Fl_Menu_Item *)

  When user picks a menu item, call this.
- const Fl_Menu_Item * prev_mvalue () const

Return a pointer to the menu item that was picked before the current one was picked.

- void remove (int)

  Deletes item `i` from the menu.
- void replace (int, const char *)

  Changes the text of item `i`.
- void **setonly** (Fl_Menu_Item *item)

  Turns the radio item "on" for the menu item and turns "off" adjacent radio items of the same group.
- void **shortcut** (int i, int s)

  Change the shortcut of item `i` to `s`.
- int size () const

  This returns the number of Fl_Menu_Item structures that make up the menu, correctly counting submenus.
- void **size** (int W, int H)
- const Fl_Menu_Item * test_shortcut ()

  Returns the menu item with the entered shortcut (key value).
- const char * text () const

  Returns the title of the last item chosen.
- const char * text (int i) const

  Returns the title of item i.
- Fl_Color textcolor () const

  Get the current color of menu item labels.
- void **textcolor** (Fl_Color c)

  Sets the current color of menu item labels.
- Fl_Font textfont () const

  Gets the current font of menu item labels.
- void textfont (Fl_Font c)

  Sets the current font of menu item labels.
- Fl_Fontsize textsize () const

  Gets the font size of menu item labels.
- void textsize (Fl_Fontsize c)

  Sets the font size of menu item labels.
- int value () const

  Return the index into the menu() of the last item chosen by the user.
- int value (const Fl_Menu_Item *)

  Set the value of a menu to the menu item `m`.
- int value (int i)

  Set the value of the menu to index `i`.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.

- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.

- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

Gets the image that is used as part of the widget label when in the active state.

- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.

- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

Gets the user data for this widget.

- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Static Protected Member Functions

- static void scheme_cb_ (Fl_Widget *w, void *)

  Internal Fl_Scheme_Choice callback function (protected).

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 , AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Choice

- void draw () override

  Draws the widget.

Protected Member Functions inherited from Fl_Menu_

- int **item_pathname_** (char *name, int namelen, const Fl_Menu_Item *finditem, const Fl_Menu_Item *menu=0) const

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.

- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Protected Attributes inherited from Fl_Menu_

- uchar **alloc**
- uchar **down_box_**
- Fl_Boxtype **menu_box_**
- Fl_Color **textcolor_**
- Fl_Font **textfont_**
- Fl_Fontsize **textsize_**

13.115.1 Constructor & Destructor Documentation

13.115.1.1 Fl_Scheme_Choice()

```
Fl_Scheme_Choice::Fl_Scheme_Choice (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

The constructor initializes the Fl_Scheme_Choice object with all known schemes.

Parameters

in	*X,Y*	Widget coordinates
in	*W,H*	Widget size (width, height)
in	*L*	Widget label (default: NULL, no label)

13.115.2 Member Function Documentation

13.115.2.1 handle()

```
int Fl_Scheme_Choice::handle (
            int event )  [override], [virtual]
```

Handle FLTK events.

This widget uses FL_SHOW and some other events to initialize its value() according to the current scheme.

All events are also handled by the base class Fl_Choice.

Parameters

in	*event*	

Returns

1 if the event was used, 0 otherwise

Reimplemented from Fl_Choice.

13.115.2.2 init_value()

```
void Fl_Scheme_Choice::init_value ( ) [virtual]
```

Public method to initialize the value of the Fl_Scheme_Choice widget.
Normally you don't need to call this unless you change the current scheme by calling Fl::scheme(const char *).
The Fl_Scheme_Choice widget does this automatically when the widget is shown (when receiving the FL_SHOW event) which should always be after Fl_Window::show(argc, argv) which may set the current scheme by interpreting the commandline.

Since

1.4.0

13.115.2.3 scheme_cb_()

```
void Fl_Scheme_Choice::scheme_cb_ (
            Fl_Widget * w,
            void * ) [static], [protected]
```

Internal Fl_Scheme_Choice callback function (protected).
You don't need to set a callback for this widget. The default callback changes the scheme (Fl::scheme()) and redraws all open windows.
You may override the callback if changing the scheme shall redraw other windows or don't redraw the window at all.

Parameters

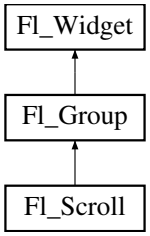
in	*w*	The Fl_Scheme_Choice widget

The documentation for this class was generated from the following files:

- Fl_Scheme_Choice.H
- Fl_Scheme_Choice.cxx

13.116 Fl_Scroll Class Reference

This container widget lets you maneuver around a set of widgets much larger than your window.
`#include <Fl_Scroll.H>`
Inheritance diagram for Fl_Scroll:



Classes

- struct Scrollbar_Data

  A local struct to manage a scrollbar's xywh region and tab values.
- struct ScrollInfo

  Structure to manage scrollbar and widget interior sizes.

Public Types

- enum {
  HORIZONTAL = 1 , **VERTICAL** = 2 , **BOTH** = 3 , **ALWAYS_ON** = 4 ,
  HORIZONTAL_ALWAYS = 5 , **VERTICAL_ALWAYS** = 6 , **BOTH_ALWAYS** = 7 }

Public Member Functions

- void **clear** ()

  Clear all but the scrollbars...
- int delete_child (int n) override

  Removes the widget at `index` from the group and deletes it.
- Fl_Scroll (int X, int Y, int W, int H, const char ∗L=0)

  Creates a new Fl_Scroll widget using the given position, size, and label string.
- int handle (int) override

  Handles the specified event.
- void resize (int X, int Y, int W, int H) override

  Resizes the Fl_Scroll widget and moves its children if necessary.
- void scroll_to (int, int)

  Moves the contents of the scroll group to a new position.
- int scrollbar_size () const

  Gets the current size of the scrollbars' troughs, in pixels.
- void scrollbar_size (int newSize)

  Sets the pixel size of the scrollbars' troughs to `newSize`, in pixels.
- int xposition () const

  Gets the current horizontal scrolling position.
- int yposition () const

  Gets the current vertical scrolling position.
- virtual ∼Fl_Scroll ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Group

- Fl_Widget ∗& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget ∗o)

  See void Fl_Group::add(Fl_Widget &w)
- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget ∗const ∗ array () const

  Returns a pointer to the internal array of children.
- Fl_Group const ∗ as_group () const override
- Fl_Group ∗ as_group () override

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget ∗ child (int n) const

  Returns the n'th child.
- int **children** () const

  Returns how many child widgets the group has.
- void clear ()

Deletes all child widgets from memory recursively.

- unsigned int clip_children () const

  Returns the current clipping mode.
- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- void end ()

  Exactly the same as current(this->parent()).
- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*
- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.
- void **forms_end** ()

  This is for forms compatibility only.
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

  This does insert(w, find(before)).
- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.
- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.
- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

Gets the current user data (long) argument that is passed to the callback function.

- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ **as_gl_window** () const
- virtual Fl_Window ∗ as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ **as_window** () const
- void bind_deimage (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗cb, void ∗p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.

- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

Sets the image to use as part of the widget label when in the active state.

- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.

- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

Get the spacing between the label and the vertical edge of the widget.

- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Public Attributes

- Fl_Scrollbar **hscrollbar**
- Fl_Scrollbar **scrollbar**

Protected Member Functions

- void bbox (int &, int &, int &, int &) const

  Returns the bounding box for the interior of the scrolling area, inside the scrollbars.
- void draw () override

  Draws the widget.
- void fix_scrollbar_order ()

  Ensure the scrollbars are the last children.
- int on_insert (Fl_Widget *, int) override

  Change insert position of a child before it is added.
- int on_move (int, int) override

  Change new position of a child before it is moved.
- void recalc_scrollbars (ScrollInfo &si) const

  Calculate visibility/size/position of scrollbars, find children's bounding box.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()

  Returns the internal array of widget sizes and positions.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- virtual void on_remove (int)

  Allow derived groups to act when a child widget is removed from the group.
- int * sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()

  Returns the currently active group.
- static void current (Fl_Group *g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.116.1 Detailed Description

This container widget lets you maneuver around a set of widgets much larger than your window.
If the child widgets are larger than the size of this object then scrollbars will appear so that you can scroll over to them:

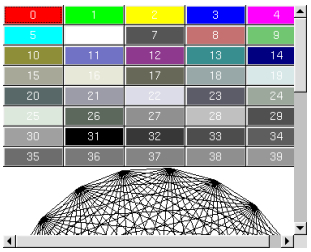


Figure 13.41 Fl_Scroll

If all of the child widgets are packed together into a solid rectangle then you want to set box() to FL_NO_BOX or one of the _FRAME types. This will result in the best output. However, if the child widgets are a sparse arrangement you must set box() to a real _BOX type. This can result in some blinking during redrawing, but that can be solved by using a Fl_Double_Window.

The Fl_Scroll widget calculates the bounding box of all its children by using their widget positions and sizes (x, y, w, h). Outside labels are not considered. If you need outside labels of any widgets or free space outside of this bounding box you can add a tiny invisible Fl_Box at the relevant corner(s) of the Fl_Scroll widget, for instance:

```
Fl_Scroll scroll(100, 100, 200, 200); // Fl_Scroll at (100, 100)
Fl_Box(100, 100, 1, 1);               // Fl_Box in top left corner
Fl_Input(150, 120, 60, 30, "Input:"); // left most widget with label
// ... more widgets ...
scroll.end();
```

By default you can scroll in both directions, and the scrollbars disappear if the data will fit in the area of the scroll. Use Fl_Scroll::type() to change this as follows :

- 0 - No scrollbars
- Fl_Scroll::HORIZONTAL - Only a horizontal scrollbar.
- Fl_Scroll::VERTICAL - Only a vertical scrollbar.
- Fl_Scroll::BOTH - The default is both scrollbars.
- Fl_Scroll::HORIZONTAL_ALWAYS - Horizontal scrollbar always on, vertical always off.
- Fl_Scroll::VERTICAL_ALWAYS - Vertical scrollbar always on, horizontal always off.
- Fl_Scroll::BOTH_ALWAYS - Both always on.

Use **scrollbar.align(int) (see void Fl_Widget::align(Fl_Align)) :** to change what side the scrollbars are drawn on.
If the FL_ALIGN_LEFT bit is on, the vertical scrollbar is on the left. If the FL_ALIGN_TOP bit is on, the horizontal scrollbar is on the top. Note that only the alignment flags in scrollbar are considered. The flags in hscrollbar however are ignored.
This widget can also be used to pan around a single child widget "canvas". This child widget should be of your own class, with a draw() method that draws the contents. The scrolling is done by changing the x() and y() of the widget, so this child must use the x() and y() to position its drawing. To speed up drawing it should test fl_not_clipped(int x,int y,int w,int h) to find out if a particular area of the widget must be drawn.
Another very useful child is a single Fl_Pack, which is itself a group that packs its children together and changes size to surround them. Filling the Fl_Pack with Fl_Tabs groups (and then putting normal widgets inside those) gives you a very powerful scrolling list of individually-openable panels.
Fluid lets you create these, but you can only lay out objects that fit inside the Fl_Scroll without scrolling. Be sure to leave space for the scrollbars, as Fluid won't show these either.
You cannot use Fl_Window as a child of this since the clipping is not conveyed to it when drawn, and it will draw over the scrollbars and neighboring objects.
Widget overlap may result in scroll tearing, see Fl_Tabs for fuller explanation.

13.116.2 Constructor & Destructor Documentation

13.116.2.1 Fl_Scroll()

```
Fl_Scroll::Fl_Scroll (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

Creates a new Fl_Scroll widget using the given position, size, and label string.
The default boxtype is FL_NO_BOX.
The destructor *also deletes all the children*. This allows a whole tree to be deleted at once, without having to keep a pointer to all the children in the user code. A kludge has been done so the Fl_Scroll and all of its children can be automatic (local) variables, but you must declare the Fl_Scroll *first*, so that it is destroyed last.

13.116.2.2 ~Fl_Scroll()

```
Fl_Scroll::~Fl_Scroll ( ) [virtual]
```

The destructor also deletes all the children.

See also

Fl_Group::~Fl_Group()

13.116.3 Member Function Documentation

13.116.3.1 bbox()

```
void Fl_Scroll::bbox (
            int & X,
            int & Y,
            int & W,
            int & H ) const  [protected]
```

Returns the bounding box for the interior of the scrolling area, inside the scrollbars.
This method does not change the scrollbars or their visibility. First the scrollbar positions and visibility are calculated as they **should** be, according to the positions and sizes of the children. Then the bounding box is calculated.
You may need to call redraw() to make sure the widget gets updated.

See also

recalc_scrollbars()

13.116.3.2 delete_child()

```
int Fl_Scroll::delete_child (
            int index )  [override], [virtual]
```

Removes the widget at `index` from the group and deletes it.
This method does nothing if `index` is out of bounds or if Fl_Group::child(index) is one of the scrollbars.

Parameters

in	*index*	index of child to be removed

Returns

success (0) or error code

Return values

0	success
1	index out of range
2	widget not allowed to be removed (see note)

See also

Fl_Group::delete_child(int index)

Since

FLTK 1.4.0

Reimplemented from Fl_Group.

13.116.3.3 draw()

```
void Fl_Scroll::draw ( )  [override], [protected], [virtual]
```

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.

Override this function to draw your own widgets.
If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();                 // calls Fl_Scrollbar::draw()
```

Reimplemented from Fl_Group.

13.116.3.4 fix_scrollbar_order()

```
void Fl_Scroll::fix_scrollbar_order ( ) [protected]
```

Ensure the scrollbars are the last children.
When Fl_Scroll is instantiated the first child of the Fl_Group is the vertical scrollbar `scrollbar` and the second child is the horizontal scrollbar `hscrollbar`.
These two widgets must always be the last two widgets and in this order to guarantee the correct drawing order and event delivery.
Since FLTK 1.4.0 the new method on_insert() modifies the insert position of other children if it would be after the scrollbars.

13.116.3.5 handle()

```
int Fl_Scroll::handle (
            int event ) [override], [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to *override* the behavior of the base class. This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to *filter* event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you *consumed* the event by returning 1 even if you didn't *do* anything with it.

Parameters

in	*event*	the kind of event received

Return values

0	if the event was not used or understood
1	if the event was used and can be deleted

See also

Fl_Event

Reimplemented from Fl_Group.

13.116.3.6 on_insert()

```
int Fl_Scroll::on_insert (
            Fl_Widget * candidate,
            int index ) [override], [protected], [virtual]
```

Change insert position of a child before it is added.
Fix insert position if the new child is planned to be inserted after the scrollbars. We can assume that the scrollbars are always the last two children!

Fl_Group calls this when a new widget is added. We return the new index if the new widget would be added after the scrollbars.

Parameters

in	*candidate*	the candidate will be added to the child array_ after this method returns.
in	*index*	add the child at this position in the array_

Returns

index to position the child as planned

a new index to force the child to a different position

-1 to keep the group from adding the candidate

Version

1.4.0

See also

Fl_Group::on_insert(Fl_Widget *candidate, int index)

Reimplemented from Fl_Group.

13.116.3.7 on_move()

```
int Fl_Scroll::on_move (
            int old_index,
            int new_index )  [override], [protected], [virtual]
```

Change new position of a child before it is moved.
Fix new position if the new child is planned to be moved after the scrollbars. We can assume that the scrollbars are always the last two children!
Fl_Group calls this when a widget is moved within the list of children. We return a new index if the widget would be moved after the scrollbars.

Parameters

old_index	the current index of the child that will be moved
new_index	the new index of the child

Returns

new index, possibly corrected to avoid last two scrollbar entries

Reimplemented from Fl_Group.

13.116.3.8 recalc_scrollbars()

```
void Fl_Scroll::recalc_scrollbars (
            ScrollInfo & si ) const  [protected]
```

Calculate visibility/size/position of scrollbars, find children's bounding box.
The `si` parameter will be filled with data from the calculations. Derived classes can make use of this call to figure out the scrolling area eg. during resize() handling.
This method does not change the scrollbars or their visibility. It calculates the scrollbar positions and visibility as they **should** be, according to the positions and sizes of the children.
You may need to call redraw() to make sure the widget gets updated.

Parameters

in,out	*si*	– ScrollInfo structure, filled with data

See also

bbox()

13.116.3.9 resize()

```
void Fl_Scroll::resize (
            int X,
            int Y,
            int W,
            int H )  [override], [virtual]
```

Resizes the Fl_Scroll widget and moves its children if necessary.
The Fl_Scroll widget first resizes itself, and then it moves all its children if (and only if) the Fl_Scroll widget has been moved. The children are moved by the same amount as the Fl_Scroll widget has been moved, hence all children keep their relative positions.

Note

Fl_Scroll::resize() does **not** call Fl_Group::resize(), and child widgets are **not** resized.

Since children of an Fl_Scroll are not resized, the resizable() widget is ignored (if it is set).
The scrollbars are moved to their proper positions, as given by Fl_Scroll::scrollbar.align(), and switched on or off as necessary.

Note

Due to current (FLTK 1.3.x) implementation constraints some of this may effectively be postponed until the Fl_Scroll is drawn the next time. This may change in a future release.

See also

Fl_Group::resizable()

Fl_Widget::resize(int,int,int,int)

Reimplemented from Fl_Group.

13.116.3.10 scroll_to()

```
void Fl_Scroll::scroll_to (
            int X,
            int Y )
```

Moves the contents of the scroll group to a new position.
This is like moving the scrollbars of the Fl_Scroll around. For instance:

```
Fl_Scroll scroll (10,10,200,200);
Fl_Box b1 ( 10, 10,50,50,"b1"); // relative (x,y) = (0,0)
Fl_Box b2 ( 60, 60,50,50,"b2"); // relative (x,y) = (50,50)
Fl_Box b3 ( 60,110,50,50,"b3"); // relative (x,y) = (50,100)
// populate scroll with more children ...
scroll.end();
scroll.scroll_to(50,100);
```

will move the logical origin of the internal scroll area to (-50,-100) relative to the origin of the Fl_Scroll (10,10), i.e. Fl_Box b3 will be visible in the top left corner of the scroll area.

13.116.3.11 scrollbar_size() [1/2]

```
int Fl_Scroll::scrollbar_size ( ) const  [inline]
```

Gets the current size of the scrollbars' troughs, in pixels.
If this value is zero (default), this widget will use the Fl::scrollbar_size() value as the scrollbar's width.

Returns

Scrollbar size in pixels, or 0 if the global Fl::scrollbar_size() is being used.

See also

Fl::scrollbar_size(int)

13.116.3.12 scrollbar_size() [2/2]

```
void Fl_Scroll::scrollbar_size (
            int newSize )  [inline]
```

Sets the pixel size of the scrollbars' troughs to `newSize`, in pixels.
Normally you should not need this method, and should use Fl::scrollbar_size(int) instead to manage the size of ALL your widgets' scrollbars. This ensures your application has a consistent UI, is the default behavior, and is normally what you want.
Only use THIS method if you really need to override the global scrollbar size. The need for this should be rare.
Setting `newSize` to the special value of 0 causes the widget to track the global Fl::scrollbar_size(), which is the default.

Parameters

in	*newSize*	Sets the scrollbar size in pixels. If 0 (default), scrollbar size tracks the global Fl::scrollbar_size()

See also

Fl::scrollbar_size()

13.116.3.13 xposition()

```
int Fl_Scroll::xposition ( ) const  [inline]
```

Gets the current horizontal scrolling position.

13.116.3.14 yposition()

```
int Fl_Scroll::yposition ( ) const  [inline]
```

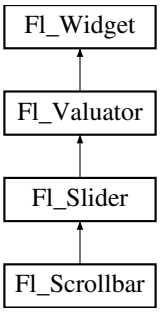
Gets the current vertical scrolling position.

The documentation for this class was generated from the following files:

- Fl_Scroll.H
- Fl_Scroll.cxx

13.117 Fl_Scrollbar Class Reference

The Fl_Scrollbar widget displays a slider with arrow buttons at the ends of the scrollbar.
`#include <Fl_Scrollbar.H>`
Inheritance diagram for Fl_Scrollbar:



Public Member Functions

- Fl_Scrollbar (int X, int Y, int W, int H, const char ∗L=0)

  Creates a new Fl_Scrollbar widget with given position, size, and label.
- int handle (int) override

  Handles the specified event.
- int **linesize** () const

  Get the size of step, in lines, that the arror keys move.
- void linesize (int i)

  This number controls how big the steps are that the arrow keys do.
- int value () const

  Gets the integer value (position) of the slider in the scrollbar.
- int value (int p)

  Sets the value (position) of the slider in the scrollbar.
- int value (int pos, int windowSize, int first_line, int total_lines)

  Sets the position, size and range of the slider in the scrollbar.
- ∼**Fl_Scrollbar** ()

  Destroys the Scrollbar.

Public Member Functions inherited from Fl_Slider

- void bounds (double a, double b)

  Sets the minimum (a) and maximum (b) values for the valuator widget.
- Fl_Slider (int X, int Y, int W, int H, const char ∗L=0)

  Creates a new Fl_Slider widget using the given position, size, and label string.
- **Fl_Slider** (uchar t, int X, int Y, int W, int H, const char ∗L)

  Creates a new Fl_Slider widget using the given type, position, size, and label string.
- int handle (int) override

  Handles the specified event.
- uchar num_ticks () const

  Gets the number of tick marks.
- Scale_Type scale () const

  Gets the scale type.
- void scale (Scale_Type s)

  Sets the scale type.
- int scrollvalue (int pos, int size, int first, int total)

  Sets the size and position of the sliding knob in the box.
- Fl_Boxtype **slider** () const

  Gets the slider box type.
- void **slider** (Fl_Boxtype c)

  Sets the slider box type.

- float **slider_size** () const

  Get the dimensions of the moving piece of slider.
- void slider_size (double v)

  Set the dimensions of the moving piece of slider.
- uchar ticks () const

  Gets the tick mark position.
- void ticks (uchar t, uchar num_ticks=11)

  Sets the tick mark position for Fl_Nice_Slider.

Public Member Functions inherited from Fl_Valuator

- void **bounds** (double a, double b)

  Sets the minimum (a) and maximum (b) values for the valuator widget.
- double **clamp** (double)

  Clamps the passed value to the valuator range.
- virtual int format (char ∗)

  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.
- virtual std::string format_str ()

  C++11 API for Fl_Valuator::format(char∗ buffer).
- double increment (double, int)

  Adds n times the step value to the passed value.
- double maximum () const

  Gets the maximum value for the valuator.
- void maximum (double a)

  Sets the maximum value for the valuator.
- double minimum () const

  Gets the minimum value for the valuator.
- void minimum (double a)

  Sets the minimum value for the valuator.
- void precision (int digits)

  Sets the step value to $1.0 / 10^{digits}$.
- void range (double a, double b)

  Sets the minimum and maximum values for the valuator.
- double round (double)

  Round the passed value to the nearest step increment.
- double step () const

  Gets or sets the step value.
- void **step** (double a, int b)

  See double Fl_Valuator::step() const
- void **step** (double s)

  See double Fl_Valuator::step() const.
- void **step** (int a)

  See double Fl_Valuator::step() const
- double value () const

  Gets the floating point(double) value.
- int value (double)

  Sets the current value.
- ∼**Fl_Valuator** () override

  Destructor is accessible despite protected constructor.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ **as_gl_window** () const
- virtual Fl_Group ∗ as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const ∗ **as_group** () const
- virtual Fl_Window ∗ as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ **as_window** () const
- void bind_deimage (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗cb, void ∗p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

Checks if the widget value changed since the last callback.

- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.

- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

Sets width ww and height hh accordingly with the label size.

- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.

- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void draw () override

  Draws the widget.

Protected Member Functions inherited from Fl_Slider

- void draw () override

  Draws the widget.
- void **draw** (int, int, int, int)
- void draw_ticks (const Fl_Rect &r, int S)

  Draw tick marks.
- int **handle** (int, int, int, int, int)
- double increment_lin_log (double v, int n, int range)

  Adds a sensible step value to the passed value.
- double position_to_value (double pos) const

  Converts a normalized position (0.0 to 1.0) to a value based on the current scale type.
- double value_to_position (double val) const

  Converts a value to a normalized position (0.0 to 1.0) based on the current scale type.

Protected Member Functions inherited from Fl_Valuator

- Fl_Valuator (int X, int Y, int W, int H, const char ∗L)

  Creates a new Fl_Valuator widget using the given position, size, and label string.
- void **handle_drag** (double newvalue)

  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.
- void **handle_push** ()

  Stores the current value in the previous value.
- void **handle_release** ()

  Called after an FL_WHEN_RELEASE event is received and before the callback.
- int **horizontal** () const

  Tells if the valuator is an FL_HORIZONTAL one.
- double **previous_value** () const

  Gets the previous floating point value before an event changed it.
- void **set_value** (double v)

  Sets the current floating point value.
- double **softclamp** (double)

  Clamps the value, but accepts v if the previous value is not already out of range.
- virtual void value_damage ()

  Asks for partial redraw.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.

- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Public Types inherited from Fl_Slider

- enum Scale_Type : uchar { LINEAR_SCALE = 0 , LOG_SCALE }

  Bitset for tick mark positions for Fl_Nice_Slider.
- enum Tick_Position : uchar {
  TICKS_NONE = 0x00 , TICKS_BELOW = 0x01 , TICKS_ABOVE = 0x02 , TICKS_LEFT = 0x01 ,
  TICKS_RIGHT = 0x02 }

  Bitset for tick mark positions for Fl_Nice_Slider.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,

AUTO_DELETE_USER_DATA = 1<<23 ,
MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
USERFLAG1 = 1<<31 }

flags possible values enumeration.

13.117.1 Detailed Description

The Fl_Scrollbar widget displays a slider with arrow buttons at the ends of the scrollbar.
Clicking on the arrows move up/left and down/right by linesize(). Scrollbars also accept FL_SHORTCUT events: the arrows move by linesize(), and vertical scrollbars take Page Up/Down (they move by the page size minus linesize()) and Home/End (they jump to the top or bottom).
Scrollbars have step(1) preset (they always return integers). If desired you can set the step() to non-integer values. You will then have to use casts to get at the floating-point versions of value() from Fl_Slider.

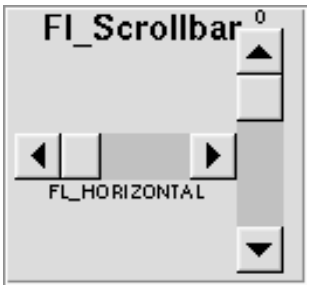


Figure 13.42 Fl_Scrollbar

13.117.2 Constructor & Destructor Documentation

13.117.2.1 Fl_Scrollbar()

```
Fl_Scrollbar::Fl_Scrollbar (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

Creates a new Fl_Scrollbar widget with given position, size, and label.
You need to do type(FL_HORIZONTAL) if you want a horizontal scrollbar.

13.117.3 Member Function Documentation

13.117.3.1 draw()

```
void Fl_Scrollbar::draw ( )  [override], [protected], [virtual]
```

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();                 // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

13.117.3.2 handle()

```
int Fl_Scrollbar::handle (
            int event )  [override], [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.

When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to *override* the behavior of the base class. This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to *filter* event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you *consumed* the event by returning 1 even if you didn't *do* anything with it.

Parameters

in	*event*	the kind of event received

Return values

0	if the event was not used or understood
1	if the event was used and can be deleted

See also

Fl_Event

Reimplemented from Fl_Widget.

13.117.3.3 linesize()

```
void Fl_Scrollbar::linesize (
            int i )  [inline]
```

This number controls how big the steps are that the arrow keys do.
In addition page up/down move by the size last sent to value() minus one linesize(). The default is 16.

13.117.3.4 value() [1/3]

```
int Fl_Scrollbar::value ( ) const  [inline]
```

Gets the integer value (position) of the slider in the scrollbar.
You can get the floating point value with Fl_Slider::value().

See also

Fl_Scrollbar::value(int p)

Fl_Scrollbar::value(int pos, int size, int first, int total)

13.117.3.5 value() [2/3]

```
int Fl_Scrollbar::value (
            int p )  [inline]
```

Sets the value (position) of the slider in the scrollbar.

See also

Fl_Scrollbar::value()

Fl_Scrollbar::value(int pos, int size, int first, int total)

13.117.3.6 value() [3/3]

```
int Fl_Scrollbar::value (
            int pos,
            int windowSize,
            int first_line,
            int total_lines )  [inline]
```

Sets the position, size and range of the slider in the scrollbar.

Parameters

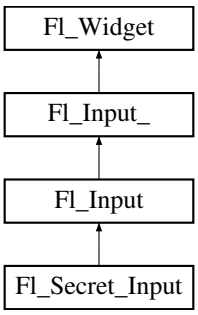
in	*pos*	position, first line displayed
in	*windowSize*	number of lines displayed
in	*first_line*	number of first line
in	*total_lines*	total number of lines

You should call this every time your window changes size, your data changes size, or your scroll position changes (even if in response to a callback from this scrollbar). All necessary calls to redraw() are done.
Calls Fl_Slider::scrollvalue(int pos, int size, int first, int total).
The documentation for this class was generated from the following files:

- Fl_Scrollbar.H
- Fl_Scrollbar.cxx

13.118 Fl_Secret_Input Class Reference

The Fl_Secret_Input class is a subclass of Fl_Input that displays its input as a string of placeholders.
`#include <Fl_Secret_Input.H>`
Inheritance diagram for Fl_Secret_Input:



Public Member Functions

- Fl_Secret_Input (int X, int Y, int W, int H, const char *l=0)

  Creates a new Fl_Secret_Input widget using the given position, size, and label string.
- int handle (int) override

  Handles the specified event.

Public Member Functions inherited from Fl_Input

- Fl_Input (int, int, int, int, const char *=0)

  Creates a new Fl_Input widget using the given position, size, and label string.

Public Member Functions inherited from Fl_Input_

- int append (const char *t, int l=0, char keep_selection=0)

Append text at the end.

- bool can_redo () const

  Check if there is a redo action available.
- bool can_undo () const

  Check if the last operation can be undone.
- int copy (int clipboard)

  Put the current selection into the clipboard.
- int copy_cuts ()

  Copies the yank buffer to the clipboard.
- Fl_Color cursor_color () const

  Gets the color of the cursor.
- void cursor_color (Fl_Color n)

  Sets the color of the cursor.
- int cut ()

  Deletes the current selection.
- int cut (int a, int b)

  Deletes all characters between index `a` and `b`.
- int cut (int n)

  Deletes the next `n` bytes rounded to characters before or after the cursor.
- double dvalue () const

  Returns the widget text interpreted as a floating point number.
- Fl_Input_ (int, int, int, int, const char *=0)

  Creates a new Fl_Input_ widget.
- unsigned int index (int i) const

  Returns the character at index `i`.
- int input_type () const

  Gets the input field type.
- void input_type (int t)

  Sets the input field type.
- int insert (const char *t, int l=0)

  Inserts text at the cursor position.
- int insert_position () const

  Gets the position of the text cursor.
- int insert_position (int p)

  Sets the cursor position and mark.
- int insert_position (int p, int m)

  Sets the index for the cursor and mark.
- int ivalue () const

  Returns the widget text interpreted as a signed integer.
- int mark () const

  Gets the current selection mark.
- int mark (int m)

  Sets the current selection mark.
- int maximum_size () const

  Gets the maximum length of the input field in characters.
- void maximum_size (int m)

  Sets the maximum length of the input field in characters.
- const char * placeholder () const

  Get the placeholder text.
- void placeholder (const char *text)

  Set the placeholder text.

- int position () const
- int position (int p)
- int position (int p, int m)
- int readonly () const

  Gets the read-only state of the input field.
- void readonly (int b)

  Sets the read-only state of the input field.
- int redo ()

  Redo previous undo operation.
- int replace (int b, int e, const char ∗text, int ilen=0)

  Deletes text from `b` to `e` and inserts the new string `text`.
- void resize (int, int, int, int) override

  Changes the size of the widget.
- int shortcut () const

  Return the shortcut key associated with this widget.
- void shortcut (int s)

  Sets the shortcut key associated with this widget.
- int size () const

  Returns the number of bytes in value().
- void size (int W, int H)

  Sets the width and height of this widget.
- int static_value (const char ∗)

  Changes the widget text.
- int static_value (const char ∗, int)

  Changes the widget text.
- int tab_nav () const

  Gets whether the Tab key causes focus navigation in multiline input fields or not.
- void tab_nav (int val)

  Sets whether the Tab key does focus navigation, or inserts tab characters into Fl_Multiline_Input.
- Fl_Color textcolor () const

  Gets the color of the text in the input field.
- void textcolor (Fl_Color n)

  Sets the color of the text in the input field.
- Fl_Font textfont () const

  Gets the font of the text in the input field.
- void textfont (Fl_Font s)

  Sets the font of the text in the input field.
- Fl_Fontsize textsize () const

  Gets the size of the text in the input field.
- void textsize (Fl_Fontsize s)

  Sets the size of the text in the input field.
- int undo ()

  Undoes previous changes to the text buffer.
- const char ∗ value () const

  Returns the text displayed in the widget.
- int value (const char ∗)

  Changes the widget text.
- int value (const char ∗, int)

  Changes the widget text.
- int value (double value)

  Changes the widget text to a floating point number ("%g").

- int value (int value)

  Changes the widget text to a signed integer number.
- int wrap () const

  Gets the word wrapping state of the input field.
- void wrap (int b)

  Sets the word wrapping state of the input field.
- ~Fl_Input_ ()

  Destroys the widget.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

Sets the current callback function for the widget.

- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.

- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

Sets the font to use.

- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.

- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` *contains the entered '&x' shortcut.*

Static Public Attributes inherited from Fl_Input

- static const char * **copy_menu_text** = "Copy"

  [this text may be customized at run-time]
- static const char * **cut_menu_text** = "Cut"

  [this text may be customized at run-time]
- static const char * **paste_menu_text** = "Paste"

  [this text may be customized at run-time]

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Input

- void draw () override

  Draws the widget.
- int handle_key ()

  Handles a keystroke.
- int handle_rmb ()

  Handle right mouse button down events.

Protected Member Functions inherited from Fl_Input_

- int apply_undo ()

  Apply the current undo/redo operation.
- void drawtext (int, int, int, int)

  Draws the text in the passed bounding box.
- void drawtext (int, int, int, int, bool draw_active)

  Draws the text in the passed bounding box.

- void handle_mouse (int, int, int, int, int keepmark=0)

  Handles mouse clicks and mouse moves.
- int handletext (int e, int, int, int, int)

  Handles all kinds of text field related events.
- int line_end (int i) const

  Finds the end of a line.
- int line_start (int i) const

  Finds the start of a line.
- int **linesPerPage** ()
- void **maybe_do_callback** (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
- int up_down_position (int, int keepmark=0)

  Moves the cursor to the column given by `up_down_pos`.
- int word_end (int i) const

  Finds the end of a word.
- int word_start (int i) const

  Finds the start of a word.
- int **xscroll** () const
- int **yscroll** () const
- void **yscroll** (int yOffset)

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

Internal use only.

- void x (int v)

  Internal use only.

- void y (int v)

  Internal use only.

13.118.1 Detailed Description

The Fl_Secret_Input class is a subclass of Fl_Input that displays its input as a string of placeholders. Depending on the platform this placeholder is either the asterisk ('∗') or the Unicode bullet character (U+2022). This subclass is usually used to receive passwords and other "secret" information.

13.118.2 Constructor & Destructor Documentation

13.118.2.1 Fl_Secret_Input()

```
Fl_Secret_Input::Fl_Secret_Input (
            int X,
            int Y,
            int W,
            int H,
            const char * l = 0 )
```

Creates a new Fl_Secret_Input widget using the given position, size, and label string.
The default boxtype is FL_DOWN_BOX.
Inherited destructor destroys the widget and any value associated with it.

13.118.3 Member Function Documentation

13.118.3.1 handle()

```
int Fl_Secret_Input::handle (
            int event )  [override], [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to *override* the behavior of the base class. This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to *filter* event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you *consumed* the event by returning 1 even if you didn't *do* anything with it.

Parameters

in	*event*	the kind of event received

Return values

0	if the event was not used or understood
1	if the event was used and can be deleted

See also

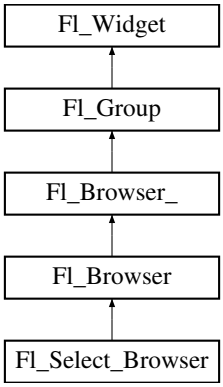
Fl_Event

Reimplemented from Fl_Input.
The documentation for this class was generated from the following files:

- Fl_Secret_Input.H
- Fl_Input.cxx

13.119 Fl_Select_Browser Class Reference

The class is a subclass of Fl_Browser which lets the user select a single item, or no items by clicking on the empty space.
`#include <Fl_Select_Browser.H>`
Inheritance diagram for Fl_Select_Browser:



Public Member Functions

- Fl_Select_Browser (int X, int Y, int W, int H, const char ∗L=0)

  Creates a new Fl_Select_Browser widget using the given position, size, and label string.

Public Member Functions inherited from Fl_Browser

- void add (const char ∗newtext, void ∗d=0)

  Adds a new line to the end of the browser.
- void bottomline (int line)

  Scrolls the browser so the bottom item in the browser is showing the specified `line`.
- void clear ()

  Removes all the lines in the browser.
- char column_char () const

  Gets the current column separator character.
- void column_char (char c)

  Sets the column separator to c.
- const int ∗ column_widths () const

  Gets the current column width array.
- void column_widths (const int ∗arr)

  Sets the current array to `arr`.
- void ∗ data (int line) const

  Returns the user data() for specified `line`.
- void data (int line, void ∗d)

  Sets the user data for specified `line` to `d`.

- void display (int line, int val=1)

  For back compatibility.
- int displayed (int line) const

  Returns non-zero if `line` has been scrolled to a position where it is being displayed.
- Fl_Browser (int X, int Y, int W, int H, const char *L=0)

  The constructor makes an empty browser.
- char format_char () const

  Gets the current format code prefix character, which by default is '@'.
- void format_char (char c)

  Sets the current format code prefix character to `c`.
- void hide () override

  Hides the entire Fl_Browser widget – opposite of show().
- void hide (int line)

  Makes `line` invisible, preventing selection by the user.
- Fl_Image * icon (int line) const

  Returns the icon currently defined for `line`.
- void icon (int line, Fl_Image *icon)

  Set the image icon for `line` to the value `icon`.
- void insert (int line, const char *newtext, void *d=0)

  Insert a new entry whose label is `newtext` above given `line`, optional data `d`.
- void lineposition (int line, Fl_Line_Position pos)

  Updates the browser so that `line` is shown at position `pos`.
- int load (const char *filename)

  Clears the browser and reads the file, adding each line from the file to the browser.
- void make_visible (int line)

  Make the item at the specified `line` visible().
- void middleline (int line)

  Scrolls the browser so the middle item in the browser is showing the specified `line`.
- void move (int to, int from)

  Line `from` is removed and reinserted at `to`.
- void remove (int line)

  Remove entry for given `line` number, making the browser one line shorter.
- void remove_icon (int line)

  Removes the icon for `line`.
- void **replace** (int a, const char *b)

  For back compatibility only.
- int select (int line, int val=1)

  Sets the selection state of the item at `line` to the value `val`.
- int selected (int line) const

  Returns 1 if specified `line` is selected, 0 if not.
- void show () override

  Shows the entire Fl_Browser widget – opposite of hide().
- void show (int line)

  Makes `line` visible, and available for selection by user.
- int size () const

  Returns how many lines are in the browser.
- void **size** (int W, int H)
- void swap (int a, int b)

  Swaps two browser lines `a` and `b`.
- const char * text (int line) const

  Returns the label text for the specified `line`.

- void text (int line, const char *newtext)

  Sets the text for the specified `line` to `newtext`.
- Fl_Fontsize **textsize** () const

  Gets the default text size (in pixels) for the lines in the browser.
- void textsize (Fl_Fontsize newSize)

  Sets the default text size (in pixels) for the lines in the browser to `newSize`.
- int topline () const

  Returns the line that is currently visible at the top of the browser.
- void topline (int line)

  Scrolls the browser so the top item in the browser is showing the specified `line`.
- int value () const

  Returns the line number of the currently selected line, or 0 if none selected.
- void value (int line)

  Sets the browser's value(), which selects the specified `line`.
- int visible (int line) const

  Returns non-zero if the specified `line` is visible, 0 if hidden.
- ~**Fl_Browser** ()

  The destructor deletes all list items and destroys the browser.

Public Member Functions inherited from Fl_Browser_

- int deselect (int docallbacks=0)

  Deselects all items in the list and returns 1 if the state changed or 0 if it did not.
- void display (void *item)

  Displays the `item`, scrolling the list as necessary.
- int handle (int event) override

  Handles the `event` within the normal widget bounding box.
- uchar **has_scrollbar** () const

  Returns the current scrollbar mode, see Fl_Browser_::has_scrollbar(uchar)
- void has_scrollbar (uchar mode)

  Sets whether the widget should have scrollbars or not (default Fl_Browser_::BOTH).
- int hposition () const

  Gets the horizontal scroll position of the list as a pixel position `pos`.
- void hposition (int)

  Sets the horizontal scroll position of the list to pixel position `pos`.
- int linespacing () const

  Return the height of additional spacing between browser lines.
- void linespacing (int pixels)

  Add some space between browser lines.
- int position () const
- void position (int pos)
- void **position** (int x, int y)
- void resize (int X, int Y, int W, int H) override

  Repositions and/or resizes the browser.
- void scrollbar_left ()

  Moves the vertical scrollbar to the lefthand side of the list.
- void scrollbar_right ()

  Moves the vertical scrollbar to the righthand side of the list.
- int scrollbar_size () const

  Gets the current size of the scrollbars' troughs, in pixels.
- void scrollbar_size (int newSize)

Sets the pixel size of the scrollbars' troughs to `newSize`, in pixels.

- int select (void *item, int val=1, int docallbacks=0)

  Sets the selection state of `item` to `val`, and returns 1 if the state changed or 0 if it did not.
- int select_only (void *item, int docallbacks=0)

  Selects `item` and returns 1 if the state changed or 0 if it did not.
- void sort (int flags=0)

  Sort the items in the browser based on `flags`.
- Fl_Color **textcolor** () const

  Gets the default text color for the lines in the browser.
- void **textcolor** (Fl_Color col)

  Sets the default text color for the lines in the browser to color `col`.
- Fl_Font textfont () const

  Gets the default text font for the lines in the browser.
- void **textfont** (Fl_Font font)

  Sets the default text font for the lines in the browser to `font`.
- Fl_Fontsize **textsize** () const

  Gets the default text size (in pixels) for the lines in the browser.
- void **textsize** (Fl_Fontsize newSize)

  Sets the default text size (in pixels) for the lines in the browser to `size`.
- int vposition () const

  Gets the vertical scroll position of the list as a pixel position `pos`.
- void vposition (int pos)

  Sets the vertical scroll position of the list to pixel position `pos`.

Public Member Functions inherited from Fl_Group

- Fl_Widget *& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget *o)

  See void Fl_Group::add(Fl_Widget &w)
- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget *const * array () const

  Returns a pointer to the internal array of children.
- Fl_Group const * as_group () const override
- Fl_Group * as_group () override

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const

  Returns the n'th child.
- int **children** () const

  Returns how many child widgets the group has.
- void clear ()

  Deletes all child widgets from memory recursively.
- unsigned int clip_children () const

  Returns the current clipping mode.
- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.

- virtual int delete_child (int n)

  Removes the widget at `index` from the group and deletes it.
- void end ()

  Exactly the same as current(this->parent()).
- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*
- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.
- void **forms_end** ()

  This is for forms compatibility only.
- int handle (int) override

  Handles the specified event.
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

  This does insert(w, find(before)).
- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.
- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.
- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- void resize (int, int, int, int) override

  Resizes the Fl_Group widget and all of its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.

- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

Sets the background color of the widget.

- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.

- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

Gets the selection color.

- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.

- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Public Types inherited from Fl_Browser

- enum Fl_Line_Position { **TOP** , **BOTTOM** , **MIDDLE** }

  For internal use only?

Public Types inherited from Fl_Browser_

- enum {
  HORIZONTAL = 1 , VERTICAL = 2 , BOTH = 3 , ALWAYS_ON = 4 ,
  HORIZONTAL_ALWAYS = 5 , VERTICAL_ALWAYS = 6 , BOTH_ALWAYS = 7 }

  Values for has_scrollbar().

Static Public Member Functions inherited from Fl_Group

- static Fl_Group ∗ current ()

  Returns the currently active group.
- static void current (Fl_Group ∗g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget ∗widget, void ∗data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char ∗t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char ∗, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Public Attributes inherited from Fl_Browser_

- Fl_Scrollbar hscrollbar

  Horizontal scrollbar.
- Fl_Scrollbar scrollbar

  Vertical scrollbar.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Browser

- FL_BLINE * _remove (int line)

  Removes the item at the specified `line`.
- const void * **bline_data** (const FL_BLINE *b) const

  Get FL_BLINE data.
- void *& **bline_data** (FL_BLINE *b) const

  Get writable reference to FL_BLINE data.
- char **bline_flags** (const FL_BLINE *b) const

  Get FL_BLINE flags.
- char & **bline_flags** (FL_BLINE *b) const

  Get writable reference to FL_BLINE flags.
- const Fl_Image * **bline_icon** (const FL_BLINE *b) const

  Get FL_BLINE icon.
- Fl_Image *& **bline_icon** (FL_BLINE *b) const

  Get writable reference to FL_BLINE icon.
- short bline_length (const FL_BLINE *b) const

  Get FL_BLINE allocated text buffer size (excl.
- const char * **bline_txt** (const FL_BLINE *b) const

  Get FL_BLINE text.
- char * **bline_txt** (FL_BLINE *b) const

  Get writable reference to FL_BLINE text.
- FL_BLINE * find_line (int line) const

  Returns the item for specified `line`.
- int full_height () const override

  The height of the entire list of all visible() items in pixels.
- int incr_height () const override

  The default 'average' item height (including inter-item spacing) in pixels.
- void insert (int line, FL_BLINE *item)

Insert specified `item` above `line`.

- void * item_at (int line) const override

  Return the item at specified `line`.
- void item_draw (void *item, int X, int Y, int W, int H) const override

  Draws `item` at the position specified by `X Y W H`.
- void * item_first () const override

  Returns the very first item in the list.
- int item_height (void *item) const override

  Returns height of `item` in pixels.
- void * item_last () const override

  Returns the very last item in the list.
- void * item_next (void *item) const override

  Returns the next item after `item`.
- void * item_prev (void *item) const override

  Returns the previous item before `item`.
- void item_select (void *item, int val) override

  Change the selection state of `item` to the value `val`.
- int item_selected (void *item) const override

  See if `item` is selected.
- void item_swap (void *a, void *b) override

  Swap the items `a` and `b`.
- const char * item_text (void *item) const override

  Returns the label text for `item`.
- int item_width (void *item) const override

  Returns width of `item` in pixels.
- int lineno (void *item) const

  Returns line number corresponding to `item`, or zero if not found.
- void swap (FL_BLINE *a, FL_BLINE *b)

  Swap the two items `a` and `b`.

Protected Member Functions inherited from Fl_Browser_

- void bbox (int &X, int &Y, int &W, int &H) const

  Returns the bounding box for the interior of the list's display window, inside the scrollbars.
- void deleting (void *item)

  This method should be used when `item` is being deleted from the list.
- int displayed (void *item) const

  Returns non-zero if `item` has been scrolled to a position where it is being displayed.
- void draw () override

  Draws the list within the normal widget bounding box.
- void * find_item (int ypos)

  This method returns the item under mouse y position `ypos`.
- Fl_Browser_ (int X, int Y, int W, int H, const char *L=0)

  The constructor makes an empty browser.
- virtual int full_width () const

  This method may be provided by the subclass to indicate the full width of the item list, in pixels.
- void inserting (void *a, void *b)

  This method should be used when an item is in the process of being inserted into the list.
- virtual int item_quick_height (void *item) const

  This method may be provided by the subclass to return the height of the `item`, in pixels.
- int leftedge () const

This method returns the X position of the left edge of the list area after adjusting for the scrollbar and border, if any.

- void new_list ()

  This method should be called when the list data is completely replaced or cleared.
- void redraw_line (void *item)

  This method should be called when the contents of `item` has changed, but not its height.
- void redraw_lines ()

  This method will cause the entire list to be redrawn.
- void replacing (void *a, void *b)

  This method should be used when item `a` is being replaced by item `b`.
- void * selection () const

  Returns the item currently selected, or NULL if there is no selection.
- void swapping (void *a, void *b)

  This method should be used when two items `a` and `b` are being swapped.
- void * **top** () const

  Returns the item that appears at the top of the list.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()

  Returns the internal array of widget sizes and positions.
- void draw () override

  Draws the widget.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget *, int)

  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)

  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)

  Allow derived groups to act when a child widget is removed from the group.
- int * sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

Draws a focus rectangle around the widget.

- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Static Protected Attributes inherited from Fl_Browser

- static constexpr char **BLINE_NOTDISPLAYED** = 2
- static constexpr char **BLINE_SELECTED** = 1

13.119.1 Detailed Description

The class is a subclass of Fl_Browser which lets the user select a single item, or no items by clicking on the empty space.
As long as the mouse button is held down on an unselected item it is highlighted. Normally the callback is done when the user presses the mouse, but you can change this with when().
See Fl_Browser for methods to add and remove lines from the browser.

13.119.2 Constructor & Destructor Documentation

13.119.2.1 Fl_Select_Browser()

```
Fl_Select_Browser::Fl_Select_Browser (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

Creates a new Fl_Select_Browser widget using the given position, size, and label string.
The default boxtype is FL_DOWN_BOX. The constructor specializes Fl_Browser() by setting the type to FL_↩ SELECT_BROWSER. The destructor destroys the widget and frees all memory that has been allocated.
The documentation for this class was generated from the following files:

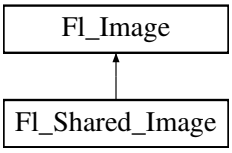
- Fl_Select_Browser.H
- Fl_Browser.cxx

13.120 Fl_Shared_Image Class Reference

This class supports caching, loading, and drawing of image files.

`#include <Fl_Shared_Image.H>`

Inheritance diagram for Fl_Shared_Image:



Classes

- class Deleter

  Custom deleter for std::unique_ptr<>.

Public Member Functions

- Fl_Shared_Image * as_shared_image () override

  Returns whether an image is an Fl_Shared_Image or not.
- void color_average (Fl_Color c, float i) override

  Averages the colors in the image with the provided FLTK color value.
- Fl_Image * copy ()

  Increments the reference counter and returns a pointer to itself.
- Fl_Image * **copy** () const
- Fl_Image * copy (int W, int H) const override

  Return a shared image of this image with the requested size.
- void desaturate () override

  Convert the image to gray scale.
- void **draw** (int X, int Y)
- void draw (int X, int Y, int W, int H, int cx=0, int cy=0) override

  Draw this image to the current graphics context.
- const Fl_Image * image () const

  Returns a pointer to the internal Fl_Image object.
- const char * **name** () const

  Returns the filename of the shared image.
- int original () const

  Returns whether this is an original image.
- int refcount () const

  Returns the number of references of this shared image.
- void release () override

  Releases and possibly destroys (if refcount <= 0) a shared image.
- virtual void **reload** ()

  Reloads the shared image from disk.
- void uncache () override

  Remove the cached device specific image data.

Public Member Functions inherited from Fl_Image

- Fl_Image * copy () const

  Creates a copy of the image in the same size.
- int count () const

  Returns the number of data values associated with the image.
- int d () const

  Returns the image depth.
- const char *const * data () const

  Returns a pointer to the current image data array.
- int **data_h** () const

  Returns the height of the image data.
- int **data_w** () const

  Returns the width of the image data.
- void draw (int X, int Y)

  Draws the image to the current drawing surface.
- int fail () const

  Returns a value that is not 0 if there is currently no image available.
- Fl_Image (int W, int H, int D)

  The constructor creates an empty image with the specified width, height, and depth.
- int h () const

  Returns the current image drawing height in FLTK units.
- void inactive ()

  The inactive() method calls color_average(FL_BACKGROUND_COLOR, 0.33f) to produce an image that appears grayed out.
- virtual void label (Fl_Menu_Item *m)

  This method is an obsolete way to set the image attribute of a menu item.
- virtual void label (Fl_Widget *w)

  This method is an obsolete way to set the image attribute of a widget or menu item.
- int ld () const

  Returns the current line data size in bytes.
- virtual void scale (int width, int height, int proportional=1, int can_expand=0)

  Sets the drawing size of the image.
- int w () const

  Returns the current image drawing width in FLTK units.
- virtual ~**Fl_Image** ()

  The destructor is a virtual method that frees all memory used by the image.

Static Public Member Functions

- static void add_handler (Fl_Shared_Handler f)

  Adds a shared image handler, which is basically a test function for adding new image formats.
- static Fl_Shared_Image * find (const char *name, int W=0, int H=0)

  Finds a shared image from its name and size specifications.
- static Fl_Shared_Image * get (const char *name, int W=0, int H=0)

  Find or load an image that can be shared by multiple widgets.
- static Fl_Shared_Image * get (Fl_RGB_Image *rgb, int own_it=1)

  Builds a shared image from a pre-existing Fl_RGB_Image.
- static Fl_Shared_Image ** images ()

  Returns the Fl_Shared_Image array.*
- static int num_images ()

  Number of shared images in their various cached sizes.
- static void **remove_handler** (Fl_Shared_Handler f)

  Removes a shared image handler.

Static Public Member Functions inherited from Fl_Image

- static Fl_Labeltype **define_FL_IMAGE_LABEL** ()
- static Fl_RGB_Scaling **RGB_scaling** ()

  Returns the currently used RGB image scaling method.
- static void RGB_scaling (Fl_RGB_Scaling)

  Sets the RGB image scaling method used for copy(int, int).
- static Fl_RGB_Scaling **scaling_algorithm** ()

  Gets what algorithm is used when resizing a source image to draw it.
- static void scaling_algorithm (Fl_RGB_Scaling algorithm)

  Sets what algorithm is used when resizing a source image to draw it.

Protected Member Functions

- void add ()

  Adds a shared image to the image pool.
- Fl_Shared_Image ∗ copy_ (int W, int H) const

  Create a resized copy of the image and wrap it into the share image class.
- Fl_Shared_Image ()

  Creates an empty shared image.
- Fl_Shared_Image (const char ∗n, Fl_Image ∗img=0)

  Creates a shared image from its filename and its corresponding Fl_Image∗ img.
- void update ()

  Update the dimensions of the shared images.
- virtual ∼Fl_Shared_Image ()

  The destructor frees all memory and server resources that are used by the image.

Protected Member Functions inherited from Fl_Image

- void **d** (int D)

  Sets the current image depth.
- void data (const char ∗const ∗p, int c)

  Sets the current data pointer and count of pointers in the array.
- void draw_empty (int X, int Y)

  The protected method draw_empty() draws a box with an X in it.
- int draw_scaled (int X, int Y, int W, int H)

  Draw the image to the current drawing surface rescaled to a given width and height.
- void h (int H)

  Sets the height of the image data.
- void ld (int LD)

  Sets the current line data size in bytes.
- void w (int W)

  Sets the width of the image data.

Static Protected Member Functions

- static int compare (Fl_Shared_Image ∗∗i0, Fl_Shared_Image ∗∗i1)

  Compares two shared images.

Static Protected Member Functions inherited from Fl_Image

- static void **labeltype** (const Fl_Label ∗lo, int lx, int ly, int lw, int lh, Fl_Align la)
- static void **measure** (const Fl_Label ∗lo, int &lw, int &lh)

Protected Attributes

- int **alloc_image_**
- Fl_Image ∗ **image_**
- const char ∗ **name_**
- int **original_**
- int **refcount_**

Static Protected Attributes

- static int **alloc_handlers_** = 0
- static int **alloc_images_** = 0
- static Fl_Shared_Handler ∗ **handlers_** = 0
- static Fl_Shared_Image ∗∗ **images_** = 0
- static int **num_handlers_** = 0
- static int **num_images_** = 0

Friends

- class **Fl_Graphics_Driver**
- class **Fl_JPEG_Image**
- class **Fl_PNG_Image**
- class **Fl_SVG_Image**

Additional Inherited Members

Static Public Attributes inherited from Fl_Image

- static const int **ERR_FILE_ACCESS** = -2
- static const int **ERR_FORMAT** = -3
- static const int **ERR_MEMORY_ACCESS** = -4
- static const int **ERR_NO_IMAGE** = -1
- static bool **register_images_done** = false

  True after fl_register_images() was called, false before.

13.120.1 Detailed Description

This class supports caching, loading, and drawing of image files.
Most applications will also want to link against the fltk_images library and call the fl_register_images() function to support standard image formats such as BMP, GIF, JPEG, PNG, and SVG (unless the library was built with the option removing SVG support).
Images can be requested (loaded) with Fl_Shared_Image::get(), find(), and some other methods. All images are cached in an internal list of shared images and should be released when they are no longer needed. A refcount is used to determine if a released image is to be destroyed with delete.

See also

fl_register_images()

Fl_Shared_Image::get()

Fl_Shared_Image::find()

Fl_Shared_Image::release()

13.120.2 Constructor & Destructor Documentation

13.120.2.1 Fl_Shared_Image() [1/2]

```
Fl_Shared_Image::Fl_Shared_Image ( )  [protected]
```

Creates an empty shared image.
The constructors create a new shared image record in the image cache.
The constructors are protected and cannot be used directly from a program. Use the get() method instead.

13.120.2.2 Fl_Shared_Image() [2/2]

```
Fl_Shared_Image::Fl_Shared_Image (
            const char * n,
            Fl_Image * img = 0 )  [protected]
```

Creates a shared image from its filename and its corresponding Fl_Image∗ img.
The constructors create a new shared image record in the image cache.
The constructors are protected and cannot be used directly from a program. Use the get() method instead.

Parameters

in	*n*	filename or pool name of the image, must be unique among shared images
in	*img*	the image that is made available using the name

13.120.2.3 ∼Fl_Shared_Image()

```
Fl_Shared_Image::~Fl_Shared_Image ( )  [protected], [virtual]
```

The destructor frees all memory and server resources that are used by the image.
The destructor is protected and cannot be used directly from a program. Use the Fl_Shared_Image::release() method instead.

13.120.3 Member Function Documentation

13.120.3.1 add()

```
void Fl_Shared_Image::add ( )  [protected]
```

Adds a shared image to the image pool.
This **protected** method adds an image to the pool, an ordered list of shared images. The pool is searched for a matching image whenever one is requested, for instance with Fl_Shared_Image::get() or Fl_Shared_Image::find().
This method does not increase or decrease reference counts!

13.120.3.2 add_handler()

```
void Fl_Shared_Image::add_handler (
            Fl_Shared_Handler f )  [static]
```

Adds a shared image handler, which is basically a test function for adding new image formats.
This function will be called when an Fl_Shared_Image is to be loaded (for instance with Fl_Shared_Image::get()) and the image type is not known to FLTK.
All registered image handlers will be called in the order of registration. You should always call fl_register_images() before adding your own handlers - unless you need to override a known image file type which should be rare.

See also

Fl_Shared_Handler for more information of the function you need to define.

13.120.3.3 as_shared_image()

```
Fl_Shared_Image * Fl_Shared_Image::as_shared_image ( )  [inline], [override], [virtual]
```

Returns whether an image is an Fl_Shared_Image or not.
This virtual method returns a pointer to an Fl_Shared_Image if this object is an instance of Fl_Shared_Image or NULL if not. This can be used to detect if a given Fl_Image object is a shared image, i.e. derived from Fl_Shared_Image.

Since

1.4.0

Reimplemented from Fl_Image.

13.120.3.4 color_average()

```
void Fl_Shared_Image::color_average (
            Fl_Color c,
            float i )  [override], [virtual]
```

Averages the colors in the image with the provided FLTK color value.
This method changes the pixel data of this specific image.

Note

It does not change any of the resized copies of this image, nor does it necessarily apply the color changes if this image is resized later.

Parameters

in	*c*	blend with this color
in	*i*	blend fraction

See also

Fl_Image::color_average(Fl_Color c, float i)

Reimplemented from Fl_Image.

13.120.3.5 compare()

```
int Fl_Shared_Image::compare (
            Fl_Shared_Image ** i0,
            Fl_Shared_Image ** i1 )  [static], [protected]
```

Compares two shared images.
The order of comparison is:

1. Image name, usually the filename used to load it
2. Image width
3. Image height

Binary search in a sorted array works only if we search for the same parameters that were also used for sorting. No special cases are possible here.
Fl_Shared_Image::find() requires a search for an element with a matching name and the original_ flags set. This is not implemented via binary search, but by a simple run of the array inside Fl_Shared_Image::find().

Parameters

in	*i0,i1*	image pointer pointer for sorting

Returns

Whether the images match or their relative sort order (see text).

Return values

0	the images match
<0	Image `i0` is *less* than image `i1`
>0	Image `i0` is *greater* than image `i1`

13.120.3.6 copy() [1/2]

```
Fl_Image * Fl_Shared_Image::copy ( )
```

Increments the reference counter and returns a pointer to itself.
When the image is no longer used, call `Fl_Shared_Image::release()`.
To get a copy of the image data, call `this->image()->copy()` instead.

Returns

pointer to an `Fl_Shared_Image` that can be safely cast

13.120.3.7 copy() [2/2]

```
Fl_Image * Fl_Shared_Image::copy (
            int W,
            int H ) const  [override], [virtual]
```

Return a shared image of this image with the requested size.
This is the same as calling `Fl_Shared_Image::get(this->name(), W, H)`.
If a shared image of the desired size already exists in the shared image pool, the existing image is returned and no copy is made. But the reference counter is incremented. When the image is no longer used, call `Fl_Shared_Image::release()`.
To get a copy of the image data, call `this->image()->copy(W, H)` instead.

Parameters

in	*W,H*	size of requested image

Returns

pointer to an `Fl_Shared_Image` that can be safely cast, or NULL if the image can't be found and can't be created.

Reimplemented from Fl_Image.

13.120.3.8 copy_()

```
Fl_Shared_Image * Fl_Shared_Image::copy_ (
            int W,
            int H ) const  [protected]
```

Create a resized copy of the image and wrap it into the share image class.
This function is usually followed by a call to `returned_image->add()` to add the image to the pool, and `this->refcounter_++` to make sure that the original shared image keeps a reference to the copy. Don't call this function if an image of the given size is already in the pool.

Parameters

in	*W,H*	new image size

Returns

a new shared image pointer that is not yet in the pool

13.120.3.9 desaturate()

```
void Fl_Shared_Image::desaturate ( )  [override], [virtual]
```

Convert the image to gray scale.
This method changes the pixel data of this specific image.

Note

It does not change any of the resized copies of this image, nor does it necessarily apply the color changes if this image is resized later.

See also

Fl_Image::desaturate()

Reimplemented from Fl_Image.

13.120.3.10 draw()

```
void Fl_Shared_Image::draw (
            int X,
            int Y,
            int W,
            int H,
            int cx = 0,
            int cy = 0 )  [override], [virtual]
```

Draw this image to the current graphics context.

Parameters

in	*X,Y,W,H*	draw at this position and size
in	*cx,cy*	image origin

Reimplemented from Fl_Image.

13.120.3.11 find()

```
Fl_Shared_Image * Fl_Shared_Image::find (
            const char * name,
            int W = 0,
            int H = 0 )  [static]
```

Finds a shared image from its name and size specifications.
This uses a binary search in the image cache.
If the image `name` exists with the exact width `W` and height `H`, then it is returned.
If `W` == 0 and the image `name` exists with another size, then the **original** image with that `name` is returned.
In either case the refcount of the returned image is increased. The found image should be released with Fl_Shared_Image::release() when no longer needed.
An image is marked `original` if it was directly loaded from a file or from memory as opposed to copied and resized images.
This comparison is used in Fl_Shared_Image::find() to find an image that matches the requested one or to find the position where a new image should be entered into the sorted list of shared images.
It is used in two steps by Fl_Shared_Image::add():

1. search with exact width and height
2. if not found, search again with width = 0 (and height = 0)

The first step will only return a match if the image exists with the same width and height. The second step will match if there is an image marked `original` with the same name, regardless of width and height.

13.120.3.12 get() [1/2]

```
Fl_Shared_Image * Fl_Shared_Image::get (
            const char * name,
            int W = 0,
            int H = 0 )  [static]
```

Find or load an image that can be shared by multiple widgets.
If the image exists with the requested size, this image will be returned.
If the image exists, but only with another size, then a new copy with the requested size (width `W` and height `H`) will be created as a resized copy of the original image. The new image is added to the internal list of shared images.
If the image does not yet exist, then a new image of the proper dimension is created from the filename `name`. The original image from filename `name` is always added to the list of shared images in its original size. If the requested size differs, then the resized copy with width `W` and height `H` is also added to the list of shared images.

Note

If the sizes differ, then *two* images are created as mentioned above. This is intentional so the original image is cached and preserved. If you request the same image with another size later, then the **original** image will be found, copied, resized, and returned.

Shared JPEG and PNG images can also be created from memory by using their named memory access constructor. You should release() the image when you're done with it.

Parameters

name	name of the image
W,H	desired size

Returns

the image at the requested size, or NULL if the image could not be found or generated

See also

Fl_Shared_Image::find(const char *name, int W, int H)

Fl_Shared_Image::release()

Fl_JPEG_Image::Fl_JPEG_Image(const char *name, const unsigned char *data)

Fl_PNG_Image::Fl_PNG_Image (const char *name_png, const unsigned char *buffer, int maxsize)

13.120.3.13 get() [2/2]

```
Fl_Shared_Image * Fl_Shared_Image::get (
            Fl_RGB_Image * rgb,
            int own_it = 1 )  [static]
```

Builds a shared image from a pre-existing Fl_RGB_Image.

Parameters

in	*rgb*	an Fl_RGB_Image used to build a new shared image.
in	*own↩ _it*	1 if the shared image should delete `rgb` when it is itself deleted, 0 otherwise

Version

1.3.4

13.120.3.14 image()

```
const Fl_Image * Fl_Shared_Image::image ( ) const  [inline]
```

Returns a pointer to the internal Fl_Image object.
The output is a pointer to the `internal` image ('Fl_Image' or subclass) which can be used to inspect or copy the image.

Do not try to modify the image! You can copy the image though if you want or need to change any attributes, size etc. If all you need to do is to resize the image you should use Fl_Shared_Image::copy(int, int) instead.

Note

The internal image (pointer) is protected for good reasons, e.g. to prevent access to the image so it can't be modified by user code. **DO NOT** cast away the 'const' attribute to modify the image.

User code should rarely need this method. Use with caution.

Returns

const Fl_Image∗ image, the internal Fl_Image

Since

1.4.0

13.120.3.15 images()

`Fl_Shared_Image ** Fl_Shared_Image::images ( ) [static]`

Returns the Fl_Shared_Image∗ array.

Returns

a pointer to an array of shared image pointers, sorted by name and size

See also

Fl_Shared_Image::num_images()

13.120.3.16 num_images()

`int Fl_Shared_Image::num_images ( ) [static]`

Number of shared images in their various cached sizes.

Returns

number of entries in the array

See also

Fl_Shared_Image::images()

13.120.3.17 original()

`int Fl_Shared_Image::original ( ) const [inline]`

Returns whether this is an original image.

Images loaded from a file or from memory are marked `original` as opposed to images created as a copy of another image with different size (width or height).

Note

This is useful for debugging (rarely used in user code).

Since

FLTK 1.4.0

13.120.3.18 refcount()

`int Fl_Shared_Image::refcount ( ) const [inline]`
Returns the number of references of this shared image.
When reference is below 1, the image is deleted.

13.120.3.19 release()

`void Fl_Shared_Image::release ( ) [override], [virtual]`
Releases and possibly destroys (if refcount <= 0) a shared image.
In the latter case, it will reorganize the shared image array so that no hole will occur.
Reimplemented from Fl_Image.

13.120.3.20 uncache()

`void Fl_Shared_Image::uncache ( ) [override], [virtual]`
Remove the cached device specific image data.

See also

Fl_Image::uncache()

Reimplemented from Fl_Image.

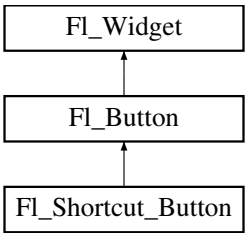
13.120.3.21 update()

`void Fl_Shared_Image::update ( ) [protected]`
Update the dimensions of the shared images.
Internal method to synchronize shared image data with the actual image data.
The documentation for this class was generated from the following files:

- Fl_Shared_Image.H
- Fl_Shared_Image.cxx

13.121 Fl_Shortcut_Button Class Reference

A button that allows the user to type a key combination to create shortcuts.
Inheritance diagram for Fl_Shortcut_Button:



Public Member Functions

- Fl_Shortcut_Button (int X, int Y, int W, int H, const char *l=0)

  Construct a shortcut button.
- Fl_Shortcut value ()

  Return the user selected shortcut.
- void value (Fl_Shortcut shortcut)

  Set the displayed shortcut.

Public Member Functions inherited from Fl_Button

- int clear ()

  Same as `value(0)`.
- uchar compact () const

  Return true if buttons are rendered as compact buttons.
- void compact (uchar v)

  Decide if buttons should be rendered in compact mode.
- Fl_Boxtype down_box () const

  Returns the current down box type, which is drawn when value() is non-zero.
- void down_box (Fl_Boxtype b)

  Sets the down box type.
- Fl_Color **down_color** () const

  (for backwards compatibility)
- void **down_color** (unsigned c)

  (for backwards compatibility)
- Fl_Button (int X, int Y, int W, int H, const char *L=0)

  The constructor creates the button using the given position, size, and label.
- int set ()

  Same as `value(1)`.
- void **setonly** ()

  Turns on this button and turns off all other radio buttons in the group (calling `value(1)` or `set()` does not do this).
- int shortcut () const

  Returns the current shortcut key for the button.
- void **shortcut** (const char *s)

  (for backwards compatibility)
- void shortcut (int s)

  Sets the shortcut key to `s`.
- char **value** () const

  Returns the current value of the button (0 or 1).
- int value (int v)

  Sets the current value of the button.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

- virtual class Fl_Gl_Window const ∗ **as_gl_window** () const
- virtual Fl_Group ∗ as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const ∗ **as_group** () const
- virtual Fl_Window ∗ as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ **as_window** () const
- void bind_deimage (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗cb, void ∗p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

Sets the background and selection color of the widget.

- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.

- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

Changes the size or position of the widget.

- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.

- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void **do_end_hot_callback** ()

  Call the callback if the user is interested.
- void draw () override

  Draw the textual representation of the shortcut button.
- int handle (int) override

  Handle keystrokes to catch the user's shortcut.

Protected Member Functions inherited from Fl_Button

- void **simulate_key_action** ()

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.

- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Protected Attributes

- Fl_Shortcut **shortcut_value**

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

flags possible values enumeration.

Static Protected Member Functions inherited from Fl_Button

- static void **key_release_timeout** (void ∗)

Static Protected Attributes inherited from Fl_Button

- static Fl_Widget_Tracker ∗ **key_release_tracker** = 0

13.121.1 Detailed Description

A button that allows the user to type a key combination to create shortcuts.
After clicked once, the button catches the following keyboard events and records the pressed keys and all modifiers. It draws a text representation of the shortcut.
The backspace key deletes the current shortcut. A second click on the button or moving focus makes the last shortcut permanent.
The Shortcut button calls the user callback after every change if FL_WHEN_CHANGED is set, and when the button is no longer recording shortcuts if FL_WHEN_RELEASE is set.

13.121.2 Constructor & Destructor Documentation

13.121.2.1 Fl_Shortcut_Button()

```
Fl_Shortcut_Button::Fl_Shortcut_Button (
            int X,
            int Y,
            int W,
            int H,
            const char * l = 0 )
```

Construct a shortcut button.

Parameters

X,Y,W,H	position and size of the button
l	label text when no shortcut is set

13.121.3 Member Function Documentation

13.121.3.1 draw()

```
void Fl_Shortcut_Button::draw (
            void ) [override], [protected], [virtual]
```

Draw the textual representation of the shortcut button.
When the button can receive shortcut key events, it's "hot". A hot button is drawn in selection color. A cold button is drawn as a regular text box containing a human readable version of the shortcut key.
Reimplemented from Fl_Button.

13.121.3.2 handle()

```
int Fl_Shortcut_Button::handle (
            int e ) [override], [protected], [virtual]
```

Handle keystrokes to catch the user's shortcut.
Reimplemented from Fl_Button.

13.121.3.3 value() [1/2]

```
Fl_Shortcut Fl_Shortcut_Button::value ( )
```

Return the user selected shortcut.

Returns

shortcut encoded as key and modifier

13.121.3.4 value() [2/2]

```
void Fl_Shortcut_Button::value (
            Fl_Shortcut shortcut )
```

Set the displayed shortcut.

Parameters

in	*shortcut*	encoded as key and modifier

The documentation for this class was generated from the following files:

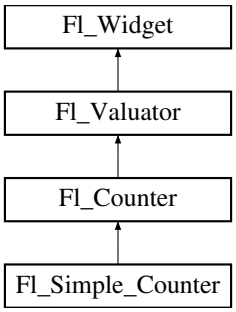
- Fl_Shortcut_Button.H
- Fl_Shortcut_Button.cxx

13.122 Fl_Simple_Counter Class Reference

This widget creates a counter with only 2 arrow buttons.

`#include <Fl_Simple_Counter.H>`

Inheritance diagram for Fl_Simple_Counter:



Public Member Functions

- **Fl_Simple_Counter** (int X, int Y, int W, int H, const char ∗L=0)

Public Member Functions inherited from Fl_Counter

- Fl_Counter (int X, int Y, int W, int H, const char ∗L=0)

  Creates a new Fl_Counter widget using the given position, size, and label string.
- int handle (int) override

  Handles the specified event.
- void lstep (double a)

  Sets the increment for the large step buttons.
- double **step** () const

  Returns the increment for normal step buttons.
- void step (double a)

  Sets the increment for the normal step buttons.
- void step (double a, double b)

  Sets the increments for the normal and large step buttons.

- Fl_Color **textcolor** () const

  Gets the font color.
- void **textcolor** (Fl_Color s)

  Sets the font color to `s`.
- Fl_Font **textfont** () const

  Gets the text font.
- void **textfont** (Fl_Font s)

  Sets the text font to `s`.
- Fl_Fontsize **textsize** () const

  Gets the font size.
- void **textsize** (Fl_Fontsize s)

  Sets the font size to `s`.
- ~**Fl_Counter** ()

  Destroys the valuator.

Public Member Functions inherited from Fl_Valuator

- void **bounds** (double a, double b)

  Sets the minimum (a) and maximum (b) values for the valuator widget.
- double **clamp** (double)

  Clamps the passed value to the valuator range.
- virtual int format (char *)

  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.
- virtual std::string format_str ()

  C++11 API for Fl_Valuator::format(char buffer).*
- double increment (double, int)

  Adds n times the step value to the passed value.
- double maximum () const

  Gets the maximum value for the valuator.
- void maximum (double a)

  Sets the maximum value for the valuator.
- double minimum () const

  Gets the minimum value for the valuator.
- void minimum (double a)

  Sets the minimum value for the valuator.
- void precision (int digits)

  Sets the step value to 1.0 / 10^{digits}.
- void range (double a, double b)

  Sets the minimum and maximum values for the valuator.
- double round (double)

  Round the passed value to the nearest step increment.
- double step () const

  Gets or sets the step value.
- void **step** (double a, int b)

  See double Fl_Valuator::step() const

- void **step** (double s)

  See double Fl_Valuator::step() const.
- void **step** (int a)

  See double Fl_Valuator::step() const

- double value () const

  Gets the floating point(double) value.
- int value (double)

  Sets the current value.
- ~**Fl_Valuator** () override

  Destructor is accessible despite protected constructor.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

Sets the current callback function and managed user data for the widget.

- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.

- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

Gets the font size in pixels.

- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.

- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Counter

- void arrow_widths (int &w1, int &w2)

  Compute sizes (widths) of arrow boxes.
- void draw () override

  Draws the widget.

Protected Member Functions inherited from Fl_Valuator

- Fl_Valuator (int X, int Y, int W, int H, const char *L)

  Creates a new Fl_Valuator widget using the given position, size, and label string.
- void **handle_drag** (double newvalue)

  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.
- void **handle_push** ()

  Stores the current value in the previous value.
- void **handle_release** ()

  Called after an FL_WHEN_RELEASE event is received and before the callback.
- int **horizontal** () const

  Tells if the valuator is an FL_HORIZONTAL one.
- double **previous_value** () const

  Gets the previous floating point value before an event changed it.
- void **set_value** (double v)

  Sets the current floating point value.
- double **softclamp** (double)

  Clamps the value, but accepts v if the previous value is not already out of range.
- virtual void value_damage ()

  Asks for partial redraw.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

13.122.1 Detailed Description

This widget creates a counter with only 2 arrow buttons.

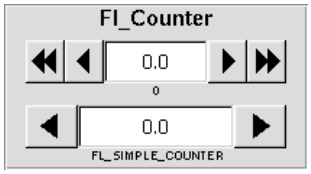


Figure 13.43 Fl_Simple_Counter

The documentation for this class was generated from the following files:

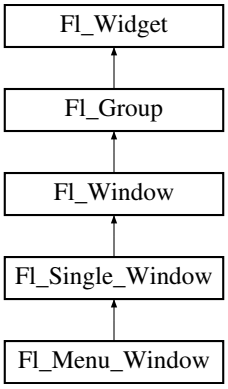
- Fl_Simple_Counter.H
- Fl_Counter.cxx

13.123 Fl_Single_Window Class Reference

This is the same as Fl_Window.

`#include <Fl_Single_Window.H>`

Inheritance diagram for Fl_Single_Window:



Public Member Functions

- **Fl_Single_Window** (int W, int H, const char ∗l=0)

  Creates a new Fl_Single_Window widget using the given size, and label (title) string.
- **Fl_Single_Window** (int X, int Y, int W, int H, const char ∗l=0)

  Creates a new Fl_Single_Window widget using the given position, size, and label (title) string.
- void flush () override

  Same as Fl_Window::flush()
- void **make_current** ()

  Same as Fl_Window::make_current()
- void show () override

  Makes a widget visible.
- void **show** (int argc, char ∗∗argv)

  Same as Fl_Window::show(int argc, char ∗∗argv)

Public Member Functions inherited from Fl_Window

- void allow_expand_outside_parent ()

  Allow this subwindow to expand outside the area of its parent window.
- virtual class Fl_Double_Window ∗ as_double_window ()

  Return non-null if this is an Fl_Double_Window object.
- virtual class Fl_Overlay_Window ∗ as_overlay_window ()

  Return non-null if this is an Fl_Overlay_Window object.
- Fl_Window const ∗ as_window () const override
- Fl_Window ∗ as_window () override

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- unsigned int **border** () const

  Returns whether the window possesses a border.
- void border (int b)

  Sets whether or not the window manager border is around the window.
- void clear_border ()

  Fast inline function to turn the window manager border off.
- void clear_modal_states ()

  Clears the "modal" flags and converts a "modal" or "non-modal" window back into a "normal" window.

- void **copy_label** (const char *a)

  Sets the window titlebar label to a copy of a character string.
- void cursor (const Fl_RGB_Image *, int, int)

  Changes the cursor for this window using the provided image as cursor's shape.
- void cursor (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)

  For back compatibility only.
- void cursor (Fl_Cursor)

  Changes the cursor for this window.
- int decorated_h () const

  Returns the window height including any window title bar and any frame added by the window manager.
- int decorated_w () const

  Returns the window width including any frame added by the window manager.
- void default_cursor (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)

  For back compatibility only.
- void default_cursor (Fl_Cursor)

  Sets the default window cursor.
- void **draw_backdrop** ()

  Draw the background image if one is set and is aligned inside.
- Fl_Window (int w, int h, const char *title=0)

  Creates a window from the given width `w`, height `h`, and `title`.
- Fl_Window (int x, int y, int w, int h, const char *title=0)

  Creates a window from the given position (x, y), size (w, h) and title.
- void free_position ()

  Undoes the effect of a previous resize() or show() so that the next time show() is called the window manager is free to position the window.
- void fullscreen ()

  Makes the window completely fill one or more screens, without any window manager border visible.
- unsigned int **fullscreen_active** () const

  Returns non zero if FULLSCREEN flag is set, 0 otherwise.
- void **fullscreen_off** ()

  Turns off any side effects of fullscreen()
- void **fullscreen_off** (int X, int Y, int W, int H)

  Turns off any side effects of fullscreen() and does resize(X,Y,W,H).
- void fullscreen_screens (int top, int bottom, int left, int right)

  Sets which screens should be used when this window is in fullscreen mode.
- uchar get_size_range (int *minw, int *minh, int *maxw=NULL, int *maxh=NULL, int *dw=NULL, int *dh=NULL, int *aspect=NULL) const

  Gets the allowable range to which the user can resize this window.
- int handle (int) override

  Handles the specified event.
- void hide () override

  Removes the window from the screen.
- void **hotspot** (const Fl_Widget &p, int offscreen=0)

  See void Fl_Window::hotspot(int x, int y, int offscreen = 0)
- void **hotspot** (const Fl_Widget *, int offscreen=0)

  See void Fl_Window::hotspot(int x, int y, int offscreen = 0)
- void hotspot (int x, int y, int offscreen=0)

  Positions the window so that the mouse is pointing at the given position, or at the center of the given widget, which may be the window itself.
- const void * icon () const

  Gets the current icon window target dependent data.

- void icon (const Fl_RGB_Image ∗)

  Sets or resets a single window icon.
- void icon (const void ∗ic)

  Platform-specific method to set the window icon usable on Windows and X11 only.
- void iconize ()

  Iconifies the window.
- const char ∗ **iconlabel** () const

  See void Fl_Window::iconlabel(const char∗)
- void **iconlabel** (const char ∗)

  Sets the icon label.
- void icons (const Fl_RGB_Image ∗[ ], int)

  Sets the window icons.
- void icons (HICON big_icon, HICON small_icon)

  Sets the window icons using HICON handles (Windows platform only).
- const char ∗ **label** () const

  See void Fl_Window::label(const char∗)
- void **label** (const char ∗)

  Sets the window title bar label.
- void **label** (const char ∗label, const char ∗iconlabel)

  Sets the icon label.
- void make_current ()

  Sets things up so that the drawing functions in <FL/fl_draw.H> will go into this window.
- void maximize ()

  Maximizes a top-level window to its current screen.
- unsigned int **maximize_active** () const

  Returns whether the window is currently maximized.
- unsigned int **menu_window** () const

  Returns true if this window is a menu window.
- unsigned int modal () const

  Returns true if this window is modal.
- unsigned int **non_modal** () const

  Returns true if this window is modal or non-modal.
- fl_uintptr_t os_id () const

  Returns a platform-specific identification of a shown window, or 0 if not shown.
- unsigned int **override** () const

  Returns non zero if OVERRIDE flag is set, 0 otherwise.
- void resize (int X, int Y, int W, int H) override

  Changes the size and position of the window.
- int screen_num () const

  The number of the screen containing the mapped window.
- void screen_num (int screen_num)

  Set the number of the screen where to map the window.
- void set_menu_window ()

  Marks the window as a menu window.
- void set_modal ()

  A "modal" window, when shown(), will prevent any events from being delivered to other windows in the same program, and will also remain on top of the other windows (if the X window manager supports the "transient for" property).
- void set_non_modal ()

A "non-modal" window (terminology borrowed from Microsoft Windows) acts like a modal() one in that it remains on top, but it has no effect on event delivery.

- void **set_override** ()

  Activates the flags NOBORDER|OVERRIDE.
- void set_tooltip_window ()

  Marks the window as a tooltip window.
- const Fl_Image ∗ shape () const

  Returns the image controlling the window shape or NULL.
- void shape (const Fl_Image &b)

  Set the window's shape with an Fl_Image.
- void shape (const Fl_Image ∗img)

  Assigns a non-rectangular shape to the window.
- void show () override

  Puts the window on the screen.
- void show (int argc, char ∗∗argv)

  Puts the window on the screen with show() and parses command-line arguments.
- int shown () const

  Returns non-zero if show() has been called (but not hide()).
- void size_range (int minw, int minh, int maxw=0, int maxh=0, int dw=0, int dh=0, int aspect=0)

  Sets the allowable range to which the user can resize this window.
- unsigned int **tooltip_window** () const

  Returns true if this window is a tooltip window.
- void un_maximize ()

  Returns a previously maximized top-level window to its previous size.
- void wait_for_expose ()

  Waits for the window to be displayed after calling show().
- int **x_root** () const

  Gets the x position of the window on the screen.
- const char ∗ xclass () const

  Returns the xclass for this window, or a default.
- void xclass (const char ∗c)

  Sets the xclass for this window.
- int **y_root** () const

  Gets the y position of the window on the screen.
- virtual ∼Fl_Window ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Group

- Fl_Widget ∗& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget ∗o)

  See void Fl_Group::add(Fl_Widget &w)
- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget ∗const ∗ array () const

  Returns a pointer to the internal array of children.
- Fl_Group const ∗ as_group () const override
- Fl_Group ∗ as_group () override

Returns an Fl_Group pointer if this widget is an Fl_Group.

- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const

  Returns the n'th child.
- int **children** () const

  Returns how many child widgets the group has.
- void clear ()

  Deletes all child widgets from memory recursively.
- unsigned int clip_children () const

  Returns the current clipping mode.
- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)

  Removes the widget at `index` from the group and deletes it.
- void end ()

  Exactly the same as current(this->parent()).
- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*
- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.
- void **forms_end** ()

  This is for forms compatibility only.
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

  This does insert(w, find(before)).
- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.
- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.
- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ **as_gl_window** () const
- void bind_deimage (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗cb, void ∗p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.

- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- int horizontal_label_margin ()

Get the spacing between the label and the horizontal edge of the widget.

- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.

- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.

- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Public Types inherited from Fl_Window

- typedef struct HICON__ ∗ **HICON**

Static Public Member Functions inherited from Fl_Window

- static Fl_Window ∗ current ()

  Returns the last window that was made current.
- static void **default_callback** (Fl_Window ∗, void ∗v)

  Back compatibility: Sets the default callback v for win to call on close event.
- static void default_icon (const Fl_RGB_Image ∗)

  Sets a single default window icon.
- static void default_icons (const Fl_RGB_Image ∗[ ], int)

  Sets the default window icons.
- static void default_icons (HICON big_icon, HICON small_icon)

  Sets the default window icons (Windows platform only).

- static const char * default_xclass ()

  Returns the default xclass.
- static void default_xclass (const char *)

  Sets the default window xclass.
- static bool is_a_rescale ()

  Returns true when a window is being rescaled.
- static char show_next_window_iconic ()

  Returns the static flag whether the next window should be opened iconified.
- static void show_next_window_iconic (char stat)

  Sets a static flag whether the next window should be opened iconified.

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()

  Returns the currently active group.
- static void current (Fl_Group *g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Window

- void default_size_range ()

  Protected method to calculate the default size range of a window.
- void draw () override

  Draws the widget.
- int force_position () const

  Returns the internal state of the window's `FORCE_POSITION` flag.
- void force_position (int force)

Sets an internal flag that tells FLTK and the window manager to honor position requests.

- void free_icons ()

  Deletes all icons previously attached to the window.
- int is_resizable ()

  Protected method to determine whether a window is resizable.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()

  Returns the internal array of widget sizes and positions.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget *, int)

  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)

  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)

  Allow derived groups to act when a child widget is removed from the group.
- int * sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

Gets the widget flags mask.

- void h (int v)

  Internal use only.

- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.

- void w (int v)

  Internal use only.

- void x (int v)

  Internal use only.

- void y (int v)

  Internal use only.

Static Protected Attributes inherited from Fl_Window

- static Fl_Window ∗ current_

  Stores the last window that was made current.

13.123.1 Detailed Description

This is the same as Fl_Window.
However, it is possible that some implementations will provide double-buffered windows by default. This subclass can be used to force single-buffering. This may be useful for modifying existing programs that use incremental update, or for some types of image data, such as a movie flipbook.

13.123.2 Member Function Documentation

13.123.2.1 flush()

`void Fl_Single_Window::flush ( ) [inline], [override], [virtual]`
Same as Fl_Window::flush()
Reimplemented from Fl_Window.

13.123.2.2 show()

`void Fl_Single_Window::show ( ) [override], [virtual]`
Makes a widget visible.
An invisible widget never gets redrawn and does not get keyboard or mouse events, but can receive a few other events like FL_SHOW.
The visible() method returns true if the widget is set to be visible. The visible_r() method returns true if the widget and all of its parents are visible. A widget is only visible if visible() is true on it *and all of its parents*.
Changing it will send FL_SHOW or FL_HIDE events to the widget. *Do not change it if the parent is not visible, as this will send false FL_SHOW or FL_HIDE events to the widget.* redraw() is called if necessary on this or the parent.

See also

hide(), visible(), visible_r()

Reimplemented from Fl_Widget.
The documentation for this class was generated from the following files:

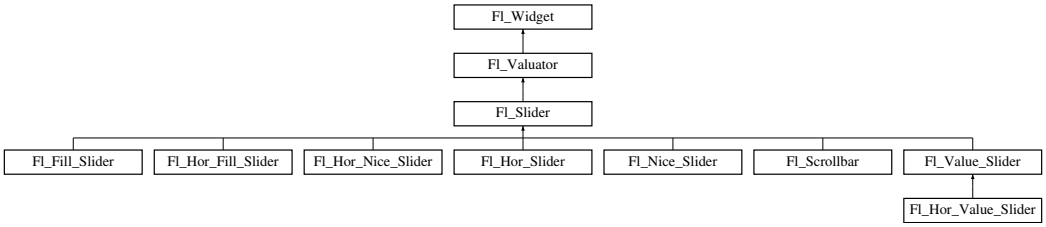
- Fl_Single_Window.H
- Fl_Single_Window.cxx

13.124 Fl_Slider Class Reference

The Fl_Slider widget contains a sliding knob inside a box.

`#include <Fl_Slider.H>`

Inheritance diagram for Fl_Slider:



Public Types

- enum Scale_Type : uchar { LINEAR_SCALE = 0 , LOG_SCALE }

  Bitset for tick mark positions for Fl_Nice_Slider.
- enum Tick_Position : uchar {
  TICKS_NONE = 0x00 , TICKS_BELOW = 0x01 , TICKS_ABOVE = 0x02 , TICKS_LEFT = 0x01 , TICKS_RIGHT = 0x02 }

  Bitset for tick mark positions for Fl_Nice_Slider.

Public Member Functions

- void bounds (double a, double b)

  Sets the minimum (a) and maximum (b) values for the valuator widget.
- Fl_Slider (int X, int Y, int W, int H, const char ∗L=0)

  Creates a new Fl_Slider widget using the given position, size, and label string.
- **Fl_Slider** (uchar t, int X, int Y, int W, int H, const char ∗L)

  Creates a new Fl_Slider widget using the given type, position, size, and label string.
- int handle (int) override

  Handles the specified event.
- uchar num_ticks () const

  Gets the number of tick marks.
- Scale_Type scale () const

  Gets the scale type.
- void scale (Scale_Type s)

  Sets the scale type.
- int scrollvalue (int pos, int size, int first, int total)

  Sets the size and position of the sliding knob in the box.
- Fl_Boxtype **slider** () const

  Gets the slider box type.
- void **slider** (Fl_Boxtype c)

  Sets the slider box type.
- float **slider_size** () const

  Get the dimensions of the moving piece of slider.
- void slider_size (double v)

  Set the dimensions of the moving piece of slider.
- uchar ticks () const

  Gets the tick mark position.
- void ticks (uchar t, uchar num_ticks=11)

  Sets the tick mark position for Fl_Nice_Slider.

Public Member Functions inherited from Fl_Valuator

- void **bounds** (double a, double b)

  Sets the minimum (a) and maximum (b) values for the valuator widget.
- double **clamp** (double)

  Clamps the passed value to the valuator range.
- virtual int format (char ∗)

  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.
- virtual std::string format_str ()

  C++11 API for Fl_Valuator::format(char∗ buffer).
- double increment (double, int)

  Adds n times the step value to the passed value.
- double maximum () const

  Gets the maximum value for the valuator.
- void maximum (double a)

  Sets the maximum value for the valuator.
- double minimum () const

  Gets the minimum value for the valuator.
- void minimum (double a)

  Sets the minimum value for the valuator.
- void precision (int digits)

  Sets the step value to 1.0 / 10^{digits}.
- void range (double a, double b)

  Sets the minimum and maximum values for the valuator.
- double round (double)

  Round the passed value to the nearest step increment.
- double step () const

  Gets or sets the step value.
- void **step** (double a, int b)

  See double Fl_Valuator::step() const
- void **step** (double s)

  See double Fl_Valuator::step() const.
- void **step** (int a)

  See double Fl_Valuator::step() const
- double value () const

  Gets the floating point(double) value.
- int value (double)

  Sets the current value.
- ∼**Fl_Valuator** () override

  Destructor is accessible despite protected constructor.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

Returns whether the widget and all of its parents are active.

- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ **as_gl_window** () const
- virtual Fl_Group ∗ as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const ∗ **as_group** () const
- virtual Fl_Window ∗ as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ **as_window** () const
- void bind_deimage (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗cb, void ∗p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

Sets a widget to accept input.

- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.

- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

Returns a pointer to the parent widget.

- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.

- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void draw () override

  Draws the widget.
- void **draw** (int, int, int, int)
- void draw_ticks (const Fl_Rect &r, int S)

  Draw tick marks.
- int **handle** (int, int, int, int, int)
- double increment_lin_log (double v, int n, int range)

  Adds a sensible step value to the passed value.
- double position_to_value (double pos) const

  Converts a normalized position (0.0 to 1.0) to a value based on the current scale type.
- double value_to_position (double val) const

  Converts a value to a normalized position (0.0 to 1.0) based on the current scale type.

Protected Member Functions inherited from Fl_Valuator

- Fl_Valuator (int X, int Y, int W, int H, const char *L)

  Creates a new Fl_Valuator widget using the given position, size, and label string.
- void **handle_drag** (double newvalue)

  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.
- void **handle_push** ()

  Stores the current value in the previous value.
- void **handle_release** ()

  Called after an FL_WHEN_RELEASE event is received and before the callback.
- int **horizontal** () const

  Tells if the valuator is an FL_HORIZONTAL one.
- double **previous_value** () const

  Gets the previous floating point value before an event changed it.
- void **set_value** (double v)

  Sets the current floating point value.
- double **softclamp** (double)

  Clamps the value, but accepts v if the previous value is not already out of range.
- virtual void value_damage ()

  Asks for partial redraw.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

Internal use only.

- void x (int v)

  Internal use only.

- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.

- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.

- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.124.1 Detailed Description

The Fl_Slider widget contains a sliding knob inside a box.
It is often used as a scrollbar. Moving the box all the way to the top/left sets it to the minimum(), and to the bottom/right to the maximum(). The minimum() may be greater than the maximum() to reverse the slider direction.
Use void Fl_Widget::type(int) to set how the slider is drawn, which can be one of the following:

- FL_VERTICAL - Draws a vertical slider (this is the default).
- FL_HORIZONTAL - Draws a horizontal slider.
- FL_VERT_FILL_SLIDER - Draws a filled vertical slider, useful as a progress or value meter.
- FL_HOR_FILL_SLIDER - Draws a filled horizontal slider, useful as a progress or value meter.
- FL_VERT_NICE_SLIDER - Draws a vertical slider with a nice looking control knob.
- FL_HOR_NICE_SLIDER - Draws a horizontal slider with a nice looking control knob.

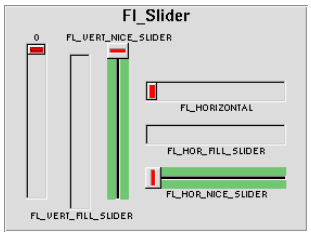


Figure 13.44 Fl_Slider

13.124.2 Member Enumeration Documentation

13.124.2.1 Scale_Type

`enum Fl_Slider::Scale_Type : uchar`

Bitset for tick mark positions for Fl_Nice_Slider.

See also

Fl_Slider::ticks(), Fl_Slider::ticks(Tick_Position)

Enumerator

LINEAR_SCALE	Linear scale (default)
LOG_SCALE	Logarithmic scale.

13.124.2.2 Tick_Position

`enum Fl_Slider::Tick_Position : uchar`

Bitset for tick mark positions for Fl_Nice_Slider.

See also

Fl_Slider::ticks(), Fl_Slider::ticks(Tick_Position)

Enumerator

TICKS_NONE	draw no ticks
TICKS_BELOW	draw ticks below a horizontal slider
TICKS_ABOVE	draw ticks above a horizontal slider
TICKS_LEFT	draw ticks to the left of a vertical slider
TICKS_RIGHT	draw ticks to the right of a vertical slider

13.124.3 Constructor & Destructor Documentation

13.124.3.1 Fl_Slider()

```
Fl_Slider::Fl_Slider (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

Creates a new Fl_Slider widget using the given position, size, and label string.
The default boxtype is FL_DOWN_BOX.

13.124.4 Member Function Documentation

13.124.4.1 bounds()

```
void Fl_Slider::bounds (
            double a,
            double b )
```

Sets the minimum (a) and maximum (b) values for the valuator widget.
if at least one of the values is changed, a partial redraw is asked.

13.124.4.2 draw()

```
void Fl_Slider::draw ( ) [override], [protected], [virtual]
```

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();                 // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.
Reimplemented in Fl_Value_Slider.

13.124.4.3 draw_ticks()

```
void Fl_Slider::draw_ticks (
            const Fl_Rect & r,
            int S ) [protected]
```

Draw tick marks.
Ported from fltk2's Slider::draw_ticks(): tick marks are placed at "nice" round-number intervals (multiples of 1, 2, or 5 times a power of ten) that are never closer together on screen than about half the slider size, and every few ticks is drawn longer and labeled with its value.

Parameters

in	*r*	motion range of the slider
in	*S*	size of the slider, horizontal or vertical, in pixels

13.124.4.4 handle()

```
int Fl_Slider::handle (
            int event ) [override], [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to *override* the behavior of the base class. This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to *filter* event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you *consumed* the event by returning 1 even if you didn't *do* anything with it.

Parameters

in	*event*	the kind of event received

Return values

0	if the event was not used or understood
1	if the event was used and can be deleted

See also

Fl_Event

Reimplemented from Fl_Widget.
Reimplemented in Fl_Value_Slider.

13.124.4.5 increment_lin_log()

```
double Fl_Slider::increment_lin_log (
            double v,
            int n,
            int range )  [protected]
```

Adds a sensible step value to the passed value.
The step size is not useful in logarithmic scales.

13.124.4.6 num_ticks()

```
uchar Fl_Slider::num_ticks ( ) const  [inline]
```

Gets the number of tick marks.

Returns

number of tick marks

13.124.4.7 position_to_value()

```
double Fl_Slider::position_to_value (
            double pos ) const  [protected]
```

Converts a normalized position (0.0 to 1.0) to a value based on the current scale type.
For linear scale, this is a simple linear interpolation. For logarithmic scale, this uses exponential interpolation.

Parameters

in	*pos*	The normalized position to convert.

Returns

The corresponding value.

13.124.4.8 scale() [1/2]

```
Scale_Type Fl_Slider::scale ( ) const  [inline]
```

Gets the scale type.

Returns

scale type

13.124.4.9 scale() [2/2]

```
void Fl_Slider::scale (
            Scale_Type s )  [inline]
```

Sets the scale type.

Parameters

in	*s*	scale type

13.124.4.10 scrollvalue()

```
int Fl_Slider::scrollvalue (
            int pos,
            int size,
            int first,
            int total )
```

Sets the size and position of the sliding knob in the box.

Parameters

in	*pos*	position of first line displayed
in	*size*	size of window in lines
in	*first*	number of first line
in	*total*	total number of lines Returns Fl_Valuator::value(p)

13.124.4.11 slider_size()

```
void Fl_Slider::slider_size (
            double v )
```

Set the dimensions of the moving piece of slider.
This is the fraction of the size of the entire widget. If you set this to 1 then the slider cannot move. The default value is .08.
For the "fill" sliders this is the size of the area around the end that causes a drag effect rather than causing the slider to jump to the mouse.

13.124.4.12 ticks() [1/2]

```
uchar Fl_Slider::ticks ( ) const  [inline]
```

Gets the tick mark position.

Returns

tick mark position bitset

13.124.4.13 ticks() [2/2]

```
void Fl_Slider::ticks (
            uchar t,
            uchar num_ticks = 11 )  [inline]
```

Sets the tick mark position for Fl_Nice_Slider.
FL_TICKS_BELOW and FL_TICKS_ABOVE can be or'd together for horizontal sliders, and FL_TICKS_LEFT and FL_TICKS_RIGHT for vertical sliders.

Parameters

in	*t*	tick mark position bitset
in	*num_ticks*	number of tick marks to draw

13.124.4.14 value_to_position()

```
double Fl_Slider::value_to_position (
            double val ) const  [protected]
```

Converts a value to a normalized position (0.0 to 1.0) based on the current scale type.
For linear scale, this is a simple linear interpolation. For logarithmic scale, this uses logarithmic interpolation.

Parameters

in	*val*	The value to convert.

Returns

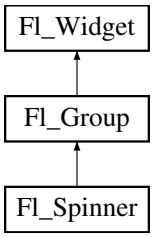
A normalized position between 0.0 and 1.0.

The documentation for this class was generated from the following files:

- Fl_Slider.H
- Fl_Slider.cxx

13.125 Fl_Spinner Class Reference

This widget is a combination of a numerical input widget and repeat buttons.
`#include <Fl_Spinner.H>`
Inheritance diagram for Fl_Spinner:



Classes

- class Fl_Spinner_Input

Public Member Functions

- Fl_Color **color** () const

  Returns the background color of the spinner widget's input field.
- void **color** (Fl_Color v)

  Sets the background color of the spinner widget's input field.
- Fl_Spinner (int X, int Y, int W, int H, const char ∗L=0)

  Creates a new Fl_Spinner widget using the given position, size, and label string.
- const char ∗ **format** () const

  Returns the format string for the value.
- void **format** (const char ∗f)

  Sets the format string for the value.
- int handle (int event) override

  Handles the specified event.
- double **maximum** () const

  Gets the maximum value of the widget.
- void **maximum** (double m)

Sets the maximum value of the widget.

- int **maximum_size** () const

  Returns the maximum width of the input field.

- void **maximum_size** (int m)

  Sets the maximum width of the input field.

- double **minimum** () const

  Gets the minimum value of the widget.

- void **minimum** (double m)

  Sets the minimum value of the widget.

- void **range** (double a, double b)

  Sets the minimum and maximum values for the widget.

- void resize (int X, int Y, int W, int H) override

  Resizes the Fl_Group widget and all of its children.

- Fl_Color **selection_color** () const

  Returns the selection color of the spinner widget's input field.

- void **selection_color** (Fl_Color val)

  Sets the selection color of the spinner widget's input field.

- double step () const

  Gets the amount to change the value when the user clicks a button.

- void step (double s)

  Sets or returns the amount to change the value when the user clicks a button.

- Fl_Color **textcolor** () const

  Gets the color of the text in the input field.

- void **textcolor** (Fl_Color c)

  Sets the color of the text in the input field.

- Fl_Font **textfont** () const

  Gets the font of the text in the input field.

- void **textfont** (Fl_Font f)

  Sets the font of the text in the input field.

- Fl_Fontsize **textsize** () const

  Gets the size of the text in the input field.

- void **textsize** (Fl_Fontsize s)

  Sets the size of the text in the input field.

- uchar type () const

  Gets the numeric representation in the input field.

- void type (uchar v)

  Sets the numeric representation in the input field.

- double **value** () const

  Gets the current value of the widget.

- void value (double v)

  Sets the current value of the input widget.

- int wrap () const

  Gets the wrap mode of the Fl_Spinner widget.

- void wrap (int set)

  Sets whether the spinner wraps around at upper and lower bounds.

Public Member Functions inherited from Fl_Group

- Fl_Widget *& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget *o)

  See void Fl_Group::add(Fl_Widget &w)
- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget *const * array () const

  Returns a pointer to the internal array of children.
- Fl_Group const * as_group () const override
- Fl_Group * as_group () override

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const

  Returns the n'th child.
- int **children** () const

  Returns how many child widgets the group has.
- void clear ()

  Deletes all child widgets from memory recursively.
- unsigned int clip_children () const

  Returns the current clipping mode.
- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)

  Removes the widget at `index` from the group and deletes it.
- void end ()

  Exactly the same as current(this->parent()).
- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*
- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.
- void **forms_end** ()

  This is for forms compatibility only.
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

  This does insert(w, find(before)).
- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.
- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.

- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.

- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

Returns whether the inactive image is managed by the widget.

- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.

- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

Returns if the widget is able to take events.

- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void draw () override

  Draws the widget.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()

  Returns the internal array of widget sizes and positions.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget *, int)

  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)

  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)

  Allow derived groups to act when a child widget is removed from the group.
- int * sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.

- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Protected Attributes

- Fl_Repeat_Button **down_button_**
- Fl_Spinner_Input **input_**
- Fl_Repeat_Button **up_button_**

Additional Inherited Members

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()

  Returns the currently active group.
- static void current (Fl_Group *g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.125.1 Detailed Description

This widget is a combination of a numerical input widget and repeat buttons.
The user can either type into the input area or use the buttons to change the value.

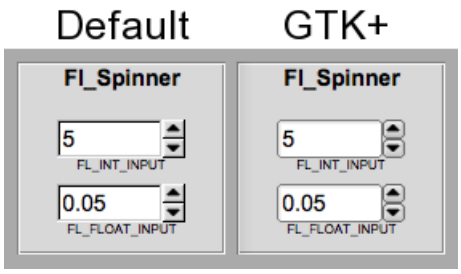


Figure 13.45 Fl_Spinner widget

13.125.2 Constructor & Destructor Documentation

13.125.2.1 Fl_Spinner()

```
Fl_Spinner::Fl_Spinner (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

Creates a new Fl_Spinner widget using the given position, size, and label string.
The inherited destructor destroys the widget and any value associated with it.

13.125.3 Member Function Documentation

13.125.3.1 draw()

```
void Fl_Spinner::draw ( )  [override], [protected], [virtual]
```

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();                 // calls Fl_Scrollbar::draw()
```

Reimplemented from Fl_Group.

13.125.3.2 handle()

```
int Fl_Spinner::handle (
            int event )  [override], [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to *override* the behavior of the base class. This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to *filter* event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you *consumed* the event by returning 1 even if you didn't *do* anything with it.

Parameters

in	*event*	the kind of event received

Return values

0	if the event was not used or understood
1	if the event was used and can be deleted

See also

FI_Event

Reimplemented from FI_Group.

13.125.3.3 resize()

```
void Fl_Spinner::resize (
            int X,
            int Y,
            int W,
            int H )  [override], [virtual]
```

Resizes the FI_Group widget and all of its children.
The FI_Group widget first resizes itself, and then it moves and resizes all its children according to the rules documented for FI_Group::resizable(FI_Widget∗)

See also

FI_Group::resizable(FI_Widget∗)

FI_Group::resizable()

FI_Widget::resize(int,int,int,int)

Reimplemented from FI_Group.

13.125.3.4 step() [1/2]

```
double Fl_Spinner::step ( ) const  [inline]
```

Gets the amount to change the value when the user clicks a button.

See also

FI_Spinner::step(double)

13.125.3.5 step() [2/2]

```
void Fl_Spinner::step (
            double s )
```

Sets or returns the amount to change the value when the user clicks a button.
Before setting step to a non-integer value, the spinner type() should be changed to floating point.

See also

double FI_Spinner::step() const

13.125.3.6 type() [1/2]

```
uchar Fl_Spinner::type ( ) const  [inline]
```

Gets the numeric representation in the input field.

See also

FI_Spinner::type(uchar)

13.125.3.7 type() [2/2]

```
void Fl_Spinner::type (
            uchar v )
```

Sets the numeric representation in the input field.
Valid values are FL_INT_INPUT and FL_FLOAT_INPUT. Also changes the format() template. Setting a new spinner type via a superclass pointer will not work.

Note

type() is not a virtual function.

13.125.3.8 value()

```
void Fl_Spinner::value (
            double v )  [inline]
```

Sets the current value of the input widget.
Before setting value to a non-integer value, the spinner type() should be changed to floating point.

13.125.3.9 wrap() [1/2]

```
int Fl_Spinner::wrap ( ) const  [inline]
```

Gets the wrap mode of the Fl_Spinner widget.

See also

void wrap(int)

Since

1.4.0

13.125.3.10 wrap() [2/2]

```
void Fl_Spinner::wrap (
            int set )  [inline]
```

Sets whether the spinner wraps around at upper and lower bounds.
If wrap mode is on the spinner value is set to the minimum() or maximum() if the value exceeds the upper or lower bounds, resp., if it was changed by one of the buttons or the FL_Up or FL_Down keys.
The spinner stops at the upper and lower bounds if wrap mode is off.
The default wrap mode is on for backwards compatibility with FLTK 1.3.x and older versions.

Note

Wrap mode does not apply to the input field if the input value is edited directly as a number. The input value is always clipped to the allowed range as if wrap mode was off when the input field is left (i.e. loses focus).

See also

minimum(), maximum()

Parameters

in	*set*	non-zero sets wrap mode, zero resets wrap mode

Since

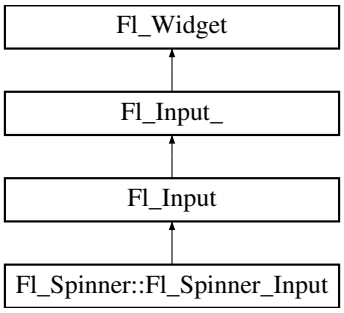
1.4.0

The documentation for this class was generated from the following files:

- Fl_Spinner.H
- Fl_Spinner.cxx

13.126 Fl_Spinner::Fl_Spinner_Input Class Reference

Inheritance diagram for Fl_Spinner::Fl_Spinner_Input:



Public Member Functions

- **Fl_Spinner_Input** (int X, int Y, int W, int H)
- int handle (int event) override

  Handles events of Fl_Spinner's embedded input widget.

Public Member Functions inherited from Fl_Input

- Fl_Input (int, int, int, int, const char *=0)

  Creates a new Fl_Input widget using the given position, size, and label string.

Public Member Functions inherited from Fl_Input_

- int append (const char *t, int l=0, char keep_selection=0)

  Append text at the end.
- bool can_redo () const

  Check if there is a redo action available.
- bool can_undo () const

  Check if the last operation can be undone.
- int copy (int clipboard)

  Put the current selection into the clipboard.
- int copy_cuts ()

  Copies the yank buffer to the clipboard.
- Fl_Color cursor_color () const

  Gets the color of the cursor.
- void cursor_color (Fl_Color n)

  Sets the color of the cursor.
- int cut ()

  Deletes the current selection.
- int cut (int a, int b)

  Deletes all characters between index `a` and `b`.
- int cut (int n)

  Deletes the next `n` bytes rounded to characters before or after the cursor.
- double dvalue () const

  Returns the widget text interpreted as a floating point number.

- Fl_Input_ (int, int, int, int, const char *=0)

  Creates a new Fl_Input_ widget.
- unsigned int index (int i) const

  Returns the character at index `i`.
- int input_type () const

  Gets the input field type.
- void input_type (int t)

  Sets the input field type.
- int insert (const char *t, int l=0)

  Inserts text at the cursor position.
- int insert_position () const

  Gets the position of the text cursor.
- int insert_position (int p)

  Sets the cursor position and mark.
- int insert_position (int p, int m)

  Sets the index for the cursor and mark.
- int ivalue () const

  Returns the widget text interpreted as a signed integer.
- int mark () const

  Gets the current selection mark.
- int mark (int m)

  Sets the current selection mark.
- int maximum_size () const

  Gets the maximum length of the input field in characters.
- void maximum_size (int m)

  Sets the maximum length of the input field in characters.
- const char * placeholder () const

  Get the placeholder text.
- void placeholder (const char *text)

  Set the placeholder text.
- int position () const
- int position (int p)
- int position (int p, int m)
- int readonly () const

  Gets the read-only state of the input field.
- void readonly (int b)

  Sets the read-only state of the input field.
- int redo ()

  Redo previous undo operation.
- int replace (int b, int e, const char *text, int ilen=0)

  Deletes text from `b` to `e` and inserts the new string `text`.
- void resize (int, int, int, int) override

  Changes the size of the widget.
- int shortcut () const

  Return the shortcut key associated with this widget.
- void shortcut (int s)

  Sets the shortcut key associated with this widget.
- int size () const

  Returns the number of bytes in value().
- void size (int W, int H)

  Sets the width and height of this widget.

- int static_value (const char ∗)

  Changes the widget text.
- int static_value (const char ∗, int)

  Changes the widget text.
- int tab_nav () const

  Gets whether the Tab key causes focus navigation in multiline input fields or not.
- void tab_nav (int val)

  Sets whether the Tab key does focus navigation, or inserts tab characters into Fl_Multiline_Input.
- Fl_Color textcolor () const

  Gets the color of the text in the input field.
- void textcolor (Fl_Color n)

  Sets the color of the text in the input field.
- Fl_Font textfont () const

  Gets the font of the text in the input field.
- void textfont (Fl_Font s)

  Sets the font of the text in the input field.
- Fl_Fontsize textsize () const

  Gets the size of the text in the input field.
- void textsize (Fl_Fontsize s)

  Sets the size of the text in the input field.
- int undo ()

  Undoes previous changes to the text buffer.
- const char ∗ value () const

  Returns the text displayed in the widget.
- int value (const char ∗)

  Changes the widget text.
- int value (const char ∗, int)

  Changes the widget text.
- int value (double value)

  Changes the widget text to a floating point number ("%g").
- int value (int value)

  Changes the widget text to a signed integer number.
- int wrap () const

  Gets the word wrapping state of the input field.
- void wrap (int b)

  Sets the word wrapping state of the input field.
- ∼Fl_Input_ ()

  Destroys the widget.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.

- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.

- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

Gets the image that is used as part of the widget label when in the active state.

- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".

- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

Gets the user data for this widget.

- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Static Public Attributes inherited from Fl_Input

- static const char * **copy_menu_text** = "Copy"

  [this text may be customized at run-time]
- static const char * **cut_menu_text** = "Cut"

  [this text may be customized at run-time]
- static const char * **paste_menu_text** = "Paste"

  [this text may be customized at run-time]

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Input

- void draw () override

  Draws the widget.
- int handle_key ()

  Handles a keystroke.
- int handle_rmb ()

  Handle right mouse button down events.

Protected Member Functions inherited from Fl_Input_

- int apply_undo ()

  Apply the current undo/redo operation.
- void drawtext (int, int, int, int)

  Draws the text in the passed bounding box.
- void drawtext (int, int, int, int, bool draw_active)

  Draws the text in the passed bounding box.
- void handle_mouse (int, int, int, int, int keepmark=0)

  Handles mouse clicks and mouse moves.
- int handletext (int e, int, int, int, int)

  Handles all kinds of text field related events.
- int line_end (int i) const

  Finds the end of a line.
- int line_start (int i) const

  Finds the start of a line.
- int **linesPerPage** ()
- void **maybe_do_callback** (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)
- int up_down_position (int, int keepmark=0)

  Moves the cursor to the column given by `up_down_pos`.
- int word_end (int i) const

  Finds the end of a word.
- int word_start (int i) const

  Finds the start of a word.
- int **xscroll** () const
- int **yscroll** () const
- void **yscroll** (int yOffset)

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

13.126.1 Member Function Documentation

13.126.1.1 handle()

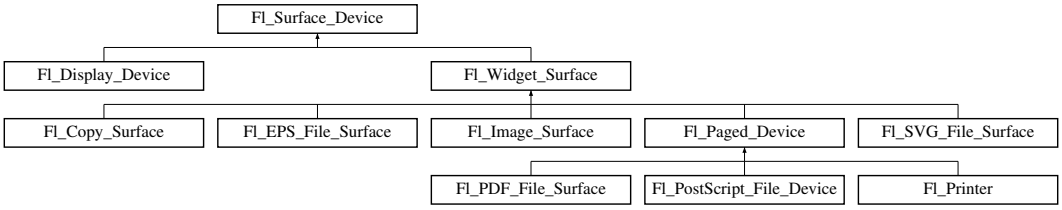
```
int Fl_Spinner::Fl_Spinner_Input::handle (
            int event )  [override], [virtual]
```

Handles events of Fl_Spinner's embedded input widget.
Works like Fl_Input::handle() but ignores FL_Up and FL_Down keys so they can be handled by the parent widget (Fl_Spinner).
Reimplemented from Fl_Input.
The documentation for this class was generated from the following files:

- Fl_Spinner.H
- Fl_Spinner.cxx

13.127 Fl_Surface_Device Class Reference

A drawing surface that's susceptible to receive graphical output.
`#include <Fl_Device.H>`
Inheritance diagram for Fl_Surface_Device:



Public Member Functions

- Fl_Graphics_Driver * **driver** ()

  Returns the graphics driver of this drawing surface.
- virtual bool is_current ()

  Is this surface the current drawing surface?
- virtual void set_current (void)

  Make this surface the current drawing surface.
- virtual ~**Fl_Surface_Device** ()

  The destructor.

Static Public Member Functions

- static Fl_Surface_Device * pop_current ()

  Removes the top element from the current drawing surface stack, and makes the new top element current.
- static void push_current (Fl_Surface_Device *new_current)

  Pushes `new_current` on top of the stack of current drawing surfaces, and makes it current.
- static Fl_Surface_Device * surface ()

  The current drawing surface.

Protected Member Functions

- void **driver** (Fl_Graphics_Driver *graphics_driver)

  Sets the graphics driver of this drawing surface.
- virtual void end_current ()

  FLTK calls this each time a surface ceases to be the current drawing surface.
- **Fl_Surface_Device** (Fl_Graphics_Driver *graphics_driver)

  Constructor that sets the graphics driver to use for the created surface.

13.127.1 Detailed Description

A drawing surface that's susceptible to receive graphical output.
Any FLTK application has at any time a current drawing surface to which all drawing requests are directed. The current surface is given by Fl_Surface_Device::surface(). When main() begins running, the current drawing surface has been set to the computer's display, an instance of the Fl_Display_Device class.
A drawing surface other than the computer's display, is typically used as follows:

1. Create `surface`, an object from an Fl_Surface_Device derived class (e.g., Fl_Image_Surface, Fl_Printer, Fl_Copy_Surface).

2. Call `Fl_Surface_Device::push_current(surface);` to redirect all graphics requests to `surface` which becomes the new current drawing surface (not necessary with classes Fl_Printer / Fl_PostScript_File_Device / Fl_PDF_File_Surface because it is done by Fl_Paged_Device::begin_page()).

3. At this point all of the Drawing functions (e.g., fl_rect()) or the Color & Font functions or Drawing Images functions (e.g., fl_draw_image(), Fl_Image::draw()) operate on the new current drawing surface. It's also possible to draw to the current surface any widget with Fl_Widget_Surface::draw(Fl_Widget*, int, int), a window and its titlebar with Fl_Widget_Surface::draw_decorated_window(), or the content of a rectangular zone of a window with Fl_Widget_Surface::print_window_part().

4. After all drawing requests have been performed, redirect graphics requests back to their previous destination with `Fl_Surface_Device::pop_current();` (not necessary with classes Fl_Printer / Fl_PostScript_File_Device / Fl_PDF_File_Surface).

5. Delete `surface`.

For back-compatibility, it is also possible to use the Fl_Surface_Device::set_current() member function to change the current drawing surface, once to the new surface, once to the previous one.

13.127.2 Member Function Documentation

13.127.2.1 end_current()

`virtual void Fl_Surface_Device::end_current ( ) [inline], [protected], [virtual]`

FLTK calls this each time a surface ceases to be the current drawing surface.
This member function is mostly of interest to developers of new Fl_Surface_Device derived classes. It allows to perform surface-specific operations necessary when this surface ceases to be current. Each implementation should end with a call to Fl_Surface_Device::end_current().
Reimplemented in Fl_PostScript_File_Device.

13.127.2.2 is_current()

`bool Fl_Surface_Device::is_current ( ) [virtual]`

Is this surface the current drawing surface?
Reimplemented in Fl_Copy_Surface, Fl_Image_Surface, Fl_PDF_File_Surface, and Fl_Printer.

13.127.2.3 pop_current()

`Fl_Surface_Device * Fl_Surface_Device::pop_current ( ) [static]`

Removes the top element from the current drawing surface stack, and makes the new top element current.

Returns

A pointer to the new current drawing surface.

See also

Fl_Surface_Device::push_current(Fl_Surface_Device *)

Version

1.4.0

13.127.2.4 push_current()

```
void Fl_Surface_Device::push_current (
            Fl_Surface_Device * new_current ) [static]
```

Pushes `new_current` on top of the stack of current drawing surfaces, and makes it current.
`new_current` will receive all future graphics requests.
Any call to push_current() must be matched by a subsequent call to Fl_Surface_Device::pop_current(). The max height of this stack is 16.

Version

1.4.0

13.127.2.5 set_current()

```
void Fl_Surface_Device::set_current (
            void ) [virtual]
```

Make this surface the current drawing surface.
This surface will receive all future graphics requests.
Since FLTK 1.4.0 the preferred API to change the current drawing surface is Fl_Surface_Device::push_current( ) / Fl_Surface_Device::pop_current().

Note

It is recommended to use this function only as follows :

- The current drawing surface is the display;
- make current another surface, e.g., an Fl_Printer or an Fl_Image_Surface object, calling set_current() on this object;
- draw to that surface;
- make the display current again with Fl_Display_Device::display_device()->set_current();
  don't do any other call to set_current() before this one.

Other scenarios of drawing surface changes should be performed via Fl_Surface_Device::push_current() and Fl_Surface_Device::pop_current().
Reimplemented in Fl_Copy_Surface, Fl_Image_Surface, Fl_PDF_File_Surface, Fl_PostScript_File_Device, and Fl_Printer.

13.127.2.6 surface()

```
static Fl_Surface_Device * Fl_Surface_Device::surface ( ) [inline], [static]
```

The current drawing surface.
In other words, the Fl_Surface_Device object that currently receives all graphics requests.

Note

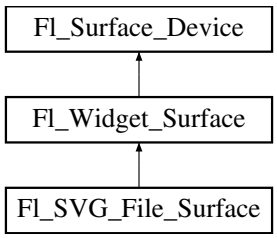
It's possible to transiently remove the GUI scaling factor in place in the current drawing surface with fl_override_scale().

The documentation for this class was generated from the following files:

- Fl_Device.H
- Fl_Device.cxx

13.128 Fl_SVG_File_Surface Class Reference

A drawing surface producing a Scalable Vector Graphics (SVG) file.
`#include <Fl_SVG_File_Surface.H>`
Inheritance diagram for Fl_SVG_File_Surface:



Public Member Functions

- int close ()

  Closes the FILE pointer where SVG data is output.

- FILE * **file** ()

  Returns the underlying FILE pointer.
- Fl_SVG_File_Surface (int width, int height, FILE *svg, int(*closef)(FILE *)=NULL)

  Constructor of the SVG drawing surface.
- void origin (int *x, int *y) override

  Computes the coordinates of the current origin of graphics functions.
- void origin (int x, int y) override

  Sets the position of the origin of graphics in the drawable part of the drawing surface.
- int printable_rect (int *w, int *h) override

  Computes the width and height of the drawable area of the drawing surface.
- void translate (int x, int y) override

  Translates the current graphics origin accounting for the current rotation.
- void untranslate () override

  Undoes the effect of a previous translate() call.
- ~Fl_SVG_File_Surface ()

  Destructor.

Public Member Functions inherited from Fl_Widget_Surface

- void draw (Fl_Widget *widget, int delta_x=0, int delta_y=0)

  Draws the widget on the drawing surface.
- void draw_decorated_window (Fl_Window *win, int x_offset=0, int y_offset=0)

  Draws a window with its title bar and frame if any.
- void print_window_part (Fl_Window *win, int x, int y, int w, int h, int delta_x=0, int delta_y=0)

  Draws a rectangular part of an on-screen window.

Public Member Functions inherited from Fl_Surface_Device

- Fl_Graphics_Driver * **driver** ()

  Returns the graphics driver of this drawing surface.
- virtual bool is_current ()

  Is this surface the current drawing surface?
- virtual void set_current (void)

  Make this surface the current drawing surface.
- virtual ~**Fl_Surface_Device** ()

  The destructor.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Surface_Device

- static Fl_Surface_Device * pop_current ()

  Removes the top element from the current drawing surface stack, and makes the new top element current.
- static void push_current (Fl_Surface_Device *new_current)

  Pushes `new_current` on top of the stack of current drawing surfaces, and makes it current.
- static Fl_Surface_Device * surface ()

  The current drawing surface.

Protected Member Functions inherited from Fl_Widget_Surface

- Fl_Widget_Surface (Fl_Graphics_Driver *d)

  The constructor.

Protected Member Functions inherited from Fl_Surface_Device

- void **driver** (Fl_Graphics_Driver *graphics_driver)

  Sets the graphics driver of this drawing surface.
- virtual void end_current ()

  FLTK calls this each time a surface ceases to be the current drawing surface.
- **Fl_Surface_Device** (Fl_Graphics_Driver *graphics_driver)

  Constructor that sets the graphics driver to use for the created surface.

Protected Attributes inherited from Fl_Widget_Surface

- int **x_offset**

  horizontal offset to the origin of graphics coordinates
- int **y_offset**

  vertical offset to the origin of graphics coordinates

13.128.1 Detailed Description

A drawing surface producing a Scalable Vector Graphics (SVG) file.
This drawing surface allows to store any FLTK graphics in vectorial form in a "Scalable Vector Graphics" file.
Usage example:

```
Fl_Window *win = ...// Window to draw to a .svg file
int ww = win->decorated_w();
int wh = win->decorated_h();
FILE *svg = fl_fopen("/path/to/mywindow.svg", "w");
if (svg) {
  Fl_SVG_File_Surface *surface = new Fl_SVG_File_Surface(ww, wh, svg);
  Fl_Surface_Device::push_current(surface);
  fl_color(FL_WHITE);
  fl_rectf(0, 0, ww, wh);
  surface->draw_decorated_window(win);
  Fl_Surface_Device::pop_current();
  delete surface; // the .svg file is not complete until the destructor was run
  fclose(svg);
}
```

Note

FLTK uses the PNG and JPEG libraries to encode images to the SVG format. For this reason, class Fl_SVG_File_Surface is placed in the fltk_images library. If JPEG is not available at application build time, PNG is enough (but produces a quite larger output). If PNG isn't available either, images don't appear in the SVG output.

13.128.2 Constructor & Destructor Documentation

13.128.2.1 Fl_SVG_File_Surface()

```
Fl_SVG_File_Surface::Fl_SVG_File_Surface (
            int width,
            int height,
            FILE * svg,
            int(*)(FILE *) closef = NULL )
```

Constructor of the SVG drawing surface.

Parameters

width,height	Width and height of the graphics area in FLTK drawing units
svg	A writable FILE pointer where the SVG data are to be sent. The resulting SVG data are not complete until after destruction of the Fl_SVG_File_Surface object or after calling close().
closef	If not NULL, the destructor and close() will call `closef(svg)` after all SVG data has been sent. If NULL, `fclose(svg)` is called instead. This allows to close the FILE pointer by, e.g., `pclose`, or, using a function such as "`int keep_open(FILE*){return 0;}`", to keep it open after completion of all output to `svg`. Function `closef` should return non zero to indicate an error.

13.128.2.2 ~Fl_SVG_File_Surface()

```
Fl_SVG_File_Surface::~Fl_SVG_File_Surface ( )
```

Destructor.
The underlying FILE pointer is processed as by close().

13.128.3 Member Function Documentation

13.128.3.1 close()

```
int Fl_SVG_File_Surface::close ( )
```

Closes the FILE pointer where SVG data is output.
The underlying FILE is closed by function fclose() unless another function was set at object's construction time. The only operation possible after this on the Fl_SVG_File_Surface object is its destruction.

Returns

The value returned by the closing function call.

13.128.3.2 origin() [1/2]

```
void Fl_SVG_File_Surface::origin (
            int * x,
            int * y )  [override], [virtual]
```

Computes the coordinates of the current origin of graphics functions.

Parameters

out	*x,y*	If non-null, ∗x and ∗y are set to the horizontal and vertical coordinates of the graphics origin.

Reimplemented from Fl_Widget_Surface.

13.128.3.3 origin() [2/2]

```
void Fl_SVG_File_Surface::origin (
            int x,
            int y )  [override], [virtual]
```

Sets the position of the origin of graphics in the drawable part of the drawing surface.
Arguments should be expressed relatively to the result of a previous printable_rect() call. That is, `printable_rect(&w, &h); origin(w/2, 0);` sets the graphics origin at the top center of the drawable area. Successive origin() calls don't combine their effects. Origin() calls are not affected by rotate() calls (for classes derived from Fl_Paged_Device).

Parameters

in	*x,y*	Horizontal and vertical positions in the drawing surface of the desired origin of graphics.

Reimplemented from Fl_Widget_Surface.

13.128.3.4 printable_rect()

```
int Fl_SVG_File_Surface::printable_rect (
            int * w,
            int * h )  [override], [virtual]
```

Computes the width and height of the drawable area of the drawing surface.
Values are in the same unit as that used by FLTK drawing functions and are unchanged by calls to origin(). If the object is derived from class Fl_Paged_Device, values account for the user-selected paper type and print orientation and are changed by scale() calls.

Returns

0 if OK, non-zero if any error

Reimplemented from Fl_Widget_Surface.

13.128.3.5 translate()

```
void Fl_SVG_File_Surface::translate (
            int x,
            int y )  [override], [virtual]
```

Translates the current graphics origin accounting for the current rotation.
Each translate() call must be matched by an untranslate() call. Successive translate() calls add up their effects.
Reimplemented from Fl_Widget_Surface.

13.128.3.6 untranslate()

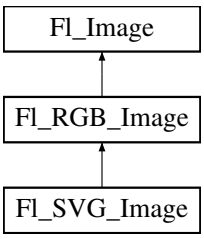
```
void Fl_SVG_File_Surface::untranslate (
            void  )  [override], [virtual]
```

Undoes the effect of a previous translate() call.
Reimplemented from Fl_Widget_Surface.
The documentation for this class was generated from the following file:

- Fl_SVG_File_Surface.H

13.129 Fl_SVG_Image Class Reference

The Fl_SVG_Image class supports loading, caching and drawing of scalable vector graphics (SVG) images.
`#include <Fl_SVG_Image.H>`
Inheritance diagram for Fl_SVG_Image:



Public Member Functions

- const Fl_SVG_Image ∗ as_svg_image () const override

  Returns whether an RGB image is an Fl_SVG_Image or not.
- Fl_SVG_Image ∗ as_svg_image () override

  Returns whether an RGB image is an Fl_SVG_Image or not.
- void color_average (Fl_Color c, float i) override

  The color_average() method averages the colors in the image with the provided FLTK color value.
- Fl_Image ∗ **copy** () const
- Fl_Image ∗ copy (int W, int H) const override

  Creates a resized copy of the image.
- void desaturate () override

  The desaturate() method converts an image to grayscale.
- void **draw** (int X, int Y)
- void draw (int X, int Y, int W, int H, int cx=0, int cy=0) override

  Draws the image to the current drawing surface with a bounding box.
- Fl_SVG_Image (const char ∗filename)

Load an SVG image from a file.

- Fl_SVG_Image (const char *sharedname, const char *svg_data)

  Load an SVG image from memory.
- Fl_SVG_Image (const char *sharedname, const unsigned char *svg_data, size_t length)

  Load an SVG image from memory.
- void normalize () override

  Makes sure the object is fully initialized.
- void resize (int width, int height)

  Have the svg data (re-)rasterized using the given `width` and `height` values.
- void scale (int w, int h, int keep_aspect=1, int can_expand=0) override

  Sets the drawing size of the image.
- virtual ~**Fl_SVG_Image** ()

  The destructor frees all memory and server resources that are used by the SVG image.

Public Member Functions inherited from Fl_RGB_Image

- int **cache_h** () const
- int **cache_w** () const
- Fl_Image * **copy** () const
- void **draw** (int X, int Y)
- Fl_RGB_Image (const Fl_Pixmap *pxm, Fl_Color bg=FL_GRAY)

  The constructor creates a new RGBA image from the specified Fl_Pixmap.
- Fl_RGB_Image (const uchar *bits, int bits_length, int W, int H, int D, int LD)

  The constructor creates a new image from the specified data.
- Fl_RGB_Image (const uchar *bits, int W, int H, int D=3, int LD=0)

  The constructor creates a new image from the specified data.
- void label (Fl_Menu_Item *m) override

  This method is an obsolete way to set the image attribute of a menu item.
- void label (Fl_Widget *w) override

  This method is an obsolete way to set the image attribute of a widget or menu item.
- void uncache () override

  If the image has been cached for display, delete the cache data.
- ~**Fl_RGB_Image** () override

  The destructor frees all memory and server resources that are used by the image.

Public Member Functions inherited from Fl_Image

- virtual class Fl_Shared_Image * as_shared_image ()

  Returns whether an image is an Fl_Shared_Image or not.
- Fl_Image * copy () const

  Creates a copy of the image in the same size.
- int count () const

  Returns the number of data values associated with the image.
- int d () const

  Returns the image depth.
- const char *const * data () const

  Returns a pointer to the current image data array.
- int **data_h** () const

  Returns the height of the image data.
- int **data_w** () const

  Returns the width of the image data.
- void draw (int X, int Y)

Draws the image to the current drawing surface.

- int fail () const

  Returns a value that is not 0 if there is currently no image available.
- Fl_Image (int W, int H, int D)

  The constructor creates an empty image with the specified width, height, and depth.
- int h () const

  Returns the current image drawing height in FLTK units.
- void inactive ()

  The inactive() method calls color_average(FL_BACKGROUND_COLOR, 0.33f) to produce an image that appears grayed out.
- int ld () const

  Returns the current line data size in bytes.
- virtual void release ()

  Releases an Fl_Image - the same as '`delete this`'.
- int w () const

  Returns the current image drawing width in FLTK units.
- virtual ~**Fl_Image** ()

  The destructor is a virtual method that frees all memory used by the image.

Public Attributes

- bool proportional

  Set this to `false` to allow image re-scaling that alters the image aspect ratio.

Public Attributes inherited from Fl_RGB_Image

- int **alloc_array**

  If non-zero, the object's data array is delete[]'d when deleting the object.
- const uchar * array

  Points to the start of the object's data array.

Additional Inherited Members

Static Public Member Functions inherited from Fl_RGB_Image

- static size_t max_size ()

  Returns the maximum allowed image size in bytes when creating an Fl_RGB_Image object.
- static void max_size (size_t size)

  Sets the maximum allowed image size in bytes when creating an Fl_RGB_Image object.

Static Public Member Functions inherited from Fl_Image

- static Fl_Labeltype **define_FL_IMAGE_LABEL** ()
- static Fl_RGB_Scaling **RGB_scaling** ()

  Returns the currently used RGB image scaling method.
- static void RGB_scaling (Fl_RGB_Scaling)

  Sets the RGB image scaling method used for copy(int, int).
- static Fl_RGB_Scaling **scaling_algorithm** ()

  Gets what algorithm is used when resizing a source image to draw it.
- static void scaling_algorithm (Fl_RGB_Scaling algorithm)

  Sets what algorithm is used when resizing a source image to draw it.

Static Public Attributes inherited from Fl_Image

- static const int **ERR_FILE_ACCESS** = -2
- static const int **ERR_FORMAT** = -3
- static const int **ERR_MEMORY_ACCESS** = -4
- static const int **ERR_NO_IMAGE** = -1
- static bool **register_images_done** = false

  True after fl_register_images() was called, false before.

Protected Member Functions inherited from Fl_Image

- void **d** (int D)

  Sets the current image depth.
- void data (const char *const *p, int c)

  Sets the current data pointer and count of pointers in the array.
- void draw_empty (int X, int Y)

  The protected method draw_empty() draws a box with an X in it.
- int draw_scaled (int X, int Y, int W, int H)

  Draw the image to the current drawing surface rescaled to a given width and height.
- void h (int H)

  Sets the height of the image data.
- void ld (int LD)

  Sets the current line data size in bytes.
- void w (int W)

  Sets the width of the image data.

Static Protected Member Functions inherited from Fl_Image

- static void **labeltype** (const Fl_Label *lo, int lx, int ly, int lw, int lh, Fl_Align la)
- static void **measure** (const Fl_Label *lo, int &lw, int &lh)

13.129.1 Detailed Description

The Fl_SVG_Image class supports loading, caching and drawing of scalable vector graphics (SVG) images.
The FLTK library performs parsing and rasterization of SVG data using a modified version of the `nanosvg` software (`https://github.com/memononen/nanosvg`). The software modification allows the option to change the aspect ratio while performing rasterization.
Use Fl_Image::fail() to check if the Fl_SVG_Image failed to load. fail() returns ERR_FILE_ACCESS if the file could not be opened or read, and ERR_FORMAT if the SVG format could not be decoded. If the image has loaded correctly, w(), h(), and d() should return values greater than zero.
Member proportional of an Fl_SVG_Image object must be set to `false` to allow the image object to be rasterized with an altered aspect ratio.
Rasterization is not done until the image is first drawn or resize() or normalize() is called. Therefore, array is NULL until then. The delayed rasterization ensures an Fl_SVG_Image is always rasterized to the exact screen resolution at which it is drawn.
The Fl_SVG_Image class draws images computed by `nanosvg` with the following known limitations

- text between `<text>` and <text> marks,
- `image` elements, and
- <use> statements

are not rendered.
The FLTK library can optionally be built without SVG support; in that case, class Fl_SVG_Image is unavailable.
Example of displaying a hard-coded svg file:

```
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
```

```
#include <FL/Fl_Box.H>
#include <FL/Fl_SVG_Image.H>

// A black rotated rectangle
const char *svg_data = "<svg viewBox=\"0 0 200 200\" version = \"1.1\">\n"
                       "<rect x=\"25\" y=\"50\" width=\"150\" height=\"100\" fill=\"black\" "
                       "transform=\"rotate(45 100 100)\"> </svg>\n";

int main(int argc, char **argv) {
  Fl_SVG_Image *svg = new Fl_SVG_Image(0, svg_data);       // create SVG object
  Fl_Window    *win = new Fl_Window(720, 486, "svg test");
  Fl_Box       *box = new Fl_Box(0, 0, win->w(), win->h());
  box->image(svg);  // assign svg object to Fl_Box
  win->end();
  win->show(argc,argv);
  return(Fl::run());
}
```

Example of displaying an svg image from a file:

```
#include <errno.h>   // errno
#include <string.h>  // strerror
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_SVG_Image.H>
#include <FL/fl_message.H>
int main(int argc, char **argv) {
  Fl_Window *win = new Fl_Window(720, 486, "svg test");
  Fl_Box    *box = new Fl_Box(0, 0, win->w(), win->h());

  // Load svg image from disk, assign to a box
  const char *svgpath = "/var/tmp/simple.svg";
  Fl_SVG_Image *svg = new Fl_SVG_Image(svgpath);  // load SVG object from disk
  switch (svg->fail()) {
    case Fl_Image::ERR_FILE_ACCESS:
      // File couldn't load? show path + os error to user
      fl_alert("%s: %s", svgpath, strerror(errno));
      return 1;
    case Fl_Image::ERR_FORMAT:
      // Parsing error
      fl_alert("%s: couldn't decode image", svgpath);
      return 1;
  }
  box->image(svg);  // assign svg object to box

  win->end();
  win->show(argc,argv);
  return(Fl::run());
}
```

Example of fitting an svg image to a resizable Fl_Box:

```
#include <FL/Fl_Window.H>
#include <FL/Fl_SVG_Image.H>
#include <FL/Fl_Box.H>

class resizable_box : public Fl_Box {
public:
  resizable_box(int w, int h) : Fl_Box(0, 0, w, h, NULL) {}
  virtual void resize(int x, int y, int w, int h) {
    image()->scale(w, h, 1, 1); // p3 = proportional, p4 = can_expand
    Fl_Box::resize(x, y, w, h);
  }
};

int main(int argc, char **argv) {
  Fl_Window *win = new Fl_Window(130, 130);
  resizable_box *box = new resizable_box(win->w(), win->h());
  Fl_SVG_Image *svg = new Fl_SVG_Image("/path/to/image.svg");
  box->image(svg);
  svg->scale(box->w(), box->h());
  win->end();
  win->resizable(win);
  win->show(argc, argv);
  return Fl::run();
}
```

13.129.2 Constructor & Destructor Documentation

13.129.2.1 Fl_SVG_Image() [1/3]

```
Fl_SVG_Image::Fl_SVG_Image (
            const char * filename )
```

Load an SVG image from a file.

This constructor loads the SVG image from a .svg or .svgz file. The reader recognizes if the data is compressed,

and decompresses it if zlib is available (HAVE_LIBZ).

Parameters

filename	the filename for a .svg or .svgz file

13.129.2.2 Fl_SVG_Image() [2/3]

```
Fl_SVG_Image::Fl_SVG_Image (
            const char * sharedname,
            const char * svg_data )
```

Load an SVG image from memory.
This constructor loads the SVG image from a block of memory. This version is commonly used for uncompressed text data, but the reader recognizes if the data is compressed, and decompresses it if zlib is available (HAVE_LIBZ).

Parameters

sharedname	if not `NULL`, a shared image will be generated with this name
svg_data	a pointer to the memory location of the SVG image data

Note

In-memory SVG data is parsed by the object constructor and is no longer needed after construction.

13.129.2.3 Fl_SVG_Image() [3/3]

```
Fl_SVG_Image::Fl_SVG_Image (
            const char * name,
            const unsigned char * svg_data,
            size_t length )
```

Load an SVG image from memory.
This constructor loads the SVG image from a block of memory. This version is commonly used for compressed binary data, but the reader recognizes if the data is uncompressed, and reads it as a text block.

Parameters

name	if not `NULL`, a shared image will be generated with this name
svg_data	a pointer to the memory location of the SVG image data
length	of `svg_data` or `0` if the length is unknown. This will protect memory outside of the `svg_data` array from illegal read operations for compressed SVG data

Note

In-memory SVG data is parsed by the object constructor and is no longer needed after construction.

13.129.3 Member Function Documentation

13.129.3.1 as_svg_image() [1/2]

```
const Fl_SVG_Image * Fl_SVG_Image::as_svg_image ( ) const  [inline], [override], [virtual]
```

Returns whether an RGB image is an Fl_SVG_Image or not.
This virtual method returns a pointer to the Fl_SVG_Image if this object is an instance of const Fl_SVG_Image or NULL if not.
Reimplemented from Fl_RGB_Image.

13.129.3.2 as_svg_image() [2/2]

```
Fl_SVG_Image * Fl_SVG_Image::as_svg_image ( ) [inline], [override], [virtual]
```

Returns whether an RGB image is an Fl_SVG_Image or not.
This virtual method returns a pointer to the Fl_SVG_Image if this object is an instance of Fl_SVG_Image or NULL if not.
Reimplemented from Fl_RGB_Image.

13.129.3.3 color_average()

```
void Fl_SVG_Image::color_average (
            Fl_Color c,
            float i ) [override], [virtual]
```

The color_average() method averages the colors in the image with the provided FLTK color value.
The first argument specifies the FLTK color to be used.
The second argument specifies the amount of the original image to combine with the color, so a value of 1.0 results in no color blend, and a value of 0.0 results in a constant image of the specified color.
An internal copy is made of the original image data before changes are applied, to avoid modifying the original image data in memory.
Reimplemented from Fl_RGB_Image.

13.129.3.4 copy()

```
Fl_Image * Fl_SVG_Image::copy (
            int W,
            int H ) const [override], [virtual]
```

Creates a resized copy of the image.
It is recommended not to call this member function to reduce the size of an image to the size of the area where this image will be drawn, and to use Fl_Image::scale() instead.
The new image should be released when you are done with it.
Note: since FLTK 1.4.0 you can use Fl_Image::release() for all types of images (i.e. all subclasses of Fl_Image) instead of operator *delete* for Fl_Image's and Fl_Image::release() for Fl_Shared_Image's.
The new image data will be converted to the requested size. RGB images are resized using the algorithm set by Fl_Image::RGB_scaling().
For the new image the following equations are true:

- w() == data_w() == `W`
- h() == data_h() == `H`

Parameters

in	*W,H*	Requested width and height of the new image

Note

The returned image can be safely cast to the same image type as that of the source image provided this type is one of Fl_RGB_Image, Fl_SVG_Image, Fl_Pixmap, Fl_Bitmap, Fl_Tiled_Image, Fl_Anim_GIF_Image and Fl_Shared_Image. Returned objects copied from images of other, derived, image classes belong to the parent class appearing in this list. For example, the copy of an Fl_GIF_Image is an object of class Fl_Pixmap.

Since FLTK 1.4.0 this method is 'const'. If you derive your own class from Fl_Image or any subclass your overridden methods of 'Fl_Image::copy() const' and 'Fl_Image::copy(int, int) const' **must** also be 'const' for inheritance to work properly. This is different than in FLTK 1.3.x and earlier where these methods have not been 'const'.

Reimplemented from Fl_RGB_Image.

13.129.3.5 desaturate()

```
void Fl_SVG_Image::desaturate ( ) [override], [virtual]
```

The desaturate() method converts an image to grayscale.
If the image contains an alpha channel (depth = 4), the alpha channel is preserved.
An internal copy is made of the original image data before changes are applied, to avoid modifying the original image data in memory.
Reimplemented from Fl_RGB_Image.

13.129.3.6 draw()

```
void Fl_SVG_Image::draw (
            int X,
            int Y,
            int W,
            int H,
            int cx = 0,
            int cy = 0 )  [override], [virtual]
```

Draws the image to the current drawing surface with a bounding box.
Arguments `X,Y,W,H` specify a bounding box for the image, with the origin (upper-left corner) of the image offset by the `cx` and `cy` arguments.
Another way to see what part of the image gets drawn and where, is to consider this alternative writing producing the same output:

```
fl_push_clip(X,Y,W,H);
this->draw(X - cx, Y - cy);
fl_pop_clip();
```

Repeated calls to this member function with the same image but varying `W, H, cx`, or `cy` arguments can be more efficiently processed using the above alternative writing.
Reimplemented from Fl_RGB_Image.

13.129.3.7 normalize()

```
void Fl_SVG_Image::normalize ( )  [override], [virtual]
```

Makes sure the object is fully initialized.
This function rasterizes the SVG image, and consequently initializes its array member, if that was not done before.
Reimplemented from Fl_RGB_Image.

13.129.3.8 resize()

```
void Fl_SVG_Image::resize (
            int width,
            int height )
```

Have the svg data (re-)rasterized using the given `width` and `height` values.
By default, the resulting image w() and h() will be close to `width` and `height` while preserving the width/height ratio of the SVG data. If proportional was set to `false`, the image is rasterized to the exact `width` and `height` values. In both cases, data_w() and data_h() values are set to w() and h(), respectively.

13.129.3.9 scale()

```
void Fl_SVG_Image::scale (
            int width,
            int height,
            int proportional = 1,
            int can_expand = 0 )  [override], [virtual]
```

Sets the drawing size of the image.
This function controls the values returned by member functions w() and h() which in turn control how the image is drawn: the full image data (whose size is given by data_w() and data_h()) are drawn scaled to an area of the drawing surface sized at w() x h() FLTK units. This can make a difference if the drawing surface has more than 1 pixel per FLTK unit because the image can be drawn at the full resolution of the drawing surface. Examples of such drawing surfaces: HiDPI displays, laser printers, PostScript files, PDF printers.

Parameters

width,height	maximum values, in FLTK units, that w() and h() should return
proportional	if not null, keep the values returned by w() and h() proportional to data_w() and data_h()
can_expand	if null, the values returned by w() and h() will not be larger than data_w() and data_h(), respectively

Note

This function generally changes the values returned by the w() and h() member functions. In contrast, the values returned by data_w() and data_h() remain unchanged.

If the processed image is an Fl_SVG_Image, setting Fl_SVG_Image::proportional to `false` is required if `proportional` here is null.

Version

1.4 (1.3.4 and FL_ABI_VERSION for Fl_Shared_Image only)

Example code: scale an image to fit in a box

```
Fl_Box *b = ...  // a box
Fl_Image *img = new Fl_PNG_Image("/path/to/picture.png"); // read a picture file
// set the drawing size of the image to the size of the box keeping its aspect ratio
img->scale(b->w(), b->h());
b->image(img); // use the image as the box image
```

Reimplemented from Fl_Image.

13.129.4 Member Data Documentation

13.129.4.1 proportional

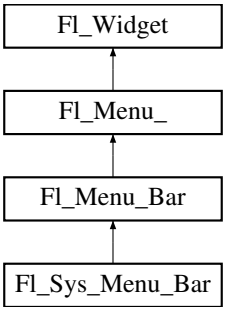
`bool Fl_SVG_Image::proportional`

Set this to `false` to allow image re-scaling that alters the image aspect ratio.
Upon object creation, proportional is set to `true`, and the aspect ratio is kept constant.
The documentation for this class was generated from the following files:

- Fl_SVG_Image.H
- Fl_SVG_Image.cxx

13.130 Fl_Sys_Menu_Bar Class Reference

A class to create and modify menus that appear on macOS in the menu bar at the top of the screen.
`#include <Fl_Sys_Menu_Bar.H>`
Inheritance diagram for Fl_Sys_Menu_Bar:



Public Types

- enum window_menu_style_enum { no_window_menu = 0 , tabbing_mode_none , tabbing_mode_automatic , tabbing_mode_preferred }

  Possible styles of the Window menu in the system menu bar.

Public Member Functions

- int add (const char *label, const char *shortcut, Fl_Callback *cb, void *user_data=0, int flags=0)

  Adds a new menu item.
- int add (const char *label, int shortcut, Fl_Callback *, void *user_data=0, int flags=0)

  Add a new menu item to the system menu bar.
- int add (const char *str)

  Forms-compatible procedure to add items to the system menu bar.
- void clear ()

  Set the Fl_Menu_Item array pointer to null, indicating a zero-length menu.
- int clear_submenu (int index)

  Clears the specified submenu pointed to by index of all menu items.
- Fl_Sys_Menu_Bar (int x, int y, int w, int h, const char *l=0)

  The constructor.
- int insert (int index, const char *label, const char *shortcut, Fl_Callback *cb, void *user_data=0, int flags=0)

  Insert a new menu item.
- int insert (int index, const char *label, int shortcut, Fl_Callback *cb, void *user_data=0, int flags=0)

  insert in the system menu bar a new menu item
- bool is_global () const

  Find out if this is a global app menubar (macOS) or part of the FLTK window.
- const Fl_Menu_Item * **menu** () const

  Return the system menu's array of Fl_Menu_Item's.
- void menu (const Fl_Menu_Item *m)

  create a system menu bar using the given list of menu structs
- int **mode** (int i) const

  Gets the flags of item i.
- void mode (int i, int fl)

  Sets the flags of item i.
- void play_menu (const Fl_Menu_Item *) override

  Opens the 1st level submenu of the menubar corresponding to `item`.
- void remove (int n)

  remove an item from the system menu bar
- void replace (int index, const char *name)

  rename an item from the system menu bar
- void **setonly** (Fl_Menu_Item *item)

  Turns the radio item "on" for the menu item and turns "off" adjacent radio items of the same group.
- void **shortcut** (int i, int s)

  Changes the shortcut of item i to n.
- void update () override

  Updates the menu bar after any change to its items.
- virtual ~**Fl_Sys_Menu_Bar** ()

  The destructor.

Public Member Functions inherited from Fl_Menu_Bar

- Fl_Menu_Bar (int X, int Y, int W, int H, const char *l=0)

  Creates a new Fl_Menu_Bar widget using the given position, size, and label string.
- int handle (int) override

  Handles the specified event.

Public Member Functions inherited from Fl_Menu_

- int add (const char *)

  This is a Forms (and SGI GL library) compatible add function, it adds many menu items, with '|' separating the menu items, and tab separating the menu item names from an optional shortcut string.
- int add (const char *, int shortcut, Fl_Callback *, void *=0, int=0)

  Adds a new menu item.
- int **add** (const char *a, const char *b, Fl_Callback *c, void *d=0, int e=0)

  See int Fl_Menu_::add(const char label, int shortcut, Fl_Callback*, void *user_data=0, int flags=0)*
- void clear ()

  Same as menu(NULL), set the array pointer to null, indicating a zero-length menu.
- int clear_submenu (int index)

  Clears the specified submenu pointed to by `index` of all menu items.
- void copy (const Fl_Menu_Item *m, void *user_data=0)

  Sets the menu array pointer with a copy of m that will be automatically deleted.
- Fl_Boxtype **down_box** () const

  This box type is used to surround the currently-selected items in the menus.
- void down_box (Fl_Boxtype b)

  Sets the box type used to surround the currently-selected items in the menus.
- Fl_Color **down_color** () const

  For back compatibility, same as selection_color()
- void **down_color** (unsigned c)

  For back compatibility, same as selection_color()
- int find_index (const char *name) const

  Find the menu item index for a given menu `pathname`, such as "Edit/Copy".
- int find_index (const Fl_Menu_Item *item) const

  Find the index into the menu array for a given `item`.
- int find_index (Fl_Callback *cb) const

  Find the index into the menu array for a given callback `cb`.
- const Fl_Menu_Item * find_item (const char *name)

  Find the menu item for a given menu `pathname`, such as "Edit/Copy".
- const Fl_Menu_Item * find_item (Fl_Callback *)

  Find the menu item for the given callback `cb`.
- const Fl_Menu_Item * find_item_with_argument (long)

  Find the menu item for the given user argument `v`.
- const Fl_Menu_Item * find_item_with_user_data (void *)

  Find the menu item for the given user data `v`.
- Fl_Menu_ (int, int, int, int, const char *=0)

  Creates a new Fl_Menu_ widget using the given position, size, and label string.
- void global ()

  Make the shortcuts for this menu work no matter what window has the focus when you type it.
- int insert (int index, const char *, int shortcut, Fl_Callback *, void *=0, int=0)

  Inserts a new menu item at the specified `index` position.
- int **insert** (int index, const char *a, const char *b, Fl_Callback *c, void *d=0, int e=0)

  See int Fl_Menu_::insert(const char label, int shortcut, Fl_Callback*, void *user_data=0, int flags=0)*
- int item_pathname (char *name, int namelen, const Fl_Menu_Item *finditem=0) const

  Get the menu 'pathname' for the specified menuitem.
- const Fl_Menu_Item * menu () const

  Returns a pointer to the array of Fl_Menu_Items.
- void menu (const Fl_Menu_Item *m)

  Sets the menu array pointer directly.

- Fl_Boxtype menu_box () const

  Get the box type for the menu popup windows.
- void menu_box (Fl_Boxtype b)

  Set the box type for the menu popup windows.
- const Fl_Menu_Item * menu_end ()

  Finishes menu modifications and returns menu().
- int mode (int i) const

  Get the flags of item i.
- void mode (int i, int fl)

  Set the flags of item i.
- const Fl_Menu_Item * mvalue () const

  Return a pointer to the last menu item that was picked.
- const Fl_Menu_Item * picked (const Fl_Menu_Item *)

  When user picks a menu item, call this.
- const Fl_Menu_Item * prev_mvalue () const

  Return a pointer to the menu item that was picked before the current one was picked.
- void remove (int)

  Deletes item `i` from the menu.
- void replace (int, const char *)

  Changes the text of item `i`.
- void **setonly** (Fl_Menu_Item *item)

  Turns the radio item "on" for the menu item and turns "off" adjacent radio items of the same group.
- void **shortcut** (int i, int s)

  Change the shortcut of item `i` to `s`.
- int size () const

  This returns the number of Fl_Menu_Item structures that make up the menu, correctly counting submenus.
- void **size** (int W, int H)
- const Fl_Menu_Item * test_shortcut ()

  Returns the menu item with the entered shortcut (key value).
- const char * text () const

  Returns the title of the last item chosen.
- const char * text (int i) const

  Returns the title of item i.
- Fl_Color textcolor () const

  Get the current color of menu item labels.
- void **textcolor** (Fl_Color c)

  Sets the current color of menu item labels.
- Fl_Font textfont () const

  Gets the current font of menu item labels.
- void textfont (Fl_Font c)

  Sets the current font of menu item labels.
- Fl_Fontsize textsize () const

  Gets the font size of menu item labels.
- void textsize (Fl_Fontsize c)

  Sets the font size of menu item labels.
- int value () const

  Return the index into the menu() of the last item chosen by the user.
- int value (const Fl_Menu_Item *)

  Set the value of a menu to the menu item `m`.
- int value (int i)

  Set the value of the menu to index `i`.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

Checks if the widget value changed since the last callback.

- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.

- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

Sets width ww and height hh accordingly with the label size.

- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.

- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Static Public Member Functions

- static void about (Fl_Callback *cb, void *data)

  Attaches a callback to the "About myprog" item of the system application menu.
- static void create_window_menu ()

  Adds a Window menu, to the end of the system menu bar.
- static window_menu_style_enum **window_menu_style** ()

  Get the style of the Window menu in the system menu bar.
- static void window_menu_style (window_menu_style_enum style)

  Set the desired style of the Window menu in the system menu bar.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Member Functions

- void draw () override

  Draws the widget.

Protected Member Functions inherited from Fl_Menu_

- int **item_pathname_** (char *name, int namelen, const Fl_Menu_Item *finditem, const Fl_Menu_Item *menu=0) const

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

Creates a widget at the given position and size.

- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Attributes inherited from Fl_Menu_

- uchar **alloc**
- uchar **down_box_**
- Fl_Boxtype **menu_box_**
- Fl_Color **textcolor_**
- Fl_Font **textfont_**
- Fl_Fontsize **textsize_**

13.130.1 Detailed Description

A class to create and modify menus that appear on macOS in the menu bar at the top of the screen.
On other than macOS platforms, Fl_Sys_Menu_Bar is a synonym of class Fl_Menu_Bar.
On the macOS platform, replace Fl_Menu_Bar with Fl_Sys_Menu_Bar, and a system menu at the top of the screen will be available. This menu will match an array of Fl_Menu_Item's exactly as in all other FLTK menus (except for the submenu with the application's own name and the 'Window' menu; see below). There is, though, an important difference between an Fl_Sys_Menu_Bar object under macOS and under other platforms: only a single object from this class can be created, because macOS uses a single system menu bar. Therefore, porting to macOS an app that creates, on other platforms, several Fl_Menu_Bar objects, one for each of several windows, is more complex that just replacing Fl_Menu_Bar by Fl_Sys_Menu_Bar.
On the macOS platform, the system menu bar of any FLTK app begins with the Application menu which the FLTK library automatically constructs. Functions Fl_Mac_App_Menu::custom_application_menu_items() and

Fl_Sys_Menu_Bar::about() can be used to further customize the Application menu. The FLTK library also automatically constructs and handles a Window menu which can be further customized (or even removed) calling Fl_Sys_Menu_Bar::window_menu_style(window_menu_style_enum style). Other member functions of this class allow the app to generate the rest of the system menu bar. It is recommended to localize the system menu bar using the standard Mac OS X localization procedure (see Internationalization).
Changes to the menu state are immediately visible in the menubar when they are made using member functions of the Fl_Sys_Menu_Bar class. Other changes (e.g., by a call to Fl_Menu_Item::set()) should be followed by a call to update() to be visible in the menubar across all platforms. macOS global variable fl_sys_menu_bar points to the unique, current system menu bar.
A few FLTK menu features are not supported by the Mac System menu:

- no symbolic labels
- no embossed labels
- no font sizes

As described above, the submenu with the application's own name (usually the second submenu from the left, immediately following the "Apple" submenu) is a special case, and can be managed with Fl_Mac_App_Menu::custom_application_menu_items(). For example, to make your own "Appname -> Preferences" dialog, you might use:

```
#include <FL/platform.H>        // for Fl_Mac_App_Menu class
#include <FL/Fl_Sys_Menu_Bar.H> // for Fl_Menu_Item
:
void prefs_cb(Fl_Widget *w, void *data) {
  // ..Open your preferences dialog here..
}
:
int main(..) {
   :
   // Items to add to the application menu
   static Fl_Menu_Item appitems[] = {
      { "Preferences", 0, prefs_cb, 0, 0 },
      { 0 }, { 0 }
    };
    Fl_Mac_App_Menu::custom_application_menu_items(appitems);  // adds it
 }
```

..the result being:

Figure 13.46 Mac Application submenu

13.130.2 Member Enumeration Documentation

13.130.2.1 window_menu_style_enum

`enum Fl_Sys_Menu_Bar::window_menu_style_enum`

Possible styles of the Window menu in the system menu bar.

Enumerator

no_window_menu	No Window menu in the system menu bar.
tabbing_mode_none	No tabbed windows, but the system menu bar contains a Window menu.
tabbing_mode_automatic	Windows are created by themselves but can be tabbed later.
tabbing_mode_preferred	Windows are tabbed when created.

13.130.3 Constructor & Destructor Documentation

13.130.3.1 Fl_Sys_Menu_Bar()

```
Fl_Sys_Menu_Bar::Fl_Sys_Menu_Bar (
            int x,
            int y,
            int w,
            int h,
            const char * l = 0 )
```

The constructor.
On Mac OS X, all arguments are unused. On other platforms they are used as by Fl_Menu_Bar::Fl_Menu_Bar().

13.130.4 Member Function Documentation

13.130.4.1 about()

```
void Fl_Sys_Menu_Bar::about (
            Fl_Callback * cb,
            void * data )  [static]
```

Attaches a callback to the "About myprog" item of the system application menu.
This cross-platform function is effective only under the MacOS platform.

Parameters

cb	a callback that will be called by "About myprog" menu item with NULL 1st argument.
data	a pointer transmitted as 2nd argument to the callback.

13.130.4.2 add() [1/3]

```
int Fl_Sys_Menu_Bar::add (
            const char * label,
            const char * shortcut,
            Fl_Callback * cb,
            void * user_data = 0,
            int flags = 0 )  [inline]
```

Adds a new menu item.

See also

Fl_Menu_::add(const char∗ label, int shortcut, Fl_Callback∗, void ∗user_data=0, int flags=0)

13.130.4.3 add() [2/3]

```
int Fl_Sys_Menu_Bar::add (
            const char * label,
            int shortcut,
            Fl_Callback * cb,
            void * user_data = 0,
            int flags = 0 )
```

Add a new menu item to the system menu bar.
Add to the system menu bar a new menu item, with a title string, shortcut int, callback, argument to the callback, and flags.

Parameters

label	- new menu item's label
shortcut	- new menu item's integer shortcut (can be 0 for none, or e.g. FL_ALT+'x')

Parameters

cb	- callback to be invoked when item selected (can be 0 for none, in which case the menubar's callback() can be used instead)
user_data	- argument to the callback
flags	- item's flags, e.g. FL_MENU_TOGGLE, etc.

Returns

the index into the menu() array, where the entry was added

See also

Fl_Menu_::add(const char∗ label, int shortcut, Fl_Callback ∗cb, void ∗user_data, int flags)

13.130.4.4 add() [3/3]

```
int Fl_Sys_Menu_Bar::add (
            const char * str )
```

Forms-compatible procedure to add items to the system menu bar.

Returns

the index into the menu() array, where the entry was added

See also

Fl_Menu_::add(const char∗ str)

13.130.4.5 clear()

```
void Fl_Sys_Menu_Bar::clear ( )
```

Set the Fl_Menu_Item array pointer to null, indicating a zero-length menu.

See also

Fl_Menu_::clear()

13.130.4.6 clear_submenu()

```
int Fl_Sys_Menu_Bar::clear_submenu (
            int index )
```

Clears the specified submenu pointed to by index of all menu items.

See also

Fl_Menu_::clear_submenu(int index)

13.130.4.7 create_window_menu()

```
void Fl_Sys_Menu_Bar::create_window_menu ( )  [static]
```

Adds a Window menu, to the end of the system menu bar.

FLTK apps typically don't need to call this function which is automatically called by the library the first time a window is shown. The default system menu bar contains a Window menu with a "Merge All Windows" item. Other Window menu styles can be obtained calling Fl_Sys_Menu_Bar::window_menu_style(window_menu_style_enum) before the first Fl_Window::show(). Alternatively, an app can call create_window_menu() after having populated the system menu bar, for example with menu(const Fl_Menu_Item ∗), and before the first Fl_Window::show().

This function does nothing on non MacOS platforms.

Version

1.4

13.130.4.8 draw()

```
void Fl_Sys_Menu_Bar::draw ( )  [override], [protected], [virtual]
```

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();                 // calls Fl_Scrollbar::draw()
```

Reimplemented from Fl_Menu_Bar.

13.130.4.9 insert() [1/2]

```
int Fl_Sys_Menu_Bar::insert (
            int index,
            const char * label,
            const char * shortcut,
            Fl_Callback * cb,
            void * user_data = 0,
            int flags = 0 )  [inline]
```

Insert a new menu item.

See also

Fl_Menu_::insert(int index, const char∗ label, const char∗ shortcut, Fl_Callback ∗cb, void ∗user_data=0, int flags=0)

13.130.4.10 insert() [2/2]

```
int Fl_Sys_Menu_Bar::insert (
            int index,
            const char * label,
            int shortcut,
            Fl_Callback * cb,
            void * user_data = 0,
            int flags = 0 )
```

insert in the system menu bar a new menu item

Insert in the system menu bar a new menu item, with a title string, shortcut int, callback, argument to the callback, and flags.

Returns

the index into the menu() array, where the entry was inserted

See also

Fl_Menu_::insert(int index, const char∗ label, int shortcut, Fl_Callback ∗cb, void ∗user_data, int flags)

13.130.4.11 is_global()

```
bool Fl_Sys_Menu_Bar::is_global ( ) const
```

Find out if this is a global app menubar (macOS) or part of the FLTK window.

Returns

true if the menu bar is located at the top of the desktop window and will not take up any space in the FLTK window.

false if the menu bar is rendered inside the FLTK window, using the coordinates and size specified in the constructor.

13.130.4.12 menu()

```
void Fl_Sys_Menu_Bar::menu (
            const Fl_Menu_Item * m )
```

create a system menu bar using the given list of menu structs

Author

Matthias Melcher

Parameters

m	Zero-ending list of Fl_Menu_Item's

13.130.4.13 mode()

```
void Fl_Sys_Menu_Bar::mode (
            int i,
            int fl )
```

Sets the flags of item i.

See also

Fl_Menu_::mode(int i, int fl)

13.130.4.14 play_menu()

```
void Fl_Sys_Menu_Bar::play_menu (
            const Fl_Menu_Item * item )  [override], [virtual]
```

Opens the 1st level submenu of the menubar corresponding to `item`.

Since

1.4.0

Reimplemented from Fl_Menu_Bar.

13.130.4.15 remove()

```
void Fl_Sys_Menu_Bar::remove (
            int index )
```

remove an item from the system menu bar

Parameters

index	the index of the item to remove

13.130.4.16 replace()

```
void Fl_Sys_Menu_Bar::replace (
            int index,
            const char * name )
```

rename an item from the system menu bar

Parameters

index	the index of the item to rename
name	the new item name as a UTF8 string

13.130.4.17 update()

```
void Fl_Sys_Menu_Bar::update ( ) [override], [virtual]
```

Updates the menu bar after any change to its items.
This is useful when the menu bar can be an Fl_Sys_Menu_Bar object.
Reimplemented from Fl_Menu_Bar.

13.130.4.18 window_menu_style()

```
void Fl_Sys_Menu_Bar::window_menu_style (
            Fl_Sys_Menu_Bar::window_menu_style_enum style ) [static]
```

Set the desired style of the Window menu in the system menu bar.
This function, to be called before the first call to Fl_Window::show(), allows to control whether the system menu bar should contain a Window menu, and if yes, whether new windows should be displayed in tabbed form. These are the effects of various values for `style` :

- `no_window_menu` : don't add a Window menu to the system menu bar
- `tabbing_mode_none` : add a simple Window menu to the system menu bar
- `tabbing_mode_automatic` : the window menu also contains "Merge All Windows" to group all windows in a single tabbed display mode. This is the **default** Window menu style for FLTK apps.
- `tabbing_mode_preferred` : new windows are displayed in tabbed mode when first created

The Window menu, if present, is entirely created and controlled by the FLTK library. Mac OS version 10.12 or later must be running for windows to be displayed in tabbed form. Under non MacOS platforms, this function does nothing.

Version

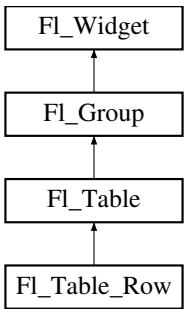
1.4

The documentation for this class was generated from the following files:

- Fl_Sys_Menu_Bar.H
- Fl_Sys_Menu_Bar.cxx

13.131 Fl_Table Class Reference

A table of widgets or other content.
`#include <Fl_Table.H>`
Inheritance diagram for Fl_Table:



Public Types

- enum TableContext {
  CONTEXT_NONE = 0 , CONTEXT_STARTPAGE = 0x01 , CONTEXT_ENDPAGE = 0x02 , CONTEXT_ROW_HEADER = 0x04 ,
  CONTEXT_COL_HEADER = 0x08 , CONTEXT_CELL = 0x10 , CONTEXT_TABLE = 0x20 , CONTEXT_RC_RESIZE = 0x40 }

  The context bit flags for Fl_Table related callbacks.

Public Member Functions

- void **add** (Fl_Widget &wgt)

  The specified widget is removed from its current group (if any) and added to the end of Fl_Table's group.
- void **add** (Fl_Widget *wgt)

  The specified widget is removed from its current group (if any) and added to the end of Fl_Table's group.
- Fl_Widget *const * array () const

  Returns a pointer to the array of children.
- void **begin** ()
- void callback (Fl_Widget *, void *)

  Callbacks will be called depending on the setting of Fl_Widget::when().
- int callback_col () const

  Returns the current column the event occurred on.
- TableContext callback_context () const

  Returns the current 'table context'.
- int callback_row () const

  Returns the current row the event occurred on.
- Fl_Widget * child (int n) const

  Returns the child widget by an index.
- int children () const

  Returns the number of children in the table.
- virtual void clear ()

  Clears the table to zero rows (rows(0)), zero columns (cols(0)), and clears any widgets (table->clear()) that were added with begin()/end() or add()/insert()/etc.
- int **col_header** () const

  Returns if column headers are enabled or not.
- void col_header (int flag)

  Enable or disable column headers.
- Fl_Color **col_header_color** () const

  Gets the color for column headers.
- void **col_header_color** (Fl_Color val)

  Sets the color for column headers and redraws the table.
- int **col_header_height** () const

  Gets the column header height.
- void **col_header_height** (int height)

  Sets the height in pixels for column headers and redraws the table.
- int **col_position** () const

  Returns the current column scroll position as a column number.
- void **col_position** (int col)

  Sets the horizontal scroll position so 'col' is at the left, and causes the screen to redraw.
- int **col_resize** () const

  Returns if column resizing by the user is allowed.
- void col_resize (int flag)

  Allows/disallows column resizing by the user.
- int **col_resize_min** () const

  Returns the current column minimum resize value.
- void col_resize_min (int val)

  Sets the current column minimum resize value.
- int **col_width** (int col) const

  Returns the current width of the specified column in pixels.
- void col_width (int col, int width)

Sets the width of the specified column in pixels, and the table is redrawn.

- void col_width_all (int width)

  Convenience method to set the width of all columns to the same value, in pixels.
- int **cols** () const

  Get the number of columns in the table.
- virtual void **cols** (int val)

  Set the number of columns in the table and redraw.
- void do_callback (TableContext context, int row, int col)

  Calls the widget callback.
- void **end** ()
- int **find** (const Fl_Widget &wgt) const
- int **find** (const Fl_Widget *wgt) const
- Fl_Table (int X, int Y, int W, int H, const char *l=0)

  The constructor for Fl_Table.
- void get_selection (int &row_top, int &col_left, int &row_bot, int &col_right) const

  Gets the region of cells selected (highlighted).
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &wgt, Fl_Widget *w2)

  The specified widget is removed from its current group (if any) and inserted into Fl_Table's group before widget 'w2'.
- void **insert** (Fl_Widget &wgt, int n)

  The specified widget is removed from its current group (if any) and inserted into the Fl_Table's group at position 'n'.
- int is_interactive_resize () const

  Returns 1 if someone is interactively resizing a row or column.
- int is_selected (int r, int c) const

  See if the cell at row `r` and column `c` is selected.
- int **move_cursor** (int R, int C)

  Same as move_cursor(R,C,1);.
- int move_cursor (int R, int C, int shiftselect)

  Moves the selection cursor a relative number of rows/columns specifed by R/C.
- void **remove** (Fl_Widget &wgt)

  The specified widget is removed from Fl_Table's group.
- void resize (int X, int Y, int W, int H) override

  Handle resize events if user resizes parent window.
- int **row_header** () const

  Returns if row headers are enabled or not.
- void row_header (int flag)

  Enables/disables showing the row headers.
- Fl_Color **row_header_color** () const

  Returns the current row header color.
- void **row_header_color** (Fl_Color val)

  Sets the row header color and causes the screen to redraw.
- int **row_header_width** () const

  Returns the current row header width (in pixels).
- void **row_header_width** (int width)

  Sets the row header width to n and causes the screen to redraw.
- int **row_height** (int row) const

  Returns the current height of the specified row as a value in pixels.
- void row_height (int row, int height)

  Sets the height of the specified row in pixels, and the table is redrawn.
- void row_height_all (int height)

Convenience method to set the height of all rows to the same value, in pixels.

- int **row_position** () const

  Returns the current row scroll position as a row number.
- void **row_position** (int row)

  Sets the vertical scroll position so 'row' is at the top, and causes the screen to redraw.
- int **row_resize** () const

  Returns if row resizing by the user is allowed.
- void row_resize (int flag)

  Allows/disallows row resizing by the user.
- int **row_resize_min** () const

  Returns the current row minimum resize value.
- void row_resize_min (int val)

  Sets the current row minimum resize value.
- int **rows** () const

  Returns the number of rows in the table.
- virtual void rows (int val)

  Sets the number of rows in the table, and the table is redrawn.
- int scrollbar_size () const

  Gets the current size of the scrollbars' troughs, in pixels.
- void scrollbar_size (int newSize)

  Sets the pixel size of the scrollbars' troughs to `newSize`, in pixels.
- void set_selection (int row_top, int col_left, int row_bot, int col_right)

  Sets the region of cells to be selected (highlighted).
- int tab_cell_nav () const

  Get state of table's 'Tab' key cell navigation flag.
- void tab_cell_nav (int val)

  Flag to control if Tab navigates table cells or not.
- void table_box (Fl_Boxtype val)

  Sets the kind of box drawn around the data table, the default being FL_NO_BOX.
- Fl_Boxtype **table_box** (void) const

  Returns the current box type used for the data table.
- int top_row () const

  Returns the current top row shown in the table.
- void top_row (int row)

  Sets which row should be at the top of the table, scrolling as necessary, and the table is redrawn.
- void visible_cells (int &r1, int &r2, int &c1, int &c2) const

  Returns the range of row and column numbers for all visible and partially visible cells in the table.
- void when (Fl_When flags)

  The Fl_Widget::when() function is used to set a group of flags, determining when the widget callback is called:
- ~Fl_Table ()

  The destructor for Fl_Table.

Public Member Functions inherited from Fl_Group

- Fl_Widget *& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget *o)

  See void Fl_Group::add(Fl_Widget &w)
- void **add_resizable** (Fl_Widget &o)

Adds a widget to the group and makes it the resizable widget.

- Fl_Widget *const * array () const

  Returns a pointer to the internal array of children.
- Fl_Group const * as_group () const override
- Fl_Group * as_group () override

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const

  Returns the n'th child.
- int **children** () const

  Returns how many child widgets the group has.
- void clear ()

  Deletes all child widgets from memory recursively.
- unsigned int clip_children () const

  Returns the current clipping mode.
- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)

  Removes the widget at `index` from the group and deletes it.
- void end ()

  Exactly the same as current(this->parent()).
- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*
- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.
- void **forms_end** ()

  This is for forms compatibility only.
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

  This does insert(w, find(before)).
- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.
- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.
- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

Marks the widget as inactive without sending events or changing focus.

- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.

- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

Returns whether this widget needs a keyboard.

- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.

- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Protected Types

- enum **ResizeFlag** {
  RESIZE_NONE = 0 , **RESIZE_COL_LEFT** = 1 , **RESIZE_COL_RIGHT** = 2 , **RESIZE_ROW_ABOVE** = 3 , **RESIZE_ROW_BELOW** = 4 }

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 , AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions

- void **change_cursor** (Fl_Cursor newcursor)

  Change mouse cursor to different type.
- long **col_scroll_position** (int col)

  Returns the scroll position (in pixels) of the specified column 'col'.
- TableContext cursor2rowcol (int &R, int &C, ResizeFlag &resizeflag)

  Find row/col for the recent mouse event.
- void damage_zone (int r1, int c1, int r2, int c2, int r3=0, int c3=0)

  Sets the damage zone to the specified row/col values.
- void draw () override

  Draws the entire Fl_Table.
- virtual void draw_cell (TableContext context, int R=0, int C=0, int X=0, int Y=0, int W=0, int H=0)

  Subclass should override this method to handle drawing the cells.
- int find_cell (TableContext context, int R, int C, int &X, int &Y, int &W, int &H)

  Find a cell's X/Y/W/H region for the specified cell in row 'R', column 'C'.
- void **get_bounds** (TableContext context, int &X, int &Y, int &W, int &H)

  Returns the (X,Y,W,H) bounding region for the specified 'context'.
- int handle (int e) override

  Handle FLTK events.
- int **is_fltk_container** () const

  Does the table contain any child fltk widgets?
- void recalc_dimensions ()

  Recalculate the dimensions of the table, and affect any children.
- void redraw_range (int topRow, int botRow, int leftCol, int rightCol)

  Define region of cells to be redrawn by specified range of rows/cols, and then sets damage(DAMAGE_CHILD).
- int row_col_clamp (TableContext context, int &R, int &C)

  Return specified row/col values R and C to within the table's current row/col limits.
- long **row_scroll_position** (int row)

  Returns the scroll position (in pixels) of the specified 'row'.
- void table_resized ()

  Call this if table was resized, to recalculate internal data.
- void table_scrolled ()

  Recalculate internals after a scroll.

Protected Member Functions inherited from Fl_Group

- Fl_Rect ∗ bounds ()

  Returns the internal array of widget sizes and positions.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget ∗, int)

  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)

  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)

  Allow derived groups to act when a child widget is removed from the group.
- int ∗ sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

Internal use only.

- void x (int v)

  Internal use only.

- void y (int v)

  Internal use only.

Static Protected Member Functions

- static void **scroll_cb** (Fl_Widget ∗, void ∗)

  Callback for when someone moves a scrollbar.

Protected Attributes

- int **botrow**

  bottom row# of currently visible table on screen

- int **current_col**

  selection cursor's current column (-1 if none)

- int **current_row**

  selection cursor's current row (-1 if none)

- Fl_Scrollbar ∗ **hscrollbar**

  child horizontal scrollbar widget

- int **leftcol**

  left column# of currently visible table on screen

- int **leftcol_scrollpos**

  precomputed scroll position for left column

- int **rightcol**

  right column# of currently visible table on screen

- int **select_col**

  extended selection column (-1 if none)

- int **select_row**

  extended selection row (-1 if none)

- Fl_Scroll ∗ **table**

  child Fl_Scroll widget container for child fltk widgets (if any)

- int **table_h**

  table's virtual height (in pixels)

- int **table_w**

  table's virtual width (in pixels)

- int **tih**

  Data table's inner h dimension, inside bounding box. See Table Dimension Diagram.

- int **tiw**

  Data table's inner w dimension, inside bounding box. See Table Dimension Diagram.

- int **tix**

  Data table's inner x dimension, inside bounding box. See Table Dimension Diagram.

- int **tiy**

  Data table's inner y dimension, inside bounding box. See Table Dimension Diagram.

- int **toh**

  Data table's outer h dimension, outside bounding box. See Table Dimension Diagram.

- int **toprow**

  top row# of currently visible table on screen

- int **toprow_scrollpos**

  precomputed scroll position for top row

- int **tow**

Data table's outer w dimension, outside bounding box. See Table Dimension Diagram.

- int **tox**

  Data table's outer x dimension, outside bounding box. See Table Dimension Diagram.

- int **toy**

  Data table's outer y dimension, outside bounding box. See Table Dimension Diagram.

- Fl_Scrollbar * **vscrollbar**

  child vertical scrollbar widget

- int **wih**

  Table widget's inner h dimension, inside bounding box. See Table Dimension Diagram.

- int **wiw**

  Table widget's inner w dimension, inside bounding box. See Table Dimension Diagram.

- int **wix**

  Table widget's inner x dimension, inside bounding box. See Table Dimension Diagram.

- int **wiy**

  Table widget's inner y dimension, inside bounding box. See Table Dimension Diagram.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()

  Returns the currently active group.

- static void current (Fl_Group *g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.

- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.

- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

13.131.1 Detailed Description

A table of widgets or other content.
This is the base class for table widgets.
To be useful it must be subclassed and several virtual functions defined. Normally applications use widgets derived from this widget, and do not use this widget directly; this widget is usually too low level to be used directly by applications.
This widget does *not* handle the data in the table. The draw_cell() method must be overridden by a subclass to manage drawing the contents of the cells.
This widget can be used in several ways:

- As a custom widget; see examples/table-simple.cxx and test/table.cxx. Very optimal for even extremely large tables.
- As a table made up of a single FLTK widget instanced all over the table, simulating a numeric spreadsheet. See examples/table-spreadsheet.cxx and examples/table-spreadsheet-with-keyboard-nav.cxx. Optimal for large tables.
- As a regular container of FLTK widgets, one widget per cell. See examples/table-as-container.cxx. *Not* recommended for large tables.

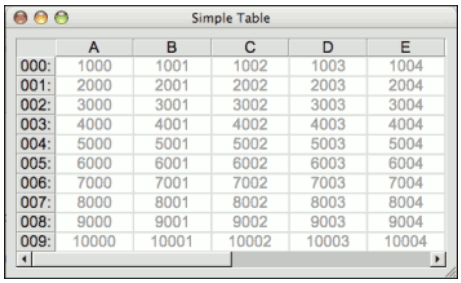


Figure 13.47 table-simple example

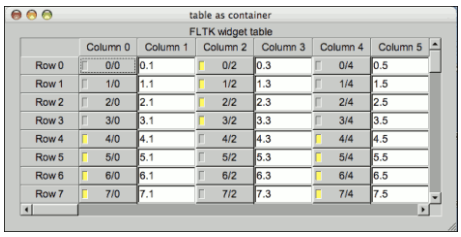


Figure 13.48 table-as-container example

When acting as part of a custom widget, events on the cells and/or headings generate callbacks when they are clicked by the user. You control when events are generated based on the setting for Fl_Table::when().
When acting as a container for FLTK widgets, the FLTK widgets maintain themselves. Although the draw_cell() method must be overridden, its contents can be very simple. See the draw_cell() code in examples/table-simple.↩cxx.
The following variables are available to classes deriving from Fl_Table:

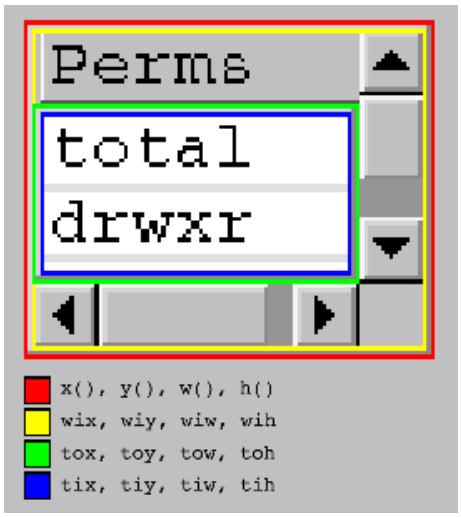


Figure 13.49 Fl_Table Dimensions

x()/y()/w()/h()	Fl_Table widget's outer dimension. The outer edge of the border of the Fl_Table. (Red in the diagram above)
wix/wiy/wiw/wih	Fl_Table widget's inner dimension. The inner edge of the border of the Fl_Table. eg. if the Fl_Table's box() is FL_NO_BOX, these values are the same as x()/y()/w()/h(). (Yellow in the diagram above)
tox/toy/tow/toh	The table's outer dimension. The outer edge of the border around the cells, but inside the row/col headings and scrollbars. (Green in the diagram above)

tix/tiy/tiw/tih	The table's inner dimension. The inner edge of the border around the cells, but inside the row/col headings and scrollbars. AKA the table's clip region. eg. if the table_box() is FL_NO_BOX, these values are the same as tox/toy/tow/toh. (Blue in the diagram above)

CORE DEVELOPERS

- Greg Ercolano : 12/16/2002 - initial implementation 12/16/02. Fl_Table, Fl_Table_Row, docs.
- Jean-Marc Lienher : 02/22/2004 - added keyboard nav + mouse selection, and ported Fl_Table into fltk-utf8-1.1.4

OTHER CONTRIBUTORS

- Inspired by the Feb 2000 version of FLVW's Flvw_Table widget. Mucho thanks to those folks.
- Mister Satan : 04/07/2003 - MinGW porting mods, and singleinput.cxx; a cool Fl_Input oriented spreadsheet example
- Marek Paliwoda : 01/08/2003 - Porting mods for Borland
- Ori Berger : 03/16/2006 - Optimizations for >500k rows/cols

LICENSE
Greg kindly gave his permission to integrate Fl_Table and Fl_Table_Row into FLTK, allowing FLTK license to apply while his widgets are part of the library. [updated by Greg, 04/26/17]

13.131.2 Member Enumeration Documentation

13.131.2.1 TableContext

`enum Fl_Table::TableContext`
The context bit flags for Fl_Table related callbacks.
Should be used in draw_cell() to determine what's being drawn, or in a callback() to determine where a recent event occurred.

Enumerator

CONTEXT_NONE	no known context
CONTEXT_STARTPAGE	before the table is redrawn
CONTEXT_ENDPAGE	after the table is redrawn
CONTEXT_ROW_HEADER	drawing or event occurred in the row header
CONTEXT_COL_HEADER	drawing or event occurred in the col header
CONTEXT_CELL	drawing or event occurred in a cell
CONTEXT_TABLE	drawing or event occurred in a dead zone of table
CONTEXT_RC_RESIZE	column or row is being resized

13.131.3 Constructor & Destructor Documentation

13.131.3.1 Fl_Table()

```
Fl_Table::Fl_Table (
            int X,
            int Y,
            int W,
            int H,
            const char * l = 0 )
```

The constructor for Fl_Table.
This creates an empty table with no rows or columns, with headers and row/column resize behavior disabled.

13.131.3.2 ~Fl_Table()

```
Fl_Table::~Fl_Table ( )
```

The destructor for Fl_Table.
Destroys the table and its associated widgets.

13.131.4 Member Function Documentation

13.131.4.1 array()

```
Fl_Widget *const * Fl_Table::array ( ) const  [inline]
```

Returns a pointer to the array of children.
This pointer is only valid until the next time a child is added or removed.

13.131.4.2 callback()

```
void Fl_Table::callback (
            Fl_Widget * ,
            void *  )
```

Callbacks will be called depending on the setting of Fl_Widget::when().
Callback functions should use the following functions to determine the context/row/column:

- Fl_Table::callback_row() returns current row
- Fl_Table::callback_col() returns current column
- Fl_Table::callback_context() returns current table context

callback_row() and callback_col() will be set to the row and column number the event occurred on. If someone clicked on a row header, `col` will be *0*. If someone clicked on a column header, `row` will be *0*.
callback_context() will return one of the following:

`Fl_Table::CONTEXT_ROW_HEADER`	Someone clicked on a row header. Excludes resizing.
`Fl_Table::CONTEXT_COL_HEADER`	Someone clicked on a column header. Excludes resizing.
`Fl_Table::CONTEXT_CELL`	Someone clicked on a cell. To receive callbacks for FL_RELEASE events, you must set when(FL_WHEN_RELEASE).
`Fl_Table::CONTEXT_RC_RESIZE`	Someone is resizing rows/columns either interactively, or via the col_width() or row_height() API. Use is_interactive_resize() to determine interactive resizing. If resizing a column, R=0 and C=column being resized. If resizing a row, C=0 and R=row being resized. NOTE: To receive resize events, you must set when(FL_WHEN↩_CHANGED).

```
class MyTable : public Fl_Table {
  [..]
private:
  // Handle events that happen on the table
  void event_callback2() {
    int R = callback_row();                        // row where event occurred
    C = callback_col();                            // column where event occurred
    TableContext context = callback_context();     // which part of table
    fprintf(stderr, "callback: Row=%d Col=%d Context=%d Event=%d\n",
            R, C, (int)context, (int)Fl::event());
  }

  // Actual static callback
  static void event_callback(Fl_Widget*, void* data) {
    MyTable *o = (MyTable*)data;
    o->event_callback2();
  }

public:
  // Constructor
  MyTable() {
    [..]
```

```
    table.callback(&event_callback, (void*)this);   // setup callback
    table.when(FL_WHEN_CHANGED|FL_WHEN_RELEASE);    // when to call it
  }
};
```

13.131.4.3 callback_col()

```
int Fl_Table::callback_col ( ) const  [inline]
```

Returns the current column the event occurred on.
This function should only be used from within the user's callback function.

13.131.4.4 callback_context()

```
TableContext Fl_Table::callback_context ( ) const  [inline]
```

Returns the current 'table context'.
This function should only be used from within the user's callback function.

13.131.4.5 callback_row()

```
int Fl_Table::callback_row ( ) const  [inline]
```

Returns the current row the event occurred on.
This function should only be used from within the user's callback function.

13.131.4.6 child()

```
Fl_Widget * Fl_Table::child (
            int n ) const  [inline]
```

Returns the child widget by an index.
When using the Fl_Table as a container for FLTK widgets, this method returns the widget pointer from the internal array of widgets in the container.
Typically used in loops, eg:

```
for ( int i=0; i<children(); i++ ) {
  Fl_Widget *w = child(i);
  [..]
}
```

13.131.4.7 children()

```
int Fl_Table::children ( ) const  [inline]
```

Returns the number of children in the table.
When using the Fl_Table as a container for FLTK widgets, this method returns how many child widgets the table has.

See also

child(int)

13.131.4.8 clear()

```
virtual void Fl_Table::clear ( )  [inline], [virtual]
```

Clears the table to zero rows (rows(0)), zero columns (cols(0)), and clears any widgets (table->clear()) that were added with begin()/end() or add()/insert()/etc.

See also

rows(int), cols(int)

Reimplemented in Fl_Table_Row.

13.131.4.9 col_header()

```
void Fl_Table::col_header (
            int flag )  [inline]
```

Enable or disable column headers.
If changed, the table is redrawn.

13.131.4.10 col_resize()

```
void Fl_Table::col_resize (
            int flag )  [inline]
```

Allows/disallows column resizing by the user.
1=allow interactive resizing, 0=disallow interactive resizing. Since interactive resizing is done via the column headers, `col_header()` must also be enabled to allow resizing.

13.131.4.11 col_resize_min()

```
void Fl_Table::col_resize_min (
            int val )  [inline]
```

Sets the current column minimum resize value.
This is used to prevent the user from interactively resizing any column to be smaller than 'pixels'. Must be a value >=1.

13.131.4.12 col_width()

```
void Fl_Table::col_width (
            int col,
            int width )
```

Sets the width of the specified column in pixels, and the table is redrawn.
callback() will be invoked with CONTEXT_RC_RESIZE if the column's width was actually changed, and when() is FL_WHEN_CHANGED.

13.131.4.13 col_width_all()

```
void Fl_Table::col_width_all (
            int width )  [inline]
```

Convenience method to set the width of all columns to the same value, in pixels.
The screen is redrawn.

13.131.4.14 cursor2rowcol()

```
Fl_Table::TableContext Fl_Table::cursor2rowcol (
            int & R,
            int & C,
            ResizeFlag & resizeflag )  [protected]
```

Find row/col for the recent mouse event.
Returns the context, and the row/column values in R/C. Also returns 'resizeflag' if mouse is hovered over a resize boundary.

13.131.4.15 damage_zone()

```
void Fl_Table::damage_zone (
            int r1,
            int c1,
            int r2,
            int c2,
            int r3 = 0,
            int c3 = 0 )  [protected]
```

Sets the damage zone to the specified row/col values.
Calls redraw_range().

13.131.4.16 do_callback()

```
void Fl_Table::do_callback (
            TableContext context,
            int row,
            int col )  [inline]
```

Calls the widget callback.
Saves the specified 'context', 'row', and 'col' values, so that the user's callback can then access them with the member functions callback_context(), callback_row() and callback_col().

13.131.4.17 draw()

```
void Fl_Table::draw (
            void ) [override], [protected], [virtual]
```

Draws the entire Fl_Table.
Lets fltk widgets draw themselves first, followed by the cells via calls to draw_cell().
Reimplemented from Fl_Group.

13.131.4.18 draw_cell()

```
virtual void Fl_Table::draw_cell (
            TableContext context,
            int R = 0,
            int C = 0,
            int X = 0,
            int Y = 0,
            int W = 0,
            int H = 0 ) [inline], [protected], [virtual]
```

Subclass should override this method to handle drawing the cells.
This method will be called whenever the table is redrawn, once per cell.
Only cells that are completely (or partially) visible will be told to draw.
`context` will be one of the following:

`Fl_Table::CONTEXT_STARTPAGE`	When table, or parts of the table, are about to be redrawn. Use to initialize static data, such as font selections. R/C will be zero, X/Y/W/H will be the dimensions of the table's entire data area. (Useful for locking a database before accessing; see also visible_cells())
`Fl_Table::CONTEXT_ENDPAGE`	When table has completed being redrawn. R/C will be zero, X/Y/W/H dimensions of table's data area. (Useful for unlocking a database after accessing)
`Fl_Table::CONTEXT_ROW_HEADER`	Whenever a row header cell needs to be drawn. R will be the row number of the header being redrawn, C will be zero, X/Y/W/H will be the fltk drawing area of the row header in the window
`Fl_Table::CONTEXT_COL_HEADER`	Whenever a column header cell needs to be drawn. R will be zero, C will be the column number of the header being redrawn, X/Y/W/H will be the fltk drawing area of the column header in the window
`Fl_Table::CONTEXT_CELL`	Whenever a data cell in the table needs to be drawn. R/C will be the row/column of the cell to be drawn, X/Y/W/H will be the fltk drawing area of the cell in the window
`Fl_Table::CONTEXT_RC_RESIZE`	Whenever table or row/column is resized or scrolled, either interactively or via col_width() or row_height(). R/C/X/Y/W/H will all be zero. Useful for fltk containers that need to resize or move the child fltk widgets.

`R` and `C` will be set to the row and column number of the cell being drawn. In the case of row headers, `C` will be *0*. In the case of column headers, `R` will be *0*.
`X/Y/W/H` will be the position and dimensions of where the cell should be drawn.

In the case of custom widgets, a minimal draw_cell() override might look like the following. With custom widgets it is up to the caller to handle drawing everything within the dimensions of the cell, including handling the selection color. Note all clipping must be handled as well; this allows drawing outside the dimensions of the cell if so desired for 'custom effects'.

```
// This is called whenever Fl_Table wants you to draw a cell
void MyTable::draw_cell(TableContext context, int R=0, int C=0, int X=0, int Y=0, int W=0, int H=0) {
    static char s[40];
    sprintf(s, "%d/%d", R, C);              // text for each cell
    switch ( context ) {
       case CONTEXT_STARTPAGE:              // Fl_Table telling us it's starting to draw page
           fl_font(FL_HELVETICA, 16);
           return;

       case CONTEXT_ROW_HEADER:             // Fl_Table telling us to draw row/col headers
       case CONTEXT_COL_HEADER:
           fl_push_clip(X, Y, W, H);
           {
               fl_draw_box(FL_THIN_UP_BOX, X, Y, W, H, color());
               fl_color(FL_BLACK);
               fl_draw(s, X, Y, W, H, FL_ALIGN_CENTER);
           }
           fl_pop_clip();
           return;

       case CONTEXT_CELL:                   // Fl_Table telling us to draw cells
           fl_push_clip(X, Y, W, H);
           {
               // BG COLOR
               fl_color(is_selected(R, C) ? selection_color() : FL_WHITE);
               fl_rectf(X, Y, W, H);

               // TEXT
               fl_color(FL_BLACK);
               fl_draw(s, X, Y, W, H, FL_ALIGN_CENTER);

               // BORDER
               fl_color(FL_LIGHT2);
               fl_rect(X, Y, W, H);
           }
           fl_pop_clip();
           return;

       default:
           return;
    }
    //NOTREACHED
}
```

13.131.4.19 find_cell()

```
int Fl_Table::find_cell (
            TableContext context,
            int R,
            int C,
            int & X,
            int & Y,
            int & W,
            int & H )  [protected]
```

Find a cell's X/Y/W/H region for the specified cell in row 'R', column 'C'.

Returns

- 0 – on success, XYWH returns the region of the specified cell.
- -1 – if R or C are out of range, and X/Y/W/H will be set to zero.

13.131.4.20 get_selection()

```
void Fl_Table::get_selection (
            int & row_top,
            int & col_left,
            int & row_bot,
            int & col_right ) const
```

Gets the region of cells selected (highlighted).

Parameters

in	*row_top*	Returns the top row of selection area
in	*col_left*	Returns the left column of selection area
in	*row_bot*	Returns the bottom row of selection area
in	*col_right*	Returns the right column of selection area

13.131.4.21 handle()

```
int Fl_Table::handle (
            int e )  [override], [protected], [virtual]
```

Handle FLTK events.
Reimplemented from Fl_Group.
Reimplemented in Fl_Table_Row.

13.131.4.22 init_sizes()

```
void Fl_Table::init_sizes ( )  [inline]
```

Resets the internal array of widget sizes and positions.

See also

Fl_Group::init_sizes()

13.131.4.23 insert()

```
void Fl_Table::insert (
            Fl_Widget & wgt,
            Fl_Widget * w2 )  [inline]
```

The specified widget is removed from its current group (if any) and inserted into Fl_Table's group before widget 'w2'. This will append if 'w2' is not in Fl_Table's group.

13.131.4.24 is_interactive_resize()

```
int Fl_Table::is_interactive_resize ( ) const  [inline]
```

Returns 1 if someone is interactively resizing a row or column.
You can currently call this only from within your callback().

13.131.4.25 is_selected()

```
int Fl_Table::is_selected (
            int r,
            int c ) const
```

See if the cell at row `r` and column `c` is selected.

Returns

1 if the cell is selected, 0 if not.

13.131.4.26 move_cursor()

```
int Fl_Table::move_cursor (
            int R,
            int C,
            int shiftselect )
```

Moves the selection cursor a relative number of rows/columns specifed by R/C.
R/C can be positive or negative, depending on the direction to move. A value of 0 for R or C prevents cursor movement on that axis.

If shiftselect is set, the selection range is extended to the new cursor position. If clear, the cursor is simply moved, and any previous selection is cancelled.
Used mainly by keyboard events (e.g. Fl_Right, FL_Home, FL_End..) to let the user keyboard navigate the selection cursor around.
The scroll positions may be modified if the selection cursor traverses into cells off the screen's edge.
Internal variables select_row/select_col and current_row/current_col are modified, among others.

```
Examples:
    R=1, C=0  -- moves the selection cursor one row downward.
    R=5, C=0  -- moves the selection cursor 5 rows downward.
    R=-5, C=0 -- moves the cursor 5 rows upward.
    R=2, C=2  -- moves the cursor 2 rows down and 2 columns to the right.
```

13.131.4.27 recalc_dimensions()

```
void Fl_Table::recalc_dimensions ( )  [protected]
```

Recalculate the dimensions of the table, and affect any children.
Internally, Fl_Group::resize() and init_sizes() are called.

13.131.4.28 redraw_range()

```
void Fl_Table::redraw_range (
            int topRow,
            int botRow,
            int leftCol,
            int rightCol )  [inline], [protected]
```

Define region of cells to be redrawn by specified range of rows/cols, and then sets damage(DAMAGE_CHILD).
Extends any previously defined range to redraw.

13.131.4.29 resize()

```
void Fl_Table::resize (
            int X,
            int Y,
            int W,
            int H )  [override], [virtual]
```

Handle resize events if user resizes parent window.
This changes the size of Fl_Table, causing it to redraw.
Reimplemented from Fl_Group.

13.131.4.30 row_col_clamp()

```
int Fl_Table::row_col_clamp (
            TableContext context,
            int & R,
            int & C )  [protected]
```

Return specified row/col values R and C to within the table's current row/col limits.

Returns

0 if no changes were made, or 1 if they were.

13.131.4.31 row_header()

```
void Fl_Table::row_header (
            int flag )  [inline]
```

Enables/disables showing the row headers.
1=enabled, 0=disabled. If changed, the table is redrawn.

13.131.4.32 row_height()

```
void Fl_Table::row_height (
            int row,
            int height )
```

Sets the height of the specified row in pixels, and the table is redrawn.
callback() will be invoked with CONTEXT_RC_RESIZE if the row's height was actually changed, and when() is FL_WHEN_CHANGED.

13.131.4.33 row_height_all()

```
void Fl_Table::row_height_all (
            int height )  [inline]
```

Convenience method to set the height of all rows to the same value, in pixels.
The screen is redrawn.

13.131.4.34 row_resize()

```
void Fl_Table::row_resize (
            int flag )  [inline]
```

Allows/disallows row resizing by the user.
1=allow interactive resizing, 0=disallow interactive resizing. Since interactive resizing is done via the row headers, row_header() must also be enabled to allow resizing.

13.131.4.35 row_resize_min()

```
void Fl_Table::row_resize_min (
            int val )  [inline]
```

Sets the current row minimum resize value.
This is used to prevent the user from interactively resizing any row to be smaller than 'pixels'. Must be a value >=1.

13.131.4.36 rows()

```
void Fl_Table::rows (
            int val )  [virtual]
```

Sets the number of rows in the table, and the table is redrawn.
Reimplemented in Fl_Table_Row.

13.131.4.37 scrollbar_size() [1/2]

```
int Fl_Table::scrollbar_size ( ) const  [inline]
```

Gets the current size of the scrollbars' troughs, in pixels.
If this value is zero (default), this widget will use the Fl::scrollbar_size() value as the scrollbar's width.

Returns

Scrollbar size in pixels, or 0 if the global Fl::scrollbar_size() is being used.

See also

Fl::scrollbar_size(int)

13.131.4.38 scrollbar_size() [2/2]

```
void Fl_Table::scrollbar_size (
            int newSize )  [inline]
```

Sets the pixel size of the scrollbars' troughs to `newSize`, in pixels.
Normally you should not need this method, and should use Fl::scrollbar_size(int) instead to manage the size of ALL your widgets' scrollbars. This ensures your application has a consistent UI, is the default behavior, and is normally what you want.

Only use THIS method if you really need to override the global scrollbar size. The need for this should be rare. Setting `newSize` to the special value of 0 causes the widget to track the global Fl::scrollbar_size(), which is the default.

Parameters

in	*newSize*	Sets the scrollbar size in pixels. If 0 (default), scrollbar size tracks the global Fl::scrollbar_size()

See also

Fl::scrollbar_size()

13.131.4.39 set_selection()

```
void Fl_Table::set_selection (
            int row_top,
            int col_left,
            int row_bot,
            int col_right )
```

Sets the region of cells to be selected (highlighted).
So for instance, set_selection(0,0,0,0) selects the top/left cell in the table. And set_selection(0,0,1,1) selects the four cells in rows 0 and 1, column 0 and 1.
To deselect all cells, use set_selection(-1,-1,-1,-1);

Parameters

in	*row_top*	Top row of selection area
in	*col_left*	Left column of selection area
in	*row_bot*	Bottom row of selection area
in	*col_right*	Right column of selection area

13.131.4.40 tab_cell_nav() [1/2]

```
int Fl_Table::tab_cell_nav ( ) const  [inline]
```

Get state of table's 'Tab' key cell navigation flag.

Returns

1 if Tab configured to navigate cells in table
0 to navigate widget focus (default)

See also

tab_cell_nav(int)

13.131.4.41 tab_cell_nav() [2/2]

```
void Fl_Table::tab_cell_nav (
            int val )  [inline]
```

Flag to control if Tab navigates table cells or not.
If on, Tab key navigates table cells. If off, Tab key navigates fltk widget focus. (default)
As of fltk 1.3, the default behavior of the Tab key is to navigate focus off of the current widget, and on to the next one. But in some applications, it's useful for Tab to be used to navigate cells in the Fl_Table.

Parameters

in	*val*	If `val` is 1, Tab key navigates cells in table, not fltk widgets. If `val` is 0, Tab key will advance focus to the next fltk widget (default), and does not navigate cells in table.

13.131.4.42 table_box()

```
void Fl_Table::table_box (
            Fl_Boxtype val )  [inline]
```

Sets the kind of box drawn around the data table, the default being FL_NO_BOX.
Changing this value will cause the table to redraw.

13.131.4.43 table_resized()

```
void Fl_Table::table_resized ( )  [protected]
```

Call this if table was resized, to recalculate internal data.
Calls recall_dimensions(), and recalculates scrollbar sizes.

13.131.4.44 table_scrolled()

```
void Fl_Table::table_scrolled ( )  [protected]
```

Recalculate internals after a scroll.
Call this if table has been scrolled or resized. Does not handle redraw(). TODO: Assumes ti[xywh] has already been recalculated.

13.131.4.45 top_row() [1/2]

```
int Fl_Table::top_row ( ) const  [inline]
```

Returns the current top row shown in the table.
This row may be partially obscured.

13.131.4.46 top_row() [2/2]

```
void Fl_Table::top_row (
            int row )  [inline]
```

Sets which row should be at the top of the table, scrolling as necessary, and the table is redrawn.
If the table cannot be scrolled that far, it is scrolled as far as possible.

13.131.4.47 visible_cells()

```
void Fl_Table::visible_cells (
            int & r1,
            int & r2,
            int & c1,
            int & c2 ) const  [inline]
```

Returns the range of row and column numbers for all visible and partially visible cells in the table.
These values can be used e.g. by your draw_cell() routine during CONTEXT_STARTPAGE to figure out what cells are about to be redrawn for the purposes of locking the data from a database before it's drawn.

```
       leftcol             rightcol
          :                   :
toprow .. .-------------------.
          |                   |
          |  V I S I B L E    |
          |                   |
          |    T A B L E      |
          |                   |
botrow .. '-------------------`
```

e.g. in a table where the visible rows are 5-20, and the visible columns are 100-120, then those variables would be:

- toprow = 5

- botrow = 20
- leftcol = 100
- rightcol = 120

13.131.4.48 when()

```
void Fl_Table::when (
            Fl_When flags )
```

The Fl_Widget::when() function is used to set a group of flags, determining when the widget callback is called:

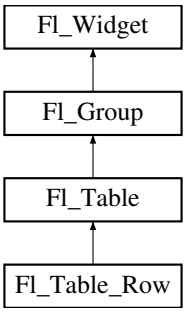
FL_WHEN_CHANGED	callback() will be called when rows or columns are resized (interactively or via col_width() or row_height()), passing CONTEXT_RC_RESIZE via callback_context().
FL_WHEN_RELEASE	callback() will be called during FL_RELEASE events, such as when someone releases a mouse button somewhere on the table.

The callback() routine is sent a TableContext that indicates the context the event occurred in, such as in a cell, in a header, or elsewhere on the table. When an event occurs in a cell or header, callback_row() and callback_col() can be used to determine the row and column. The callback can also look at the regular fltk event values (ie. Fl::event() and Fl::event_button()) to determine what kind of event is occurring.
The documentation for this class was generated from the following files:

- Fl_Table.H
- Fl_Table.cxx

13.132 Fl_Table_Row Class Reference

A table with row selection capabilities.
`#include <Fl_Table_Row.H>`
Inheritance diagram for Fl_Table_Row:



Public Types

- enum **TableRowSelectMode** { **SELECT_NONE** , **SELECT_SINGLE** , **SELECT_MULTI** }

Public Types inherited from Fl_Table

- enum TableContext {
  CONTEXT_NONE = 0 , CONTEXT_STARTPAGE = 0x01 , CONTEXT_ENDPAGE = 0x02 , CONTEXT_ROW_HEADER = 0x04 ,
  CONTEXT_COL_HEADER = 0x08 , CONTEXT_CELL = 0x10 , CONTEXT_TABLE = 0x20 , CONTEXT_RC_RESIZE = 0x40 }

  The context bit flags for Fl_Table related callbacks.

Public Member Functions

- void clear () override

  Clears the table to zero rows (rows(0)), zero columns (cols(0)), and clears any widgets (table->clear()) that were added with begin()/end() or add()/insert()/etc.
- Fl_Table_Row (int X, int Y, int W, int H, const char *l=0)

  The constructor for the Fl_Table_Row.
- int row_selected (int row)

  Checks to see if 'row' is selected.
- int **rows** ()
- void rows (int val) override

  Sets the number of rows in the table, and the table is redrawn.
- void select_all_rows (int flag=1)

  This convenience function changes the selection state for all rows based on 'flag'.
- int select_row (int row, int flag=1)

  Changes the selection state for `'row'`, depending on the value of `'flag'`.
- TableRowSelectMode **type** () const
- void type (TableRowSelectMode val)

  Sets the table selection mode.
- ~Fl_Table_Row ()

  The destructor for the Fl_Table_Row.

Public Member Functions inherited from Fl_Table

- void **add** (Fl_Widget &wgt)

  The specified widget is removed from its current group (if any) and added to the end of Fl_Table's group.
- void **add** (Fl_Widget *wgt)

  The specified widget is removed from its current group (if any) and added to the end of Fl_Table's group.
- Fl_Widget *const * array () const

  Returns a pointer to the array of children.
- void **begin** ()
- void callback (Fl_Widget *, void *)

  Callbacks will be called depending on the setting of Fl_Widget::when().
- int callback_col () const

  Returns the current column the event occurred on.
- TableContext callback_context () const

  Returns the current 'table context'.
- int callback_row () const

  Returns the current row the event occurred on.
- Fl_Widget * child (int n) const

  Returns the child widget by an index.
- int children () const

  Returns the number of children in the table.
- int **col_header** () const

  Returns if column headers are enabled or not.
- void col_header (int flag)

  Enable or disable column headers.
- Fl_Color **col_header_color** () const

  Gets the color for column headers.
- void **col_header_color** (Fl_Color val)

  Sets the color for column headers and redraws the table.
- int **col_header_height** () const

Gets the column header height.

- void **col_header_height** (int height)

  Sets the height in pixels for column headers and redraws the table.
- int **col_position** () const

  Returns the current column scroll position as a column number.
- void **col_position** (int col)

  Sets the horizontal scroll position so 'col' is at the left, and causes the screen to redraw.
- int **col_resize** () const

  Returns if column resizing by the user is allowed.
- void col_resize (int flag)

  Allows/disallows column resizing by the user.
- int **col_resize_min** () const

  Returns the current column minimum resize value.
- void col_resize_min (int val)

  Sets the current column minimum resize value.
- int **col_width** (int col) const

  Returns the current width of the specified column in pixels.
- void col_width (int col, int width)

  Sets the width of the specified column in pixels, and the table is redrawn.
- void col_width_all (int width)

  Convenience method to set the width of all columns to the same value, in pixels.
- int **cols** () const

  Get the number of columns in the table.
- virtual void **cols** (int val)

  Set the number of columns in the table and redraw.
- void do_callback (TableContext context, int row, int col)

  Calls the widget callback.
- void **end** ()
- int **find** (const Fl_Widget &wgt) const
- int **find** (const Fl_Widget *wgt) const
- Fl_Table (int X, int Y, int W, int H, const char *l=0)

  The constructor for Fl_Table.
- void get_selection (int &row_top, int &col_left, int &row_bot, int &col_right) const

  Gets the region of cells selected (highlighted).
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &wgt, Fl_Widget *w2)

  The specified widget is removed from its current group (if any) and inserted into Fl_Table's group before widget 'w2'.
- void **insert** (Fl_Widget &wgt, int n)

  The specified widget is removed from its current group (if any) and inserted into the Fl_Table's group at position 'n'.
- int is_interactive_resize () const

  Returns 1 if someone is interactively resizing a row or column.
- int is_selected (int r, int c) const

  See if the cell at row `r` and column `c` is selected.
- int **move_cursor** (int R, int C)

  Same as move_cursor(R,C,1);.
- int move_cursor (int R, int C, int shiftselect)

  Moves the selection cursor a relative number of rows/columns specifed by R/C.
- void **remove** (Fl_Widget &wgt)

  The specified widget is removed from Fl_Table's group.
- void resize (int X, int Y, int W, int H) override

Handle resize events if user resizes parent window.

- int **row_header** () const

  Returns if row headers are enabled or not.
- void row_header (int flag)

  Enables/disables showing the row headers.
- Fl_Color **row_header_color** () const

  Returns the current row header color.
- void **row_header_color** (Fl_Color val)

  Sets the row header color and causes the screen to redraw.
- int **row_header_width** () const

  Returns the current row header width (in pixels).
- void **row_header_width** (int width)

  Sets the row header width to n and causes the screen to redraw.
- int **row_height** (int row) const

  Returns the current height of the specified row as a value in pixels.
- void row_height (int row, int height)

  Sets the height of the specified row in pixels, and the table is redrawn.
- void row_height_all (int height)

  Convenience method to set the height of all rows to the same value, in pixels.
- int **row_position** () const

  Returns the current row scroll position as a row number.
- void **row_position** (int row)

  Sets the vertical scroll position so 'row' is at the top, and causes the screen to redraw.
- int **row_resize** () const

  Returns if row resizing by the user is allowed.
- void row_resize (int flag)

  Allows/disallows row resizing by the user.
- int **row_resize_min** () const

  Returns the current row minimum resize value.
- void row_resize_min (int val)

  Sets the current row minimum resize value.
- int **rows** () const

  Returns the number of rows in the table.
- int scrollbar_size () const

  Gets the current size of the scrollbars' troughs, in pixels.
- void scrollbar_size (int newSize)

  Sets the pixel size of the scrollbars' troughs to `newSize`, in pixels.
- void set_selection (int row_top, int col_left, int row_bot, int col_right)

  Sets the region of cells to be selected (highlighted).
- int tab_cell_nav () const

  Get state of table's 'Tab' key cell navigation flag.
- void tab_cell_nav (int val)

  Flag to control if Tab navigates table cells or not.
- void table_box (Fl_Boxtype val)

  Sets the kind of box drawn around the data table, the default being FL_NO_BOX.
- Fl_Boxtype **table_box** (void) const

  Returns the current box type used for the data table.
- int top_row () const

  Returns the current top row shown in the table.
- void top_row (int row)

  Sets which row should be at the top of the table, scrolling as necessary, and the table is redrawn.

- void visible_cells (int &r1, int &r2, int &c1, int &c2) const

  Returns the range of row and column numbers for all visible and partially visible cells in the table.
- void when (Fl_When flags)

  The Fl_Widget::when() function is used to set a group of flags, determining when the widget callback is called:
- ~Fl_Table ()

  The destructor for Fl_Table.

Public Member Functions inherited from Fl_Group

- Fl_Widget *& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget *o)

  See void Fl_Group::add(Fl_Widget &w)
- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget *const * array () const

  Returns a pointer to the internal array of children.
- Fl_Group const * as_group () const override
- Fl_Group * as_group () override

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const

  Returns the n'th child.
- int **children** () const

  Returns how many child widgets the group has.
- void clear ()

  Deletes all child widgets from memory recursively.
- unsigned int clip_children () const

  Returns the current clipping mode.
- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)

  Removes the widget at `index` from the group and deletes it.
- void end ()

  Exactly the same as current(this->parent()).
- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*
- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.
- void **forms_end** ()

  This is for forms compatibility only.
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

This does insert(w, find(before)).

- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.
- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.
- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

Sets the box type for the widget.

- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.

- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

Sets the label color.

- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.

- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

Gets the widget position in its window.

- int y () const

  Gets the widget position in its window.

- virtual ~Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- int **find_cell** (TableContext context, int R, int C, int &X, int &Y, int &W, int &H)
- int handle (int event) override

  Handle FLTK events.

Protected Member Functions inherited from Fl_Table

- void **change_cursor** (Fl_Cursor newcursor)

  Change mouse cursor to different type.

- long **col_scroll_position** (int col)

  Returns the scroll position (in pixels) of the specified column 'col'.

- TableContext cursor2rowcol (int &R, int &C, ResizeFlag &resizeflag)

  Find row/col for the recent mouse event.

- void damage_zone (int r1, int c1, int r2, int c2, int r3=0, int c3=0)

  Sets the damage zone to the specified row/col values.

- void draw () override

  Draws the entire Fl_Table.

- virtual void draw_cell (TableContext context, int R=0, int C=0, int X=0, int Y=0, int W=0, int H=0)

  Subclass should override this method to handle drawing the cells.

- int find_cell (TableContext context, int R, int C, int &X, int &Y, int &W, int &H)

  Find a cell's X/Y/W/H region for the specified cell in row 'R', column 'C'.

- void **get_bounds** (TableContext context, int &X, int &Y, int &W, int &H)

  Returns the (X,Y,W,H) bounding region for the specified 'context'.

- int **is_fltk_container** () const

  Does the table contain any child fltk widgets?

- void recalc_dimensions ()

  Recalculate the dimensions of the table, and affect any children.

- void redraw_range (int topRow, int botRow, int leftCol, int rightCol)

  Define region of cells to be redrawn by specified range of rows/cols, and then sets damage(DAMAGE_CHILD).

- int row_col_clamp (TableContext context, int &R, int &C)

  Return specified row/col values R and C to within the table's current row/col limits.

- long **row_scroll_position** (int row)

  Returns the scroll position (in pixels) of the specified 'row'.

- void table_resized ()

  Call this if table was resized, to recalculate internal data.

- void table_scrolled ()

  Recalculate internals after a scroll.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()

  Returns the internal array of widget sizes and positions.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget *, int)

  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)

  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)

  Allow derived groups to act when a child widget is removed from the group.
- int * sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

Internal use only.

- void x (int v)

  Internal use only.

- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()

  Returns the currently active group.

- static void current (Fl_Group *g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.

- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.

- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Table

- enum **ResizeFlag** {
  RESIZE_NONE = 0 , **RESIZE_COL_LEFT** = 1 , **RESIZE_COL_RIGHT** = 2 , **RESIZE_ROW_ABOVE** = 3 , **RESIZE_ROW_BELOW** = 4 }

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7 ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11 ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15 ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 , AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Static Protected Member Functions inherited from Fl_Table

- static void **scroll_cb** (Fl_Widget *, void *)

  Callback for when someone moves a scrollbar.

Protected Attributes inherited from Fl_Table

- int **botrow**

  bottom row# of currently visible table on screen
- int **current_col**

  selection cursor's current column (-1 if none)
- int **current_row**

  selection cursor's current row (-1 if none)
- Fl_Scrollbar * **hscrollbar**

  child horizontal scrollbar widget
- int **leftcol**

  left column# of currently visible table on screen
- int **leftcol_scrollpos**

  precomputed scroll position for left column
- int **rightcol**

  right column# of currently visible table on screen
- int **select_col**

  extended selection column (-1 if none)
- int **select_row**

  extended selection row (-1 if none)
- Fl_Scroll * **table**

  child Fl_Scroll widget container for child fltk widgets (if any)
- int **table_h**

  table's virtual height (in pixels)
- int **table_w**

  table's virtual width (in pixels)
- int **tih**

  Data table's inner h dimension, inside bounding box. See Table Dimension Diagram.
- int **tiw**

  Data table's inner w dimension, inside bounding box. See Table Dimension Diagram.
- int **tix**

  Data table's inner x dimension, inside bounding box. See Table Dimension Diagram.
- int **tiy**

  Data table's inner y dimension, inside bounding box. See Table Dimension Diagram.
- int **toh**

  Data table's outer h dimension, outside bounding box. See Table Dimension Diagram.
- int **toprow**

  top row# of currently visible table on screen
- int **toprow_scrollpos**

  precomputed scroll position for top row
- int **tow**

  Data table's outer w dimension, outside bounding box. See Table Dimension Diagram.
- int **tox**

  Data table's outer x dimension, outside bounding box. See Table Dimension Diagram.
- int **toy**

  Data table's outer y dimension, outside bounding box. See Table Dimension Diagram.
- Fl_Scrollbar * **vscrollbar**

  child vertical scrollbar widget
- int **wih**

  Table widget's inner h dimension, inside bounding box. See Table Dimension Diagram.
- int **wiw**

Table widget's inner w dimension, inside bounding box. See Table Dimension Diagram.

- int **wix**

  Table widget's inner x dimension, inside bounding box. See Table Dimension Diagram.

- int **wiy**

  Table widget's inner y dimension, inside bounding box. See Table Dimension Diagram.

13.132.1 Detailed Description

A table with row selection capabilities.
This class implements a simple table with the ability to select rows. This widget is similar to an Fl_Browser with columns. Most methods of importance will be found in the Fl_Table widget, such as Fl_Table::rows() and Fl_Table::cols().
To be useful it must be subclassed and at minimum the draw_cell() method must be overridden to provide the content of the cells. This widget does *not* manage the cell's data content; it is up to the parent class's draw_cell() method override to provide this.
Events on the cells and/or headings generate callbacks when they are clicked by the user. You control when events are generated based on the values you supply for Fl_Table::when().

13.132.2 Constructor & Destructor Documentation

13.132.2.1 Fl_Table_Row()

```
Fl_Table_Row::Fl_Table_Row (
            int X,
            int Y,
            int W,
            int H,
            const char * l = 0 )  [inline]
```

The constructor for the Fl_Table_Row.
This creates an empty table with no rows or columns, with headers and row/column resize behavior disabled.

13.132.2.2 ~Fl_Table_Row()

```
Fl_Table_Row::~Fl_Table_Row ( )  [inline]
```

The destructor for the Fl_Table_Row.
Destroys the table and its associated widgets.

13.132.3 Member Function Documentation

13.132.3.1 clear()

```
void Fl_Table_Row::clear ( )  [inline], [override], [virtual]
```

Clears the table to zero rows (rows(0)), zero columns (cols(0)), and clears any widgets (table->clear()) that were added with begin()/end() or add()/insert()/etc.

See also

rows(int), cols(int)

Reimplemented from Fl_Table.

13.132.3.2 handle()

```
int Fl_Table_Row::handle (
            int e )  [override], [protected], [virtual]
```

Handle FLTK events.
Reimplemented from Fl_Table.

13.132.3.3 row_selected()

```
int Fl_Table_Row::row_selected (
            int row )
```

Checks to see if 'row' is selected.
Returns 1 if selected, 0 if not. You can change the selection of a row by clicking on it, or by using select_row(row, flag)
`row` **should** be a valid row. If the row is out of range the return value is 0 (zero).

Parameters

in	*row*	row to be checked

Returns

whether given row is selected

Return values

1	row is selected
0	row is not selected or `row` is out of range

13.132.3.4 rows()

```
void Fl_Table_Row::rows (
            int val ) [override], [virtual]
```

Sets the number of rows in the table, and the table is redrawn.
Reimplemented from Fl_Table.

13.132.3.5 select_all_rows()

```
void Fl_Table_Row::select_all_rows (
            int flag = 1 )
```

This convenience function changes the selection state for *all* rows based on 'flag'.
0=deselect, 1=select, 2=toggle existing state.

13.132.3.6 select_row()

```
int Fl_Table_Row::select_row (
            int row,
            int flag = 1 )
```

Changes the selection state for `'row'`, depending on the value of `'flag'`.
The optional `flag` can be:

- 0: clear selection
- 1: set selection (default)
- 2: toggle selection

Parameters

in	*row*	row to be selected, deselected, or toggled
in	*flag*	change mode, see description

Returns

result of modification, see description

Return values

0	selection state did not change
1	selection state changed
-1	row out of range or incorrect selection mode (`flag`)

13.132.3.7 type()

```
void Fl_Table_Row::type (
            TableRowSelectMode val )
```

Sets the table selection mode.

- `Fl_Table_Row::SELECT_NONE` - No selection allowed
- `Fl_Table_Row::SELECT_SINGLE` - Only single rows can be selected
- `Fl_Table_Row::SELECT_MULTI` - Multiple rows can be selected

The documentation for this class was generated from the following files:

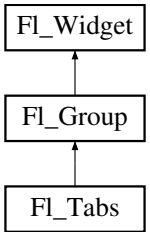
- Fl_Table_Row.H
- Fl_Table_Row.cxx

13.133 Fl_Tabs Class Reference

The Fl_Tabs widget is a container widget that displays a set of tabs, with each tab representing a different child widget.

`#include <Fl_Tabs.H>`

Inheritance diagram for Fl_Tabs:



Public Types

- enum { OVERFLOW_COMPRESS = 0 , OVERFLOW_CLIP , OVERFLOW_PULLDOWN , OVERFLOW_DRAG }

Public Member Functions

- void client_area (int &rx, int &ry, int &rw, int &rh, int tabh=0)

  Returns the position and size available to be used by its children.
- Fl_Tabs (int X, int Y, int W, int H, const char *L=0)

  Creates a new Fl_Tabs widget using the given position, size, and label string.
- int handle (int) override

  Handle all events in the tabs area and forward the rest to the selected child.
- void handle_overflow (int ov)

Set a method to handle an overflowing tab bar.

- Fl_Widget * push () const

  Returns the tab group for the tab the user has currently down-clicked on and remains over until FL_RELEASE.
- int push (Fl_Widget *)

  This is called by the tab widget's handle() method to set the tab group widget the user last FL_PUSH'ed on.
- void resize (int, int, int, int) override

  Make sure that we redraw all tabs when the widget size changes.
- void show () override

  Ensure proper placement of selected tab.
- Fl_Align tab_align () const

  Gets the tab label alignment.
- void tab_align (Fl_Align a)

  Sets the tab label alignment.
- Fl_Widget * value ()

  Gets the currently visible widget/tab.
- int value (Fl_Widget *)

  Sets the widget to become the currently visible widget/tab.
- int value (int)

  Sets the widget to become the currently visible tab.
- virtual Fl_Widget * which (int event_x, int event_y)

  Return a pointer to the child widget with a tab at the given coordinates.
- ~**Fl_Tabs** () override

  Delete allocated resources and destroy all children.

Public Member Functions inherited from Fl_Group

- Fl_Widget *& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget *o)

  See void Fl_Group::add(Fl_Widget &w)
- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget *const * array () const

  Returns a pointer to the internal array of children.
- Fl_Group const * as_group () const override
- Fl_Group * as_group () override

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const

  Returns the n'th child.
- int **children** () const

  Returns how many child widgets the group has.
- void clear ()

  Deletes all child widgets from memory recursively.
- unsigned int clip_children () const

  Returns the current clipping mode.
- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.

- virtual int delete_child (int n)

  Removes the widget at `index` from the group and deletes it.
- void end ()

  Exactly the same as current(this->parent()).
- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*
- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.
- void **forms_end** ()

  This is for forms compatibility only.
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

  This does insert(w, find(before)).
- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.
- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.
- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.

- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

Checks if w is a child of this widget.

- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.

- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

Sets the selection color.

- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.

- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void **check_overflow_menu** ()

  Check if the tabs overflow and sets the has_overflow_menu flag accordingly.
- virtual void clear_tab_positions ()

  Clear internal array of tab positions and widths.
- void draw () override

  Draw the tabs area, the optional pulldown button, and all children.
- void **draw_overflow_menu_button** ()

  Draw square button-like graphics with a down arrow in the top or bottom right corner.
- virtual void draw_tab (int x1, int x2, int W, int H, Fl_Widget ∗o, int flags, int sel)

  Draw a tab in the top or bottom tabs area.
- void handle_overflow_menu ()

  This is called when the user clicks the overflow pulldown menu button.
- virtual int hit_close (Fl_Widget ∗o, int event_x, int event_y)

  Check whether the coordinates fall within the "close" button area of the tab.
- virtual int hit_overflow_menu (int event_x, int event_y)

  Determine if the coordinates are in the area of the overflow menu button.
- virtual int hit_tabs_area (int event_x, int event_y)

  Determine if the coordinates are within the tabs area.
- int maybe_do_callback (Fl_Widget ∗o)

  Set tab o as selected and call callbacks if needed.
- int on_insert (Fl_Widget ∗, int) override

  Make sure that we redraw all tabs when new children are added.
- int on_move (int, int) override

  Make sure that we redraw all tabs when children are moved.
- void on_remove (int) override

  Make sure that we redraw all tabs when new children are removed.
- virtual void redraw_tabs ()

  Redraw all tabs (and only the tabs).

- virtual int tab_height ()

  Return space (height) in pixels usable for tabs.
- virtual int tab_positions ()

  Calculate tab positions and widths.
- void take_focus (Fl_Widget *o)

  Take keyboard focus if o is not NULL.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()

  Returns the internal array of widget sizes and positions.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- int * sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

Sets a flag in the flags mask.

- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Protected Attributes

- int **has_overflow_menu**

  set in OVERFLOW_PULLDOWN mode if tabs overflow. The actual menu array is created only on demand
- int overflow_type
- Fl_Align **tab_align_**

  tab label alignment
- int tab_count

  Array size of tab positions etc.
- int * tab_flags

  Array of tab flag of tabs per child.
- int **tab_offset**

  for pulldown and drag overflow, this is the horizontal offset when the tabs bar is dragged by the user
- int * tab_pos

  Array of x-offsets of tabs per child + 1.
- int * tab_width

  Array of widths of tabs per child.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()

  Returns the currently active group.
- static void current (Fl_Group *g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,

GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU = 1<<19 ,
NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 , AUTO_DELETE_USER_DATA = 1<<23 ,
MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 , USERFLAG1 = 1<<31 }

flags possible values enumeration.

13.133.1 Detailed Description

The Fl_Tabs widget is a container widget that displays a set of tabs, with each tab representing a different child widget.
The user can select a tab by clicking on it, and the corresponding child widget will be displayed. The Fl_Tabs widget is useful for organizing a large number of controls or other widgets into a compact space, allowing the user to switch between different sets of controls as needed.

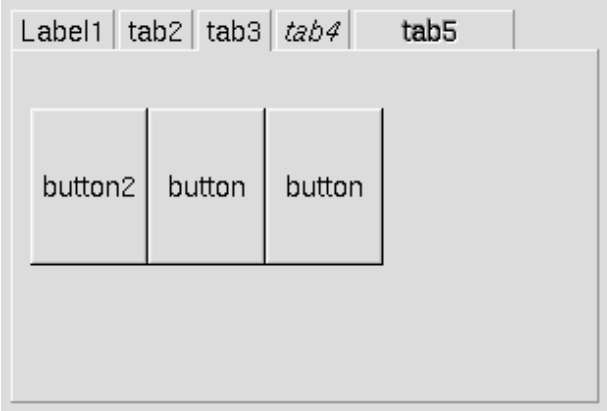


Figure 13.50 Fl_Tabs

Clicking the tab makes a child visible() by calling show() on it, and all other children are made invisible by calling hide() on them. Usually the children are Fl_Group widgets containing several widgets themselves.
Each child makes a card, and its label() is printed on the card tab, including the label font and style. The selection color of that child is used to color the tab, while the color of the child determines the background color of the pane. '&' in labels are used to prefix a shortcut that is drawn underlined and that activates the corresponding tab; repeated '&&' avoids that.
The size of the tabs is controlled by the bounding box of the children (there should be some space between the children and the edge of the Fl_Tabs), and the tabs may be placed "inverted" on the bottom - this is determined by which gap is larger. It is easiest to lay this out in FLUID, using the FLUID browser to select each child group and resize them until the tabs look the way you want them to.
Note: The widgets contained in each child should leave some clear space (five pixels as of FLTK 1.4.x) at the top or bottom of the group (where the tabs are displayed). Otherwise drawing the children may interfere with the selection border between the tabs and the children. This is particularly important if the child group is an Fl_Scroll widget: either the Fl_Scroll widget must be inset by five pixels relative to other children or it can be wrapped inside another Fl_Group and inset by five pixels within this group so the contents of the Fl_Scroll widget are kept away from the tabs by this amount.
The background area behind and to the right of the tabs is "transparent", exposing the background detail of the parent. The value of Fl_Tabs::box() does not affect this area. So if Fl_Tabs is resized by itself without the parent, force the appropriate parent (visible behind the tabs) to redraw() to prevent artifacts.
See "Resizing Caveats" below on how to keep tab heights constant. See "Callback's Use Of when()" on how to control the details of how clicks invoke the callback().
A typical use of the Fl_Tabs widget:

```
// Typical use of Fl_Tabs
Fl_Tabs *tabs = new Fl_Tabs(10,10,300,200);
{
    Fl_Group *grp1 = new Fl_Group(20,30,280,170,"Tab1");
    {
        ..widgets that go in tab#1..
    }
    grp1->end();
    Fl_Group *grp2 = new Fl_Group(20,30,280,170,"Tab2");
    {
        ..widgets that go in tab#2..
    }
    grp2->end();
}
tabs->end();
```

Default Appearance

The appearance of each "tab" is taken from the label() and color() of the child group corresponding to that "tab" and panel. Where the "tabs" appear depends on the position and size of the child groups that make up the panels within the Fl_Tabs widget, i.e. whether there is more space above or below them. The height of the "tabs" depends on how much free space is available.

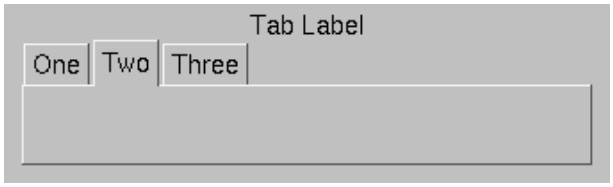


Figure 13.51 Fl_Tabs Default Appearance

Highlighting The Selected Tab

The selected "tab" can be highlighted further by setting the selection_color() of the Fl_Tab itself, e.g.

```
..
tabs = new Fl_Tabs(..);
tabs->selection_color(FL_DARK3);
..
```

The result of the above looks like:

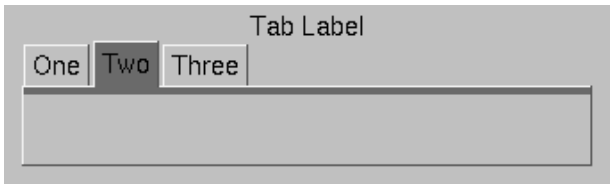


Figure 13.52 Highlighting the selected tab

Uniform Tab and Panel Appearance

In order to have uniform tab and panel appearance, not only must the color() and selection_color() for each child group be set, but also the selection_color() of the Fl_Tab itself any time a new "tab" is selected. This can be achieved within the Fl_Tab callback, e.g.

```
void MyTabCallback(Fl_Widget *w, void*) {
  Fl_Tabs *tabs = (Fl_Tabs*)w;
  // When tab changed, make sure it has same color as its group
  tabs->selection_color( (tabs->value())->color() );
}
..
int main(..) {
  // Define tabs widget
  tabs = new Fl_Tabs(..);
  tabs->callback(MyTabCallback);

  // Create three tabs each colored differently
  grp1 = new Fl_Group(.. "One");
   grp1->color(9);
```

```
    grp1->selection_color(9);
   grp1->end();

   grp2 = new Fl_Group(.. "Two");
    grp2->color(10);
    grp2->selection_color(10);
   grp2->end();

   grp3 = new Fl_Group(.. "Three");
    grp3->color(14);
    grp3->selection_color(14);
   grp3->end();
   ..
   // Make sure default tab has same color as its group
   tabs->selection_color( (tab->value())->color() );
   ..
   return Fl::run();
}
```

The result of the above looks like:

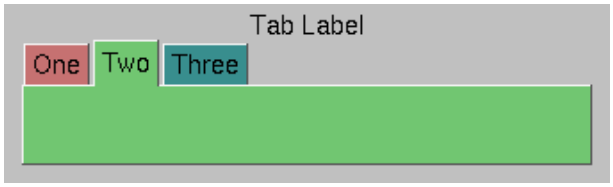


Figure 13.53 Fl_Tabs with uniform colors

If Fl_Tabs has no children, the widget will be drawn as a flat rectangle in the background color set by color().

Close Button on Tabs

The Fl_Tabs widget allows you to specify that a child widget should display a close button in its tab. If the FL_WHEN_CLOSED flag is set for the child widget, an "X" symbol will be displayed to the left of the label text in the tab. When the close button is clicked, the child widget's callback function will be called with the FL_REASON_CLOSED reason. It is then the responsibility of the child widget to remove itself from the Fl_Tabs container.

Tabs that are in a compressed state will not display a close button until they are fully expanded.

Overflowing Tabs

When the combined width of the tabs exceeds that of the Fl_Tabs widget, the tabs will overflow. Fl_Tabs provides four options for managing tabs overflow:

- Fl_Tabs::OVERFLOW_COMPRESS: proportionally compress the tabs to the left and right of the selected tab until they all fit within the widget.
- Fl_Tabs::OVERFLOW_CLIP: clip any tabs that extend beyond the right edge of the Fl_Tabs widget, making some tabs unreachable.
- Fl_Tabs::OVERFLOW_PULLDOWN: don't compress the tabs but instead generate a pulldown menu at the right end of the tabs area, displaying all available tabs.
- Fl_Tabs::OVERFLOW_DRAG: maintain the tabs' original sizes, allowing horizontal dragging of the tabs area using the mouse, a horizontal mouse wheel, or the horizontal scrolling gesture on touchpads.

Resizing Caveats

When Fl_Tabs is resized vertically, the default behavior scales the tab's height as well as its children. To keep the tab height constant during resizing, set the tab widget's resizable() to one of the tab's child groups, i.e.

```
tabs = new Fl_Tabs(..);
grp1 = new Fl_Group(..);
..
grp2 = new Fl_Group(..);
..
tabs->end();
tabs->resizable(grp1);      // keeps tab height constant
```

Callback's Use Of when()

As of FLTK 1.3.3, Fl_Tabs() supports the following flags for when():

- FL_WHEN_NEVER – callback never invoked (all flags off)
- FL_WHEN_CHANGED – if flag set, invokes callback when a tab has been changed (on click or keyboard navigation)
- FL_WHEN_NOT_CHANGED – if flag set, invokes callback when the tabs remain unchanged (on click or keyboard navigation)
- FL_WHEN_RELEASE – if flag set, invokes callback on RELEASE of mouse button or keyboard navigation

Notes:

1. The above flags can be logically OR-ed (|) or added (+) to combine behaviors.
2. The default value for when() is FL_WHEN_RELEASE (inherited from Fl_Widget).
3. If FL_WHEN_RELEASE is the *only* flag specified, the behavior will be as if (FL_WHEN_RELEASE|FL_WHEN_CHANGED) was specified.
4. The value of changed() will be valid during the callback.
5. If both FL_WHEN_CHANGED and FL_WHEN_NOT_CHANGED are specified, the callback is invoked whether the tab has been changed or not. The changed() method can be used to determine the cause.
6. FL_WHEN_NOT_CHANGED can happen if someone clicks on an already selected tab, or if a keyboard navigation attempt results in no change to the tabs, such as using the arrow keys while at the left or right end of the tabs.
7. Fl::callback_reason() returns FL_REASON_SELECTED or FL_REASON_RESELECTED

13.133.2 Member Enumeration Documentation

13.133.2.1 anonymous enum

```
anonymous enum
```

Enumerator

OVERFLOW_COMPRESS	Tabs will be compressed and overlaid on top of each other.
OVERFLOW_CLIP	Only the first tabs that fit will be displayed.
OVERFLOW_PULLDOWN	Tabs that do not fit will be placed in a pull-down menu.
OVERFLOW_DRAG	The tab bar can be dragged horizontally to reveal additional tabs.

13.133.3 Constructor & Destructor Documentation

13.133.3.1 Fl_Tabs()

```
Fl_Tabs::Fl_Tabs (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

Creates a new Fl_Tabs widget using the given position, size, and label string.
The default boxtype is FL_THIN_UP_BOX.

Use add(Fl_Widget*) to add each child, which are usually Fl_Group widgets. The children should be sized to stay away from the top or bottom edge of the Fl_Tabs widget, which is where the tabs will be drawn.
All children of Fl_Tabs should have the same size and exactly fit on top of each other. They should only leave space above or below where the tabs will go, but not on the sides. If the first child of Fl_Tabs is set to "resizable()", the riders will not resize when the tabs are resized.
The destructor *also deletes all the children*. This allows a whole tree to be deleted at once, without having to keep a pointer to all the children in the user code. A kludge has been done so the Fl_Tabs and all of its children can be automatic (local) variables, but you must declare the Fl_Tabs widget *first* so that it is destroyed last.

13.133.4 Member Function Documentation

13.133.4.1 clear_tab_positions()

```
void Fl_Tabs::clear_tab_positions ( ) [protected], [virtual]
```

Clear internal array of tab positions and widths.

See also

tab_positions().

13.133.4.2 client_area()

```
void Fl_Tabs::client_area (
            int & rx,
            int & ry,
            int & rw,
            int & rh,
            int tabh = 0 )
```

Returns the position and size available to be used by its children.
If there isn't any child yet the `tabh` parameter will be used to calculate the return values. This assumes that the children's labelsize is the same as the Fl_Tabs' labelsize and adds a small border.
If there are already children, the values of child(0) are returned, and `tabh` is ignored.

Note

Children should always use the same positions and sizes.

This function requires access to the display drivers to determine the selected font height. If `client_area` is invoked before the first window is displayed, ensure that `fl_open_display()` is called beforehand.

`tabh` can be one of

- 0: calculate label size, tabs on top
- -1: calculate label size, tabs on bottom
- > 0: use given `tabh` value, tabs on top (height = tabh)
- < -1: use given `tabh` value, tabs on bottom (height = -tabh)

Parameters

in	*tabh*	position and optional height of tabs (see above)
out	*rx,ry,rw,rh*	(x,y,w,h) of client area for children

Since

FLTK 1.3.0

13.133.4.3 draw()

```
void Fl_Tabs::draw (
            void ) [override], [protected], [virtual]
```

Draw the tabs area, the optional pulldown button, and all children.
Reimplemented from Fl_Group.

13.133.4.4 draw_tab()

```
void Fl_Tabs::draw_tab (
            int x1,
            int x2,
            int W,
            int H,
            Fl_Widget * o,
            int flags,
            int what ) [protected], [virtual]
```

Draw a tab in the top or bottom tabs area.
Tabs can be selected, or on the left or right side of the selected tab. If overlapping, left tabs are drawn bottom to top using clipping. The selected tab is then the topmost, followed by the right side tabs drawn top to bottom.
Tabs with the FL_WHEN_CLOSE bit set will draw a cross on their left side only if they are not compressed/overlapping.

Parameters

in	*x1*	horizontal position of the left visible edge of the tab
in	*x2*	horizontal position of the following tab
in	*W,H*	width and height of the tab
in	*o*	the child widget that corresponds to this tab
in	*flags*	if bit 1 is set, this tab is overlapped by another tab
in	*what*	can be LEFT, SELECTED, or RIGHT to indicate if the tab is to the left side or the right side of the selected tab, or the selected tab itself

13.133.4.5 handle()

```
int Fl_Tabs::handle (
            int event ) [override], [virtual]
```

Handle all events in the tabs area and forward the rest to the selected child.

Parameters

in	*event*	handle this event

Returns

1 if the event was handled

Reimplemented from Fl_Group.

13.133.4.6 handle_overflow()

```
void Fl_Tabs::handle_overflow (
            int ov )
```

Set a method to handle an overflowing tab bar.
The Fl_Tabs widget allows you to specify how to handle the situation where there are more tabs than can be displayed at once. The available options are:

- `OVERFLOW_COMPRESS`: Tabs will be compressed and overlaid on top of each other.
- `OVERFLOW_CLIP`: Only the first tabs that fit will be displayed.
- `OVERFLOW_PULLDOWN`: Tabs that do not fit will be placed in a pull-down menu.
- `OVERFLOW_DRAG`: The tab bar can be dragged horizontally to reveal additional tabs.

You can set the desired behavior using the overflow() method.

Parameters

ov	overflow type

See also

OVERFLOW_COMPRESS, OVERFLOW_CLIP, OVERFLOW_PULLDOWN, OVERFLOW_DRAG

13.133.4.7 handle_overflow_menu()

```
void Fl_Tabs::handle_overflow_menu ( ) [protected]
```

This is called when the user clicks the overflow pulldown menu button.
This method creates a menu item array that contains the titles of all tabs in the Fl_Tabs group. Visible and invisible tabs are separated by dividers to indicate their state.
The menu is then presented until the user selects an item or cancels. The chosen tab is then selected and made visible.
The menu item array is then deleted.

13.133.4.8 hit_close()

```
int Fl_Tabs::hit_close (
            Fl_Widget * o,
            int event_x,
            int event_y ) [protected], [virtual]
```

Check whether the coordinates fall within the "close" button area of the tab.
The Fl_Tabs::hit_close() method checks whether the given event coordinates fall within the area of the "close" button on the tab of the specified child widget. This method should be called after the Fl_Tabs::which() method, which updates a lookup table used to determine the width of each tab.

Parameters

o	check the tab of this widget
event_x,event↩_y	event coordinates

Returns

1 if we hit the close button, and 0 otherwise

13.133.4.9 hit_overflow_menu()

```
int Fl_Tabs::hit_overflow_menu (
            int event_x,
            int event_y ) [protected], [virtual]
```

Determine if the coordinates are in the area of the overflow menu button.

Parameters

event_x,event↩ _y	event coordinates

Returns

1 if we hit the overflow menu button, and 0 otherwise

13.133.4.10 hit_tabs_area()

```
int Fl_Tabs::hit_tabs_area (
            int event_x,
            int event_y )  [protected], [virtual]
```

Determine if the coordinates are within the tabs area.

Parameters

event_x,event↩ _y	event coordinates

Returns

1 if we hit the tabs area, and 0 otherwise

13.133.4.11 maybe_do_callback()

```
int Fl_Tabs::maybe_do_callback (
            Fl_Widget * o )  [protected]
```

Set tab o as selected and call callbacks if needed.

Parameters

in	*o*	the newly selected tab

Returns

0 if o is invalid or was deleted by the callback and must no longer be used

13.133.4.12 on_insert()

```
int Fl_Tabs::on_insert (
            Fl_Widget * candidate,
            int index )  [override], [protected], [virtual]
```

Make sure that we redraw all tabs when new children are added.
Reimplemented from Fl_Group.

13.133.4.13 on_move()

```
int Fl_Tabs::on_move (
            int a,
            int b )  [override], [protected], [virtual]
```

Make sure that we redraw all tabs when children are moved.
Reimplemented from Fl_Group.

13.133.4.14 on_remove()

```
void Fl_Tabs::on_remove (
            int index )  [override], [protected], [virtual]
```

Make sure that we redraw all tabs when new children are removed.
Reimplemented from Fl_Group.

13.133.4.15 push() [1/2]

```
Fl_Widget * Fl_Tabs::push ( ) const  [inline]
```

Returns the tab group for the tab the user has currently down-clicked on and remains over until FL_RELEASE. Otherwise, returns NULL.
While the user is down-clicked on a tab, the return value is the tab group for that tab. But as soon as the user releases, or drags off the tab with the button still down, the return value will be NULL.

See also

push(Fl_Widget*).

13.133.4.16 push() [2/2]

```
int Fl_Tabs::push (
            Fl_Widget * o )
```

This is called by the tab widget's handle() method to set the tab group widget the user last FL_PUSH'ed on.
Set back to zero on FL_RELEASE.
As of this writing, the value is mainly used by draw_tab() to determine whether or not to draw a 'down' box for the tab when it's clicked, and to turn it off if the user drags off it.

See also

push().

13.133.4.17 redraw_tabs()

```
void Fl_Tabs::redraw_tabs ( )  [protected], [virtual]
```

Redraw all tabs (and only the tabs).
This method sets the Fl_Tab's damage flags so the tab area is redrawn.

13.133.4.18 resize()

```
void Fl_Tabs::resize (
            int X,
            int Y,
            int W,
            int H )  [override], [virtual]
```

Make sure that we redraw all tabs when the widget size changes.
Reimplemented from Fl_Group.

13.133.4.19 show()

```
void Fl_Tabs::show ( )  [override], [virtual]
```

Ensure proper placement of selected tab.
Reimplemented from Fl_Widget.

13.133.4.20 tab_align() [1/2]

```
Fl_Align Fl_Tabs::tab_align ( ) const  [inline]
```

Gets the tab label alignment.

See also

tab_align(Fl_Align)

13.133.4.21 tab_align() [2/2]

```
void Fl_Tabs::tab_align (
            Fl_Align a ) [inline]
```

Sets the tab label alignment.
The default is FL_ALIGN_CENTER so tab labels are centered, but since the label space is measured (per label) to fit the labels, there wouldn't be any difference if labels were aligned left or right.
If you want to show an image (icon) next to the group's label you can set a different label alignment. FL_ALIGN_↩ IMAGE_NEXT_TO_TEXT is the recommended alignment to show the icon left of the text.

13.133.4.22 tab_height()

```
int Fl_Tabs::tab_height ( ) [protected], [virtual]
```

Return space (height) in pixels usable for tabs.
The calculated height is the largest space between all children and the upper and lower widget boundaries, respectively. If the space at the bottom is larger than at the top, the value will be negative and the tabs should be placed at the bottom.

Returns

Vertical space that can be used for the tabs.

Return values

>0	To put the tabs at the top of the widget.
<0	To put the tabs on the bottom.
Full	height, if children() == 0.

13.133.4.23 tab_positions()

```
int Fl_Tabs::tab_positions ( ) [protected], [virtual]
```

Calculate tab positions and widths.
This protected method calculates the horizontal display positions and widths of all tabs. If the number of children `'nc'` (see below) is > 0 three internal arrays are allocated, otherwise the arrays are free'd and the pointers are set to NULL. Note that the first array is larger (nc+1).

- tab_pos[nc+1] : The left edges of each tab plus a fake left edge for a tab past the right-hand one.
- tab_width[nc] : The width of each tab
- tab_flags[nc] : Flags, bit 0 is set if the tab is compressed

If needed, these arrays are (re)allocated.
These positions are actually of the left edge of the slope. They are either separated by the correct distance or by EXTRASPACE or by zero.
In OVERFLOW_COMPRESS mode, tab positions and widths are compressed to make the entire tabs bar fit into the width of Fl_Tabs while keeping the selected tab fully visible.
In other overflow modes, the tabs area may be dragged horizontally using tab_offset. The tab_pos array is not adjusted to the horizontal offset, but starts at this->x() plus the box's left margin.
The protected variable `tab_count` is set to the currently allocated size, i.e. the number of children (`nc`).

Returns

Index of the selected item

Return values

-1	If the number of children is 0 (zero).

Note

Return values in 1.3 were not documented. Return values before Sep 2023 were documented as 1 based index and 0 if there were no children. This was actually never the case. It always returned a 0 based index and the (useless) value of also 0 if there were no children. The current version returns -1 if there are no children.

For this method to work, only a single child should be selected. Calling the method value() before calling tab_positions() will ensure that exactly one child is selected and return a pointer to that child.

See also

clear_tab_positions()

13.133.4.24 take_focus()

```
void Fl_Tabs::take_focus (
            Fl_Widget * o )  [protected]
```

Take keyboard focus if o is not NULL.

Parameters

in	*o*	selected tab

13.133.4.25 value() [1/3]

```
Fl_Widget * Fl_Tabs::value ( )
```

Gets the currently visible widget/tab.
The Fl_Tabs::value() method returns a pointer to the currently visible child widget of the Fl_Tabs container. The visible child is the first child that is currently being displayed, or the last child if none of the children are being displayed.
If child widgets have been added, moved, or deleted, this method ensures that only one tab is visible at a time.

Returns

a pointer to the currently visible child

13.133.4.26 value() [2/3]

```
int Fl_Tabs::value (
            Fl_Widget * newvalue )
```

Sets the widget to become the currently visible widget/tab.
The Fl_Tabs::value() method allows you to set a particular child widget of the Fl_Tabs container to be the currently visible widget. If the specified widget is a child of the Fl_Tabs container, it will be made visible and all other children will be hidden. The method returns 1 if the value was changed, and 0 if the specified value was already set.

Parameters

in	*newvalue*	a poiner to a child widget

Returns

1 if a different tab was chosen

0 if there was no change (new value already set)

13.133.4.27 value() [3/3]

```
int Fl_Tabs::value (
            int ix )
```

Sets the widget to become the currently visible tab.

Parameters

in	*ix*	index of the child widget to show

Returns

1 if a different tab was chosen

0 if there was no change (new value already set, ix out of range)

13.133.4.28 which()

```
Fl_Widget * Fl_Tabs::which (
            int event_x,
            int event_y )  [virtual]
```

Return a pointer to the child widget with a tab at the given coordinates.
The Fl_Tabs::which() method returns a pointer to the child widget of the Fl_Tabs container that corresponds to the tab at the given event coordinates. If the event coordinates are outside the area of the tabs or if the Fl_Tabs container has no children, the method returns NULL.

Parameters

event_x,event_y	event coordinates

Returns

pointer to the selected child widget, or NULL

13.133.5 Member Data Documentation

13.133.5.1 overflow_type

```
int Fl_Tabs::overflow_type  [protected]
```

See also

OVERFLOW_COMPRESS, OVERFLOW_CLIP, etc.

13.133.5.2 tab_count

```
int Fl_Tabs::tab_count  [protected]
```

Array size of tab positions etc.

See also

tab_positions()

13.133.5.3 tab_flags

```
int* Fl_Tabs::tab_flags  [protected]
```

Array of tab flag of tabs per child.

See also

tab_positions()

13.133.5.4 tab_pos

`int* Fl_Tabs::tab_pos [protected]`
Array of x-offsets of tabs per child + 1.

See also

tab_positions()

13.133.5.5 tab_width

`int* Fl_Tabs::tab_width [protected]`
Array of widths of tabs per child.

See also

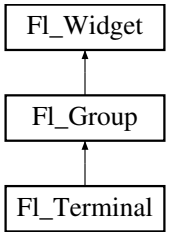
tab_positions()

The documentation for this class was generated from the following files:

- Fl_Tabs.H
- Fl_Tabs.cxx

13.134 Fl_Terminal Class Reference

Terminal widget supporting Unicode/utf-8, ANSI/xterm escape codes with full RGB color control.
`#include <Fl_Terminal.H>`
Inheritance diagram for Fl_Terminal:



Classes

- class CharStyle
- class Cursor
- class EscapeSeq
- class Margin
- class PartialUtf8Buf
- class RingBuffer
- class Selection
- class Utf8Char

Public Types

- enum Attrib {
  NORMAL = 0x00 , BOLD = 0x01 , DIM = 0x02 , ITALIC = 0x04 ,
  UNDERLINE = 0x08 , _RESERVED_1 = 0x10 , INVERSE = 0x20 , _RESERVED_2 = 0x40 ,
  STRIKEOUT = 0x80 }

Bits for the per-character attributes, which control text features such as italic, bold, underlined text, etc.

- enum CharFlags {
  FG_XTERM = 0x01 , BG_XTERM = 0x02 , EOL = 0x04 , **RESV_A** = 0x08 ,
  RESV_B = 0x10 , **RESV_C** = 0x20 , **RESV_D** = 0x40 , **RESV_E** = 0x80 ,
  COLORMASK = (FG_XTERM | BG_XTERM) }

  Per-character 8 bit flags (uchar) used to manage special states for characters.
- enum OutFlags { OFF = 0x00 , CR_TO_LF = 0x01 , LF_TO_CR = 0x02 , LF_TO_CRLF = 0x04 }

  Output translation flags for special control character translations.
- enum RedrawStyle { NO_REDRAW =0 , RATE_LIMITED , PER_WRITE }

  Determines when Fl_Terminal calls redraw() if new text is added.
- enum ScrollbarStyle { SCROLLBAR_OFF = 0x00 , SCROLLBAR_AUTO = 0x01 , SCROLLBAR_ON = 0x02 }

  Behavior of scrollbars.

Public Member Functions

- void ansi (bool val)

  Enable/disable the ANSI mode flag.
- bool ansi (void) const

  Return the state of the ANSI flag.
- void append (const char *s, int len=-1)

  Appends string `s` to the terminal at the current cursor position using the current text color/attributes.
- void append_ascii (const char *s)

  Append NULL terminated ASCII string to terminal, slightly more efficient than append_utf8().
- void append_utf8 (const char *buf, int len=-1)

  Append NULL terminated UTF-8 string to terminal.
- void box (Fl_Boxtype val)

  Sets the box type, updates terminal margins et al.
- Fl_Boxtype **box** (void) const

  Returns the current box type.
- void clear (Fl_Color val)

  Clears the screen to a specific color `val` and homes the cursor.
- void clear (void)

  Clears the screen to the current textbgcolor(), and homes the cursor.
- void **clear_history** (void)

  Clears the scroll history buffer and adjusts scrollbar, forcing it to redraw().
- void clear_screen (bool scroll_to_hist=true)

  Clear the terminal screen only; does not affect the cursor position.
- void clear_screen_home (bool scroll_to_hist=true)

  Clear the terminal screen and home the cursor.
- void color (Fl_Color val)

  Sets the background color for the terminal's Fl_Group::box().
- Fl_Color **color** (void) const

  Return base widget Fl_Group's box() color()
- int **cursor_col** (void) const

  Return the cursor's current column position on the screen.
- void **cursor_home** (void)

  Move cursor to the home position (top/left).
- int **cursor_row** (void) const

  Return the cursor's current row position on the screen.
- void **cursorbgcolor** (Fl_Color val)

  Set the cursor's background color used for the cursor itself.
- Fl_Color **cursorbgcolor** (void) const

Get the cursor's background color used for the cursor itself.

- void **cursorfgcolor** (Fl_Color val)

  Set the cursor's foreground color used for text under the cursor.
- Fl_Color **cursorfgcolor** (void) const

  Get the cursor's foreground color used for text under the cursor.
- void display_columns (int val)

  Set terminal's display width in columns of text characters.
- int display_columns (void) const

  Return terminal's display width in columns of text characters.
- void display_rows (int val)

  Set terminal's display height in lines of text (rows).
- int display_rows (void) const

  Return terminal's display height in lines of text (rows).
- void draw (void) override

  Draws the entire Fl_Terminal.
- void error_char (const char ∗val)

  Sets the "error character" UTF-8 string shown for invalid UTF-8 or bad ANSI sequences if show_unknown() is true.
- const char ∗ error_char (void) const

  Returns the "error character" UTF-8 string, which is shown for invalid UTF-8 or bad ANSI sequences if show_unknown() is true.
- Fl_Terminal (int X, int Y, int W, int H, const char ∗L, int rows, int cols, int hist)

  Same as the default FLTK constructor, but lets the user force the rows, columns and history to specific sizes on creation.
- Fl_Terminal (int X, int Y, int W, int H, const char ∗L=0)

  The constructor for Fl_Terminal.
- int handle (int e) override

  Handle FLTK events.
- void history_lines (int val)

  Set the number of lines of screen history.
- int **history_lines** (void) const

  Return the number of lines of screen history.
- void **history_rows** (int val)

  Set terminal's scrollback history buffer size in lines of text (rows).
- int **history_rows** (void) const

  Return terminal's scrollback history buffer size in lines of text (rows).
- int history_use (void) const

  Returns how many lines are "in use" by the screen history buffer.
- void hscrollbar_style (ScrollbarStyle val)

  Set the horizontal scrollbar behavior style.
- ScrollbarStyle hscrollbar_style (void) const

  Get the horizontal scrollbar behavior style.
- void **margin_bottom** (int val)

  Set the bottom margin; see Margins.
- int **margin_bottom** (void) const

  Return the bottom margin; see Margins.
- void **margin_left** (int val)

  Set the left margin; see Margins.
- int **margin_left** (void) const

  Return the left margin; see Margins.
- void **margin_right** (int val)

  Set the right margin; see Margins.

- int **margin_right** (void) const

  Return the right margin; see Margins.
- void **margin_top** (int val)

  Set the top margin; see Margins.
- int **margin_top** (void) const

  Return the top margin; see Margins.
- void output_translate (Fl_Terminal::OutFlags val)

  Sets the combined output translation flags to `val`.
- Fl_Terminal::OutFlags **output_translate** (void) const

  Return the current combined output translation flags.
- void plot_char (char c, int drow, int dcol)

  Plot the ASCII character `c` at the terminal's display position (drow,dcol).
- void plot_char (const char *text, int len, int drow, int dcol)

  Plot the UTF-8 character `text` of length `len` at display position (drow,dcol).
- void print_char (char c)

  Prints single ASCII char `c` at current cursor position, and advances the cursor.
- void print_char (const char *text, int len=-1)

  Prints single UTF-8 char `text` of optional byte length `len` at current cursor position, and advances the cursor if the character is printable.
- void printf (const char *fmt,...)

  Appends printf formatted messages to the terminal.
- void redraw_rate (float val)

  Set the maximum rate redraw speed in floating point seconds if redraw_style() is set to RATE_LIMITED.
- float **redraw_rate** (void) const

  Get max rate redraw speed in floating point seconds.
- void redraw_style (RedrawStyle val)

  Set how Fl_Terminal manages screen redrawing.
- RedrawStyle redraw_style (void) const

  Get the redraw style.
- void reset_terminal (void)

  Resets terminal to default colors, clears screen, history and mouse selection, homes cursor, resets tabstops.
- void resize (int X, int Y, int W, int H) override

  Handle widget resizing, such as if user resizes parent window.
- int scrollbar_actual_size (void) const

  Returns the scrollbar's actual "trough size", which is the width of FL_VERTICAL scrollbars, or height of FL_↩ HORIZONTAL scrollbars.
- void scrollbar_size (int val)

  Set the pixel size of both horizontal and vertical scrollbar's "trough" to `val`.
- int scrollbar_size (void) const

  Get current pixel size of all the scrollbar's troughs for this widget, or zero if the global Fl::scrollbar_size() is being used (default).
- const char * selection_text (void) const

  Return text selection (for copy()/paste() operations)
- int selection_text_len (void) const

  Return byte length of all UTF-8 chars in selection, or 0 if no selection.
- void **selectionbgcolor** (Fl_Color val)

  Set mouse selection background color.
- Fl_Color **selectionbgcolor** (void) const

  Get mouse selection background color.
- void **selectionfgcolor** (Fl_Color val)

  Set mouse selection foreground color.

- Fl_Color **selectionfgcolor** (void) const

  Get mouse selection foreground color.
- void show_unknown (bool val)

  Set the "show unknown" flag.
- bool show_unknown (void) const

  Return the "show unknown" flag.
- const char * text (bool lines_below_cursor=false) const

  Return a string copy of all lines in the terminal (including history).
- uchar textattrib () const

  Get text attribute bits (underline, inverse, etc).
- void textattrib (uchar val)

  Set text attribute bits (underline, inverse, etc).
- void textbgcolor (Fl_Color val)

  Set text background color to fltk color `val` used by any new text added.
- Fl_Color **textbgcolor** (void) const

  Return text's current background color.
- void textbgcolor_default (Fl_Color val)

  Set the default text background color used by any new text added after a reset (`<ESC>c`, `<ESC>[0m`, or reset_terminal()).
- Fl_Color textbgcolor_default (void) const

  Return text's default background color.
- void textbgcolor_xterm (uchar val)

  Sets the background text color as one of the 8 'xterm color' values.
- void textcolor (Fl_Color val)

  Set the text color for the terminal.
- Fl_Color **textcolor** (void) const

  Return textcolor(). This is a convenience method that returns textfgcolor_default()
- void textfgcolor (Fl_Color val)

  Set text foreground drawing color to fltk color `val` used by any new text added.
- Fl_Color **textfgcolor** (void) const

  Return text's current foreground color.
- void textfgcolor_default (Fl_Color val)

  Set the default text foreground color used by `<ESC>c`, `<ESC>[0m`, and reset_terminal().
- Fl_Color textfgcolor_default (void) const

  Return text's default foreground color.
- void textfgcolor_xterm (uchar val)

  Sets the foreground text color as one of the 8 'xterm color' values.
- void textfont (Fl_Font val)

  Sets the font used for all text displayed in the terminal.
- Fl_Font **textfont** (void) const

  Return text font used to draw all text in the terminal.
- void textsize (Fl_Fontsize val)

  Sets the font size used for all text displayed in the terminal.
- Fl_Fontsize **textsize** (void) const

  Return text font size used to draw all text in the terminal.
- void vprintf (const char *fmt, va_list ap)

  Appends printf formatted messages to the terminal.
- ~Fl_Terminal (void)

  The destructor for Fl_Terminal.

Public Member Functions inherited from Fl_Group

- Fl_Widget *& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget *o)

  See void Fl_Group::add(Fl_Widget &w)
- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget *const * array () const

  Returns a pointer to the internal array of children.
- Fl_Group const * as_group () const override
- Fl_Group * as_group () override

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const

  Returns the n'th child.
- int **children** () const

  Returns how many child widgets the group has.
- void clear ()

  Deletes all child widgets from memory recursively.
- unsigned int clip_children () const

  Returns the current clipping mode.
- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)

  Removes the widget at `index` from the group and deletes it.
- void end ()

  Exactly the same as current(this->parent()).
- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*
- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.
- void **forms_end** ()

  This is for forms compatibility only.
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

  This does insert(w, find(before)).
- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.
- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.

- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.

- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

Returns whether the inactive image is managed by the widget.

- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.

- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

Returns if the widget is able to take events.

- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Public Attributes

- Fl_Scrollbar * hscrollbar

  Horizontal scrollbar.
- Fl_Scrollbar * scrollbar

  Vertical scrollbar.

Protected Member Functions

- void **clear_eod** (void)

  Clear from cursor to End Of Display (EOD), like "<ESC>[J" or "<ESC>[0J".
- void **clear_eol** (void)

  Clear from cursor to End Of Line (EOL), like "<ESC>[K".
- void **clear_line** (int row)

  Clear entire line for specified row.
- void **clear_line** (void)

  Clear entire line cursor is currently on.
- void **clear_mouse_selection** (void)

  Clear any current mouse selection.
- void **clear_sod** (void)

  Clear from cursor to Start Of Display (SOD), like "<ESC>[1J".
- void **clear_sol** (void)

  Clear from cursor to Start Of Line (SOL), like "<ESC>[1K".
- void **current_style** (const CharStyle &sty)

  Set current style for rendering text.
- CharStyle & **current_style** (void) const

  Return reference to internal current style for rendering text.
- void cursor_col (int col)

  Move cursor to the specified column `col`.
- void cursor_cr (void)

  Move cursor as if a CR (\r) was received.
- void **cursor_crlf** (int count=1)

  Move cursor as if a CR/LF pair (\r\n) was received.
- void cursor_down (int count=1, bool do_scroll=false)

  Moves cursor down `count` lines.
- void **cursor_eol** (void)

  Move cursor to the last column (at the far right) on the current line.
- void **cursor_left** (int count=1)

  Moves cursor left `count` columns, and cursor stops (does not wrap) if it hits screen edge.
- void cursor_right (int count=1, bool do_scroll=false)

  Moves cursor right `count` columns.
- void cursor_row (int row)

  Move cursor to the specified row `row`.
- void **cursor_sol** (void)

  Move cursor to the first column (at the far left) on the current line.
- void **cursor_tab_left** (int count=1)

  Tab left, do not wrap beyond left edge.
- void **cursor_tab_right** (int count=1)

  Tab right, do not wrap beyond right edge.
- void cursor_up (int count=1, bool do_scroll=false)

  Moves cursor up `count` lines.
- void **delete_chars** (int drow, int dcol, int rep)

Delete char(s) at (drow,dcol) for 'rep' times.

- void **delete_chars** (int rep)

  Delete char(s) at cursor position for 'rep' times.
- void delete_rows (int count)

  Delete (count) rows at cursor position.
- int **disp_cols** (void) const

  Return the number of columns in the display area.
- int **disp_erow** (void) const

  Return the ending row# in the display area.
- int **disp_rows** (void) const

  Return the number of rows in the display area.
- int **disp_srow** (void) const

  Return the starting row# in the display area.
- void draw_buff (int Y) const

  Draws the buffer position we are scrolled to onto the FLTK screen starting at pixel position Y.
- void draw_row (int grow, int Y) const

  Draw the specified global row, which is the row in ring_chars[].
- void draw_row_bg (int grow, int X, int Y) const

  Draw the background for the specified ring_chars[] global row `grow` starting at FLTK coords `X` and `Y`.
- bool get_selection (int &srow, int &scol, int &erow, int &ecol) const

  Return mouse selection's start/end position in the ring buffer, if any.
- int h_to_row (int H) const

  Given a height in pixels, return number of rows that "fits" into that area.
- int handle_unknown_char (int drow, int dcol)

  Handle an unknown char by either emitting an error symbol to the tty, or do nothing, depending on the user configurable value of show_unknown().
- int handle_unknown_char (void)

  Handle an unknown char by either emitting an error symbol to the tty, or do nothing, depending on the user configurable value of show_unknown().
- int **hist_cols** (void) const

  Return the number of columns in the scrollback history.
- int **hist_erow** (void) const

  Return the ending row# of the scrollback history.
- int **hist_rows** (void) const

  Return the number of rows in the scrollback history.
- int **hist_srow** (void) const

  Return the starting row# of the scrollback history.
- int **hist_use** (void) const

  Return number of rows in use by the scrollback history.
- int **hist_use_srow** (void) const

  Return the starting row of the "in use" scrollback history.
- void insert_char (char c, int rep)

  Insert char 'c' at the current cursor position for 'rep' times.
- void **insert_char_eol** (char c, int drow, int dcol, int rep)

  Insert char 'c' for 'rep' times at display row `'drow'` and column `'dcol'`.
- void insert_rows (int count)

  Insert (count) rows at current cursor position.
- bool is_inside_selection (int row, int col) const

  Is global row/column inside the current mouse selection?
- bool **is_selection** (void) const

  Returns true if there's a mouse selection.

- int **offset** (void) const

  Returns the current offset into the ring buffer.
- void **restore_cursor** (void)

  Restore previously saved cursor position, if any. Used by ESC [u.
- int **ring_cols** (void) const

  Return the number of columns in the ring buffer.
- int **ring_erow** (void) const

  Return the ending row# in the ring buffer (Always ring_rows()-1)
- int **ring_rows** (void) const

  Return the number of rows in the ring buffer.
- int **ring_srow** (void) const

  Return the starting row# in the ring buffer. (Always 0)
- void **save_cursor** (void)

  Save current cursor position. Used by ESC [s.
- void scroll (int rows)

  Scroll the display up(+) or down(-) the specified `rows`.
- void **select_line** (int grow)

  Select the entire row.
- void **select_word** (int grow, int gcol)

  Select the word around the given row and column.
- bool selection_extend (int X, int Y)

  Extend selection to FLTK coords X,Y.
- Utf8Char ∗ **u8c_cursor** (void)

  Return the Utf8Char∗ for character under cursor.
- Utf8Char ∗ u8c_disp_row (int drow)

  Return pointer to the first u8c character in row `drow` of the display.
- const Utf8Char ∗ **u8c_disp_row** (int drow) const

  See docs for non-const version of u8c_disp_row(int)
- Utf8Char ∗ u8c_hist_row (int hrow)

  Return u8c for beginning of a row inside the scrollback history.
- const Utf8Char ∗ **u8c_hist_row** (int hrow) const

  See docs for non-const version of u8c_hist_row(int)
- const Utf8Char ∗ **u8c_hist_use_row** (int hrow) const

  See docs for non-const version of u8c_hist_use_row(int)
- Utf8Char ∗ u8c_hist_use_row (int hurow)

  Return u8c for beginning of row `hurow` inside the 'in use' part of the scrollback history.
- Utf8Char ∗ u8c_ring_row (int grow)

  Return UTF-8 char for row `grow` in the ring buffer.
- const Utf8Char ∗ **u8c_ring_row** (int grow) const

  See docs for non-const version of u8c_ring_row(int)
- const Utf8Char ∗ utf8_char_at_disp (int drow, int dcol) const

  Return Utf8Char∗ for char at specified display row and column.
- const Utf8Char ∗ utf8_char_at_glob (int grow, int gcol) const

  Return Utf8Char∗ for char at specified global (grow,gcol).
- int w_to_col (int W) const

  Given a width in pixels, return number of columns that "fits" into that area.
- const Utf8Char ∗ walk_selection (const Utf8Char ∗u8c, int &row, int &col) const

  Walk the mouse selection one character at a time from beginning to end, returning a Utf8Char∗ to the next character in the selection, or NULL if the end was reached, or if there's no selection.

Protected Member Functions inherited from Fl_Group

- Fl_Rect ∗ bounds ()

  Returns the internal array of widget sizes and positions.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget ∗, int)

  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)

  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)

  Allow derived groups to act when a child widget is removed from the group.
- int ∗ sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

Internal use only.

- void x (int v)

  Internal use only.

- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()

  Returns the currently active group.

- static void current (Fl_Group *g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.

- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.

- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.134.1 Detailed Description

Terminal widget supporting Unicode/utf-8, ANSI/xterm escape codes with full RGB color control.

13.134.2 The Fl_Terminal Widget

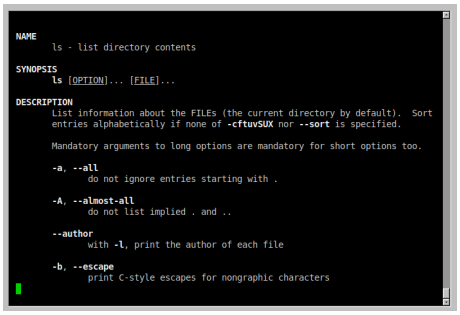


Figure 13.54 Fl_Terminal widget showing a linux manual page

Fl_Terminal is an output-only text widget supporting ASCII and UTF-8/Unicode. It supports most terminal text features, such as most VT100/xterm style escape sequences (see The Escape Codes Fl_Terminal Supports), text colors/attributes, scrollback history, mouse selection, etc.
It is recommended that accessing features such as setting text colors and cursor positioning is best done with ANSI/XTERM escape sequences. But if one sets ansi(false) then this is not possible, so the public API can be used for common operations, e.g.

Public API	ESC code equivalent	Description
clear_screen_home()	ESC [H ESC [2 J	Clear screen, home cursor
cursor_home()	ESC [H	Home the cursor
clear_history()	ESC [3 J	Clear scrollback history
reset_terminal()	ESC c	Reset terminal

To access more advanced API calls, one can derive a class from Fl_Terminal to access protected methods and manipulate the terminal more directly, e.g.

Protected API	ESC code equiv.	Description
current_style()	ESC [# m	Set text attributes
clear_eod()	ESC [0 J	Clear from cursor to end of display
clear_sod()	ESC [1 J	Clear from cursor to start of display
clear_eol()	ESC [0 K	Clear from cursor to end of line
clear_sol()	ESC [1 K	Clear from cursor to start of line
clear_line()	ESC [2 K	Clear line cursor is on
scroll(int) // >0 for up	ESC [1 S	Scroll up one line
scroll(int) // <0 for down	ESC [1 T	Scroll down one line
cursor_left()	ESC [1 D	Move cursor left (no wrap)
cursor_right()	ESC [1 C	Move cursor right (no wrap)
cursor_up()	ESC [1 A	Move cursor up (no scroll or wrap)
cursor_down()	ESC [1 B	Move cursor down (no scroll or wrap)
cursor_row() cursor_col()	ESC [# ; # H	Move cursor to row# / column#
insert_char()	ESC [# @	Insert a char at cursor position
delete_chars()	ESC [# P	Delete chars at cursor position
insert_rows()	ESC [# L	Insert rows at cursor position
delete_rows()	ESC [# M	Delete rows at cursor position
etc...	etc...	etc...

Many commonly used API functions are public, such as textfgcolor() for setting text colors. Others, such as cursor_up() are protected to prevent common misuse, and are available only to subclasses.
Some common operations:

- Set the terminal's background color, see color(Fl_Color)
- Set the terminal's default text color, see textfgcolor_default(Fl_Color)
- Printing text to the terminal, see Fl_Terminal::printf() and Fl_Terminal::append()
- Clearing the screen, see clear()
- Getting the terminal's buffer contents, see text()
- Getting single UTF-8 characters by row/col from the terminal display, see utf8_char_at_disp()
- Getting the text from a text selection, see get_selection()

For applications that need input support, the widget can be subclassed to provide keyboard input, and advanced features like pseudo ttys, termio, serial port I/O, etc., as such features are beyond the scope of FLTK.

13.134.2.1 Examples

```
//
// Fl_Terminal: Simple Use
//
Fl_Terminal *tty = new Fl_Terminal(...);
:
tty->append("Hello world.\n");                      // simple strings
tty->append("\033[31mThis text is red.\033[0m\n");  // colored text
tty->append("\033[32mThis text is green.\033[0m\n");
tty->printf("The value of x is %.02f\n", x);        // printf() formatting
```

There are also public methods for doing what most "\033[" escape codes do, so that if ansi(bool) is set to "false", one can still change text colors or clear the screen via application control, e.g.

```
tty->cursor_home();           // home the cursor
tty->clear_screen();          // clear the screen
tty->textfgcolor(0xff000000); // change the text color to RED
tty->textbgcolor(0x0000ff00); // change the background color to BLUE
//
//                 R G B
```

When creating the widget, the width/height determine the default column and row count for the terminal's display based on the current font size. The column width determines where text will wrap.

You can specify wider column sizes than the screen using display_columns(colwidth). When this value is larger than the widget's width, text will wrap off-screen, and can be revealed by resizing the widget wider.

13.134.2.2 Writing To Terminal From Applications

An application needing terminal output as part of its user interface can instance Fl_Terminal, and write text strings with:

- append() to append strings
- printf() to append formatted strings

Single character output can be done with:

- print_char() to print a single ASCII/UTF-8 char at the cursor
- plot_char() to put single ASCII/UTF-8 char at an x,y position

13.134.2.3 Text Attributes

The terminal's text supports these attributes:

- Italic - italicized text: `\033[3m`
- Bold - brighter/thicker text: `\033[1m`
- Dim - lower brightness text: `\033[2m`
- Underline - text that is underlined: `\033[4m`
- Strikeout - text that has a line through the text: `\033[9m`
- Inverse - text whose background and foreground colors are swapped: `\033[7m`
- Normal - normal text: `\033[0m`

Figure 13.55 Fl_Terminal screen

13.134.2.4 Text and Background Colors

There's at least two ways to specify colors for text and background colors:

- 3-bit / 8-color values
- Full 24-bit R/G/B colors

Example of 3-bit colors:

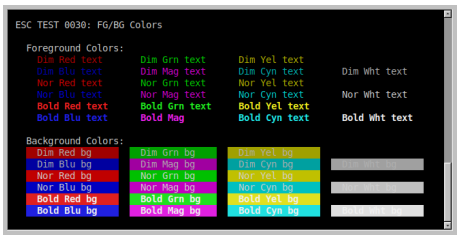


Figure 13.56 Fl_Terminal 3-bit colors

Example application source code using 3-bit colors:

```
//
// Text colors
//
tty->append("\033[31m Red text.\033[0m\n");   // Print red text..
tty->append("\033[32m Green text.\033[0m\n");
:
tty->append("\033[36m Cyan text.\033[0m\n");
tty->append("\033[37m White text.\033[0m\n");
//
// Background colors
//
tty->append("\033[41m Red Background.\033[0m\n");  // background will be red
tty->append("\033[42m Green Background.\033[0m\n");
:
tty->append("\033[46m Cyan Background.\033[0m\n");
tty->append("\033[47m White Background.\033[0m\n");
```

Example of 24-bit colors:

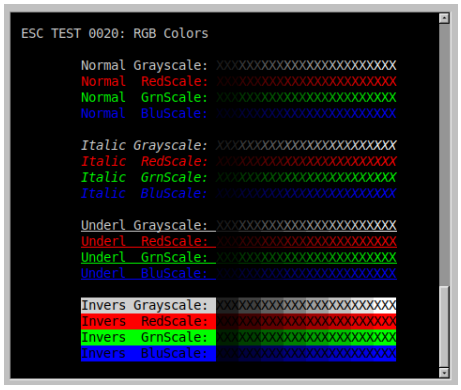


Figure 13.57 Fl_Terminal 24-bit colors

Example application source code using 24-bit colors:

```
//
// 24-bit Text Color
//
tty->append("\033[38;2;0;0;255m Text is BLUE.\033[0m\n");                // RGB: R=0,   G=0,  B=255
tty->append("\033[38;2;255;0;0m Text is RED.\033[0m\n");                 // RGB: R=255, G=0,  B=0
tty->append("\033[38;2;127;64;0m Text is DARK ORANGE.\033[0m\n");        // RGB: R=127, G=64, B=0
//
// 24-bit Background Color
//
tty->append("\033[48;2;0;0;255m Background is BLUE.\033[0m\n");          // RGB: R=0,   G=0,  B=255
tty->append("\033[48;2;255;0;0m Background is RED.\033[0m\n");           // RGB: R=255, G=0,  B=0
tty->append("\033[48;2;127;64;0m Background is DARK ORANGE.\033[0m\n");  // RGB: R=127, G=64, B=0
```

For more on the ANSI escape codes, see The Escape Codes Fl_Terminal Supports.

13.134.2.5 Features

Most standard terminal behaviors are supported, e.g.

- ASCII + UTF-8/Unicode
- scrollback history management
- mouse selection + copy/paste (^C, ^A)
- autoscroll during selection

Most popular ANSI/DEC VT100/Xterm escape sequences are supported (see The Escape Codes Fl_Terminal Supports), including:

- per-character colors for text and background
- per-character text attributes: bold/dim, underline, strikeout
- scrolling up/down
- character insert/delete for characters/rows/screen
- clearing characters/rows/screen

Does not (yet) support:

- programmable regions (scroll regions and attribute blocks)
- dynamic line wrap (where resizing display dynamically re-wraps long lines)

Will likely never implement as part of this widget:

- pty/termio management (such features should be *subclassed*)
- Different per-character font family + sizes (font family/size is global only)
- variable width fonts

Regarding the font family+size; the way the terminal is currently designed, the font family and size must not vary within text; rows have to be consistent height. Varying widths are tricky too, esp. when it comes to moving the cursor up/down within a column; varying *widths* are supported (due to Unicode characters sometimes being "wide", but not heights.

13.134.2.6 Margins

The margins define the amount of space (in pixels) around the outside of the text display area, the space between the widget's inner edge (inside the box()) and the text display area's outer edge. The margins can be inspected and changed with the margin_left(), margin_right(), margin_top() and margin_bottom() methods.

```
.
     TERMINAL WIDGET (Fl_Terminal)

                             ↑ margin_top
         TEXT DISPLAY AREA   ↓

                 S
                 c
          display_rows       r
                 o
                 l
                 l
          display_columns
                 B
                 a
                 r



      margin_left            ↑                              margin_right
                             ↓ margin_bottom

                              Fl_Terminal Margins



                            :::::::::::::
                            ::##:::::##:
```

```
::###:::###::
::####:####::  Fl_Terminal::color()
::##:###:##::
::##::#::##::
::##:.:::##::
::##::::::#::
::::::::::::
 Fl_Terminal::textbgcolor()

                                    Fl_Terminal::textfgcolor()
```

.

13.134.2.7 Caveats

- This widget is not a full terminal emulator; it does not do stdio redirection, pseudo ttys/termios/character cooking, keyboard input processing, full curses(3) support. However, such features CAN be implemented with subclassing.

- The printf() and vprintf() functions are limited to strings no longer than 1024 characters (including NULL). For printing longer strings, use append() which has no string limits.

- For background colors textbgcolor() and textbgcolor_default(), the special color value 0xffffffff lets the widget's box() color() show through behind the text.

13.134.3 Member Enumeration Documentation

13.134.3.1 Attrib

`enum Fl_Terminal::Attrib`

Bits for the per-character attributes, which control text features such as italic, bold, underlined text, etc.
NOTE: enum names with a leading underbar must not be used, and are reserved for future use.

Enumerator

NORMAL	all attributes off
BOLD	bold text: uses bold font, color brighter than normal
DIM	dim text; color slightly darker than normal
ITALIC	italic font text
UNDERLINE	underlined text
_RESERVED↩_1	*(reserved for internal future use)*
INVERSE	inverse text; fg/bg color are swapped
_RESERVED↩_2	*(reserved for internal future use)*
STRIKEOUT	strikeout text

13.134.3.2 CharFlags

`enum Fl_Terminal::CharFlags`

Per-character 8 bit flags (uchar) used to manage special states for characters.

Enumerator

FG_XTERM	this char's fg color is an XTERM color; can be affected by Dim+Bold
BG_XTERM	this char's bg color is an XTERM color; can be affected by Dim+Bold
EOL	TODO: char at EOL, used for line re-wrap during screen resizing.

13.134.3.3 OutFlags

`enum Fl_Terminal::OutFlags`

Output translation flags for special control character translations.

Enumerator

OFF	no output translation
CR_TO_LF	carriage return generates a vertical line-feed (\r -> \n)
LF_TO_CR	line-feed generates a carriage return (\n -> \r)
LF_TO_CRLF	line-feed generates a carriage return line-feed (\n -> \r\n)

13.134.3.4 RedrawStyle

`enum Fl_Terminal::RedrawStyle`

Determines when Fl_Terminal calls redraw() if new text is added.
RATE_LIMITED is the recommended setting, using redraw_rate(float) to determine the maximum rate of redraws.

See also

redraw_style(), redraw_rate()

Enumerator

NO_REDRAW	app must call redraw() as needed to update text to screen
RATE_LIMITED	timer controlled redraws. (DEFAULT)
PER_WRITE	redraw triggered after *every* append() / printf() / etc. operation

13.134.3.5 ScrollbarStyle

`enum Fl_Terminal::ScrollbarStyle`

Behavior of scrollbars.

Enumerator

SCROLLBAR_OFF	scrollbar always invisible
SCROLLBAR_AUTO	scrollbar visible if widget resized in a way that hides columns (default)
SCROLLBAR_ON	scrollbar always visible

13.134.4 Constructor & Destructor Documentation

13.134.4.1 Fl_Terminal() [1/2]

```
Fl_Terminal::Fl_Terminal (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

The constructor for Fl_Terminal.
This creates an empty terminal with defaults:

- white on black text; see textfgcolor(Fl_Color), textbgcolor(Fl_Color)
- rows/cols based on the `W` and `H` values, see display_rows(), display_columns()

- scrollback history of 100 lines, see history_rows()
- redraw_style() set to RATE_LIMITED, redraw_rate() set to 0.10 seconds

Note: While Fl_Terminal derives from Fl_Group, it's not intended for user code to use it as a parent for other widgets, so end() is called.

Parameters

in	*X,Y,W,H*	position and size.
in	*L*	label string (optional), may be NULL.

13.134.4.2 Fl_Terminal() [2/2]

```
Fl_Terminal::Fl_Terminal (
            int X,
            int Y,
            int W,
            int H,
            const char * L,
            int rows,
            int cols,
            int hist )
```

Same as the default FLTK constructor, but lets the user force the rows, columns and history to specific sizes on creation.
Since the row/cols/hist are specified directly, this prevents the widget from auto-calculating the initial text buffer size based on the widget's pixel width/height, bypassing calls to the font system before the widget is displayed.

Note

fluid uses this constructor internally to avoid font calculations that open the display, useful for when running in a headless context. (issue 837)

13.134.4.3 ~Fl_Terminal()

```
Fl_Terminal::~Fl_Terminal (
            void )
```

The destructor for Fl_Terminal.
Destroys the terminal display, scroll history, and associated widgets.

13.134.5 Member Function Documentation

13.134.5.1 ansi() [1/2]

```
void Fl_Terminal::ansi (
            bool val )
```

Enable/disable the ANSI mode flag.
If true, ANSI and VT100/xterm codes will be processed. If false, these codes won't be processed and will either be ignored or print the error character "¿", depending on the value of show_unknown().

See also

show_unknown(), The Escape Codes Fl_Terminal Supports

13.134.5.2 ansi() [2/2]

```
bool Fl_Terminal::ansi (
            void ) const
```

Return the state of the ANSI flag.

See also

ansi(bool)

13.134.5.3 append()

```
void Fl_Terminal::append (
            const char * s,
            int len = -1 )
```

Appends string `s` to the terminal at the current cursor position using the current text color/attributes.
If `s` is NULL, the UTF-8 character cache is cleared, which is recommended before starting a block reading loop, and again after the block loop has completed.
If `len` is not specified, it's assumed `s` is a NULL terminated string. If `len` IS specified, it can be used for writing strings that aren't NULL terminated, such as block reads on a pipe, network, or other block oriented data source.
Redraws of the terminal widget are by default handled automatically, but can be changed with redraw_rate() and redraw_style().
Block I/O
When reading block oriented sources (such as pipes), append() will handle partial UTF-8 chars straddling the block boundaries. It does this using an internal byte cache, which should be cleared before and after block I/O loops by calling `append(NULL)` as shown in the example below, to prevent the possibilities of partial UTF-8 characters left behind by an interrupted or incomplete block loop.

```
// Example block reading a command pipe in Unix

  // Run command and read as a pipe
  FILE *fp = popen("ls -la", "r");
  if (!fp) { ..error_handling.. }

  // Enable non-blocking I/O
  int fd = fileno(fp);
  fcntl(fd, F_SETFL, O_NONBLOCK);

  // Clear UTF-8 character cache before starting block loop
  G_tty->append(NULL);                              // prevents leftover partial UTF-8 bytes

  // Block read loop
  while (1) {
    Fl::wait(0.05);                                 // give fltk .05 secs of cpu to manage UI
    ssize_t bytes = read(fd, s, sizeof(s));         // read block from pipe
    if (bytes == -1 && errno == EAGAIN) continue;   // no data yet? continue
    if (bytes > 0) G_tty->append(s, bytes);         // append output to terminal
    else break;                                     // end of pipe?
  }

  // Flush cache again after block loop completes
  G_tty->append(NULL);

  // Close pipe, done
  pclose(fp);
```

Note

- String can contain ASCII or UTF-8 chars
- `len` is optional; if unspecified, expects `s` to be a NULL terminated string
- Handles partial UTF-8 chars split between calls (e.g. block oriented writes)
- If `s` is NULL, this clears the "partial UTF-8" character cache
- Redraws are managed automatically by default; see redraw_style()

13.134.5.4 append_ascii()

```
void Fl_Terminal::append_ascii (
            const char * s )
```

Append NULL terminated ASCII string to terminal, slightly more efficient than append_utf8().

- If `s` is NULL, behavior is to do nothing
- Redraws are triggered automatically, depending on redraw_style()

13.134.5.5 append_utf8()

```
void Fl_Terminal::append_utf8 (
            const char * buf,
            int len = -1 )
```

Append NULL terminated UTF-8 string to terminal.

- If buf is NULL, UTF-8 cache buffer is cleared
- If optional `len` isn't specified or is -1, strlen(buf) is used.
- If `len` is 0 or <-1, no changes are made
- Handles UTF-8 chars split across calls (e.g. block writes from pipes, etc)
- Redraws are triggered automatically, depending on redraw_style()

13.134.5.6 box()

```
void Fl_Terminal::box (
            Fl_Boxtype val )  [inline]
```

Sets the box type, updates terminal margins et al.
Default is FL_DOWN_FRAME.
FL_XXX_FRAME types are handled in a special way by this widget, and guarantee the background is a flat field.
FL_XXX_BOX may draw gradients as inherited by Fl::scheme().

13.134.5.7 clear() [1/2]

```
void Fl_Terminal::clear (
            Fl_Color val )
```

Clears the screen to a specific color `val` and homes the cursor.

See also

clear_screen(), clear_screen_home(), cursor_home()

13.134.5.8 clear() [2/2]

```
void Fl_Terminal::clear (
            void  )
```

Clears the screen to the current textbgcolor(), and homes the cursor.

See also

clear_screen(), clear_screen_home(), cursor_home()

13.134.5.9 clear_screen()

```
void Fl_Terminal::clear_screen (
            bool scroll_to_hist = true )
```

Clear the terminal screen only; does not affect the cursor position.
Also clears the current mouse selection.
If `'scroll_to_hist'` is true, the screen is cleared by scrolling the contents into the scrollback history, where it can be retrieved with the scrollbar. This is the default behavior. If false, the screen is cleared and the scrollback history is unchanged.
Similar to the escape sequence `"<ESC>[2J"`.

See also

clear_screen_home()

13.134.5.10 clear_screen_home()

```
void Fl_Terminal::clear_screen_home (
            bool scroll_to_hist = true )
```

Clear the terminal screen and home the cursor.
Also clears the current mouse selection.
If `'scroll_to_hist'` is true, the screen is cleared by scrolling the contents into the scrollback history, where it can be retrieved with the scrollbar. This is the default behavior. If false, the screen is cleared and the scrollback history is unchanged.
Similar to the escape sequence `"<ESC>[2J<ESC>[H"`.

See also

clear_screen()

13.134.5.11 color()

```
void Fl_Terminal::color (
            Fl_Color val )
```

Sets the background color for the terminal's Fl_Group::box().
If the textbgcolor() and textbgcolor_default() are set to the special "see through" color 0xffffffff when any text was added, changing color() affects the color that shows through behind that existing text.
Otherwise, whatever specific background color was set for existing text will persist after changing color().
To see the effects of a change to color(), follow up with a call to redraw().
The default value is 0x0.

13.134.5.12 cursor_col()

```
void Fl_Terminal::cursor_col (
            int col )  [protected]
```

Move cursor to the specified column `col`.
This value is clamped to the range (0..display_columns()-1).

13.134.5.13 cursor_cr()

```
void Fl_Terminal::cursor_cr (
            void  )  [protected]
```

Move cursor as if a CR (\r) was received.
Same as cursor_sol()

13.134.5.14 cursor_down()

```
void Fl_Terminal::cursor_down (
            int count = 1,
            bool do_scroll = false )  [protected]
```

Moves cursor down `count` lines.
If cursor hits screen bottom, it either stops (does not wrap) if `do_scroll` is false, or wraps and scrolls up if `do_scroll` is true.

Parameters

count	Number of lines to move cursor down
do_scroll	Enable scrolling if set to true

13.134.5.15 cursor_right()

```
void Fl_Terminal::cursor_right (
            int count = 1,
```

```
            bool do_scroll = false )  [protected]
```

Moves cursor right `count` columns.
If cursor hits right edge of screen, it either stops (does not wrap) if `do_scroll` is false, or wraps and scrolls up one line if `do_scroll` is true.

13.134.5.16 cursor_row()

```
void Fl_Terminal::cursor_row (
            int row )  [protected]
```

Move cursor to the specified row `row`.
This value is clamped to the range (0..display_rows()-1).

13.134.5.17 cursor_up()

```
void Fl_Terminal::cursor_up (
            int count = 1,
            bool do_scroll = false )  [protected]
```

Moves cursor up `count` lines.
If cursor hits screen top, it either stops (does not wrap) if `do_scroll` is false, or scrolls down if `do_scroll` is true.

13.134.5.18 delete_rows()

```
void Fl_Terminal::delete_rows (
            int count )  [protected]
```

Delete (count) rows at cursor position.
Causes rows to scroll up, and empty lines created at bottom of screen. Lines deleted by scroll up are NOT moved into the scroll history.

13.134.5.19 display_columns() [1/2]

```
void Fl_Terminal::display_columns (
            int dcols )
```

Set terminal's display width in columns of text characters.
This value is normally managed automatically by resize() based on the current font size, and should not be changed.
You CAN make the display_columns() larger than the width of the widget; text in the terminal will simply run off the screen edge and be clipped; the only way to reveal that text is if the user enlarges the widget, or the font size made smaller.
To change the display width, it is best to use resize() instead.

13.134.5.20 display_columns() [2/2]

```
int Fl_Terminal::display_columns (
            void  ) const
```

Return terminal's display width in columns of text characters.
This value is normally managed automatically by resize() based on the current font size.

13.134.5.21 display_rows() [1/2]

```
void Fl_Terminal::display_rows (
            int drows )
```

Set terminal's display height in lines of text (rows).
This value is normally managed automatically by resize() based on the current font size, and should not be changed.
To change the display height, use resize() instead.

13.134.5.22 display_rows() [2/2]

```
int Fl_Terminal::display_rows (
            void  ) const
```

Return terminal's display height in lines of text (rows).
This value is normally managed automatically by resize() based on the current font size.

13.134.5.23 draw()

```
void Fl_Terminal::draw (
            void ) [override], [virtual]
```

Draws the entire Fl_Terminal.
Lets the group draw itself first (scrollbars should be only members), followed by the terminal's screen contents.
Reimplemented from Fl_Group.

13.134.5.24 draw_buff()

```
void Fl_Terminal::draw_buff (
            int Y ) const [protected]
```

Draws the buffer position we are scrolled to onto the FLTK screen starting at pixel position Y.
This can be anywhere in the ring buffer, not just the 'active display'; depends on what position the scrollbar is set to.
Handles attributes, colors, text selections, cursor.

Parameters

in	*Y*	top position of top left character in the window in FLTK coordinates

13.134.5.25 draw_row()

```
void Fl_Terminal::draw_row (
            int grow,
            int Y ) const [protected]
```

Draw the specified global row, which is the row in ring_chars[].
The global row includes history + display buffers.

Parameters

in	*grow*	row number
in	*Y*	top position of characters in the row in FLTK coordinates

13.134.5.26 draw_row_bg()

```
void Fl_Terminal::draw_row_bg (
            int grow,
            int X,
            int Y ) const [protected]
```

Draw the background for the specified ring_chars[] global row `grow` starting at FLTK coords `X` and `Y`.
Note we may be called to draw display, or even history if we're scrolled back. If there's any change in bg color, we draw the filled rects here.
If the bg color for a character is the special "see through" color 0xffffffff, no pixels are drawn.

Parameters

in	*grow*	row number
in	*X,Y*	top left corner of the row in FLTK coordinates

13.134.5.27 error_char() [1/2]

```
void Fl_Terminal::error_char (
```

```
const char * val )  [inline]
```

Sets the "error character" UTF-8 string shown for invalid UTF-8 or bad ANSI sequences if show_unknown() is true. Default: "¿". This method internally makes a copy of the specified string. `val` must be a valid null terminated UTF-8 string containing exactly one character of at most four bytes, or undefined behavior will result.

See also

show_unknown(bool)

13.134.5.28 error_char() [2/2]

```
const char * Fl_Terminal::error_char (
            void  ) const  [inline]
```

Returns the "error character" UTF-8 string, which is shown for invalid UTF-8 or bad ANSI sequences if show_unknown() is true.

See also

show_unknown(bool)

13.134.5.29 get_selection()

```
bool Fl_Terminal::get_selection (
            int & srow,
            int & scol,
            int & erow,
            int & ecol ) const  [protected]
```

Return mouse selection's start/end position in the ring buffer, if any.
Ensures (start <= end) to allow walking 'forward' thru selection, left-to-right, top-to-bottom. The row/col values are indexes into the entire ring buffer.
Example: walk the characters of the mouse selection:

```
// Get selection
int srow,scol,erow,ecol;
if (get_selection(srow,scol,erow,ecol)) {                // mouse selection exists?
  // Walk entire selection from start to end
  for (int row=srow; row<=erow; row++) {                 // walk rows of selection
    const Utf8Char *u8c = u8c_ring_row(row);             // ptr to first character in row
    int col_start = (row==srow) ? scol : 0;              // start row? start at scol
    int col_end   = (row==erow) ? ecol : ring_cols()-1;  // end row?   end at ecol
    u8c += col_start;                                    // include col offset (if any)
    for (int col=col_start; col<=col_end; col++,u8c++) { // walk columns
      ..do something with each char at *u8c..
    }
  }
}
```

Returns:

- true – valid selection values returned
- false – no selection was made, returned values undefined

See also

walk_selection(), is_selection()

Parameters

srow	starting row for selection
scol	starting column for selection
erow	ending row for selection
ecol	ending column for selection

13.134.5.30 h_to_row()

```
int Fl_Terminal::h_to_row (
            int H ) const  [protected]
```

Given a height in pixels, return number of rows that "fits" into that area.
This is used by the constructor to size the row/cols to fit the widget size.

13.134.5.31 handle()

```
int Fl_Terminal::handle (
            int e )  [override], [virtual]
```

Handle FLTK events.
Reimplemented from Fl_Group.

13.134.5.32 handle_unknown_char() [1/2]

```
int Fl_Terminal::handle_unknown_char (
            int drow,
            int dcol )  [protected]
```

Handle an unknown char by either emitting an error symbol to the tty, or do nothing, depending on the user configurable value of show_unknown().
This writes the "unknown" character to the display position (drow,dcol) if show_unknown() is true.
Returns 1 if tty modified, 0 if not.

See also

show_unknown()

13.134.5.33 handle_unknown_char() [2/2]

```
int Fl_Terminal::handle_unknown_char (
            void  )  [protected]
```

Handle an unknown char by either emitting an error symbol to the tty, or do nothing, depending on the user configurable value of show_unknown().
This writes the "unknown" character to the output stream if show_unknown() is true.
Note: This method uses print_char() to emit the error_char(); if ansi() is enabled and an ansi sequence is in progress, the error_char() may be consumed as part of the ANSI sequence. It is therefore best to call escseq.reset() first.
Returns 1 if tty modified, 0 if not.

See also

show_unknown()

13.134.5.34 history_lines()

```
void Fl_Terminal::history_lines (
            int val )
```

Set the number of lines of screen history.
Large values can be briefly heavy on cpu and memory usage.

13.134.5.35 history_use()

```
int Fl_Terminal::history_use (
            void  ) const
```

Returns how many lines are "in use" by the screen history buffer.
This value will be 0 if history was recently cleared with e.g. clear_history() or "<ESC>c".
Return value will be in the range 0 .. history_lines().

13.134.5.36 hscrollbar_style() [1/2]

```
void Fl_Terminal::hscrollbar_style (
            ScrollbarStyle val )
```

Set the horizontal scrollbar behavior style.
This determines when the scrollbar is visible.

ScrollbarStyle enum	Description
SCROLLBAR_ON	Horizontal scrollbar always displayed.
SCROLLBAR_OFF	Horizontal scrollbar never displayed.
SCROLLBAR_AUTO	Horizontal scrollbar displayed whenever widget width hides columns.

The default style is SCROLLBAR_AUTO.

See also

ScrollbarStyle

13.134.5.37 hscrollbar_style() [2/2]

```
Fl_Terminal::ScrollbarStyle Fl_Terminal::hscrollbar_style (
            void ) const
```

Get the horizontal scrollbar behavior style.
This determines when the scrollbar is visible.
Value will be one of the Fl_Terminal::ScrollbarStyle enum values.

See also

hscrollbar_style(Fl_Terminal::ScrollbarStyle)

13.134.5.38 insert_char()

```
void Fl_Terminal::insert_char (
            char c,
            int rep ) [protected]
```

Insert char 'c' at the current cursor position for 'rep' times.
Does not wrap; characters at end of line are lost.

13.134.5.39 insert_rows()

```
void Fl_Terminal::insert_rows (
            int count ) [protected]
```

Insert (count) rows at current cursor position.
Causes rows below to scroll down, and empty lines created. Lines deleted by scroll down are NOT moved into the scroll history.

13.134.5.40 is_inside_selection()

```
bool Fl_Terminal::is_inside_selection (
            int grow,
            int gcol ) const [protected]
```

Is global row/column inside the current mouse selection?

Returns

- true – (`grow`, `gcol`) is inside a valid selection.
- false – (`grow`, `gcol`) is outside, or no valid selection.

13.134.5.41 output_translate()

```
void Fl_Terminal::output_translate (
            Fl_Terminal::OutFlags val )
```

Sets the combined output translation flags to `val`.
`val` can be sensible combinations of the OutFlags bit flags.
The default is LF_TO_CRLF, so that \n will generate both carriage-return (CR) and line-feed (LF).
For \r and \n to be handled literally, use output_translate(Fl_Terminal::OutFlags::OFF);
To disable all output translations, use 0 or Fl_Terminal::OutFlags::OFF.

13.134.5.42 plot_char() [1/2]

```
void Fl_Terminal::plot_char (
            char c,
            int drow,
            int dcol )
```

Plot the ASCII character `c` at the terminal's display position (drow,dcol).
The character MUST be printable (in range 0x20 - 0x7e), and is displayed using the current text color/attributes.
Characters outside that range are either ignored or print the error character (¿), depending on show_unknown(bool).
This is a very low level method.
No range checking is done on drow,dcol:

- `drow` must be in range 0..(display_rows()-1)
- `dcol` must be in range 0..(display_columns()-1)
- Does not trigger redraws
- Does NOT handle control codes, ANSI or XTERM escape sequences.

See also

show_unknown(bool), handle_unknown_char(), is_printable()

13.134.5.43 plot_char() [2/2]

```
void Fl_Terminal::plot_char (
            const char * text,
            int len,
            int drow,
            int dcol )
```

Plot the UTF-8 character `text` of length `len` at display position (drow,dcol).
The character is displayed using the current text color/attributes.
This is a very low level method.
No range checking is done on drow,dcol:

- `drow` must be in range 0..(display_rows()-1)
- `dcol` must be in range 0..(display_columns()-1)
- Does not trigger redraws
- Does not handle control codes, ANSI or XTERM escape sequences.
- Invalid UTF-8 chars show the error character (¿) depending on show_unknown(bool).

See also

handle_unknown_char()

13.134.5.44 print_char() [1/2]

```
void Fl_Terminal::print_char (
            char c )
```

Prints single ASCII char `c` at current cursor position, and advances the cursor.
The character is displayed at the current cursor position using the current text color/attributes.

- `c` must be ASCII, not UTF-8
- Does not trigger redraws

13.134.5.45 print_char() [2/2]

```
void Fl_Terminal::print_char (
            const char * text,
            int len = -1 )
```

Prints single UTF-8 char `text` of optional byte length `len` at current cursor position, and advances the cursor if the character is printable.
Handles ASCII and control codes (CR, LF, etc).
The character is displayed at the current cursor position using the current text color/attributes.
Handles control codes and can be used to construct ANSI/XTERM escape sequences.

- If optional `len` isn't specified or <0, fl_utf8len1(*text) is used.
- `text` must not be NULL.
- `len` must not be 0.
- `text` must be a single char only (whether UTF-8 or ASCII)
- `text` can be an ASCII character, though not as efficient as print_char()
- Invalid UTF-8 chars show the error character (¿) depending on show_unknown(bool).
- Does not trigger redraws

See also

show_unknown(bool), handle_unknown_char()

13.134.5.46 printf()

```
void Fl_Terminal::printf (
            const char * fmt,
             ...  )
```

Appends printf formatted messages to the terminal.
The string can contain UTF-8, crlf's, and ANSI sequences are also supported. Example:

```
#include <FL/Fl_Terminal.H>
int main(..) {
    :
    // Create a terminal, and append some messages to it
    Fl_Terminal *tty = new Fl_Terminal(..);
    :
    // Append three lines of formatted text to the buffer
    tty->printf("The current date is: %s.\nThe time is: %s\n", date_str, time_str);
    tty->printf("The current PID is %ld.\n", (long)getpid());
    :
```

Note

The expanded string is currently limited to 1024 characters (including NULL). For longer strings use append() which has no string limits.

13.134.5.47 redraw_rate()

```
void Fl_Terminal::redraw_rate (
            float val )
```

Set the maximum rate redraw speed in floating point seconds if redraw_style() is set to RATE_LIMITED.
When output is sent to the terminal, rather than calling redraw() right away, a timer is started with this value indicating how long to wait before calling redraw(), causing the output to be shown. 0.10 is recommended (1/10th of a second), to limit redraws to no more than 10 redraws per second.
The value that works best depends on how fast data arrives, and how fast the font system can draw text at runtime. Values too small cause too many redraws to occur, causing the terminal to get backlogged if large bursts of data arrive quickly. Values too large cause realtime output to be too "choppy".

13.134.5.48 redraw_style() [1/2]

```
void Fl_Terminal::redraw_style (
            RedrawStyle val )
```

Set how Fl_Terminal manages screen redrawing.
This setting is relevant when Fl_Terminal is used for high bandwidth data; too many redraws will slow things down, too few cause redraws to be 'choppy' when realtime data comes in.
Redrawing can be cpu intensive, depending on how many rows/cols are being displayed; worst case: large display + small font. Speed largely depends on the end user's graphics hardware and font drawing system.

RedrawStyle enum	Description
NO_REDRAW	App must call redraw() as needed to update text to screen
RATE_LIMITED	Rate limited, timer controlled redraws. (DEFAULT) See redraw_rate()
PER_WRITE	Redraw triggered *every* call to append() / printf() / etc.

The default style is RATE_LIMITED, which is the easiest to use, and automates redrawing to be capped at 10 redraws per second max. See redraw_rate(float) to control this automated redraw speed.

See also

redraw_rate(), RedrawStyle

13.134.5.49 redraw_style() [2/2]

```
Fl_Terminal::RedrawStyle Fl_Terminal::redraw_style (
            void ) const
```

Get the redraw style.
This determines when the terminal redraws itself while text is being added to it.
Value will be one of the Fl_Terminal::RedrawStyle enum values.

See also

redraw_style(Fl_Terminal::RedrawStyle)

13.134.5.50 reset_terminal()

```
void Fl_Terminal::reset_terminal (
            void )
```

Resets terminal to default colors, clears screen, history and mouse selection, homes cursor, resets tabstops.
Same as "<ESC>c"

13.134.5.51 resize()

```
void Fl_Terminal::resize (
            int X,
            int Y,
            int W,
            int H )  [override], [virtual]
```

Handle widget resizing, such as if user resizes parent window.
This may increase the column width of the widget if the width of the widget is made larger than it was.

Note

Resizing currently does not rewrap existing text. Currently enlarging makes room for longer lines, and shrinking the size lets long lines run off the right edge of the display, hidden from view. This behavior may change in the future to rewrap.

Reimplemented from Fl_Group.

13.134.5.52 scroll()

```
void Fl_Terminal::scroll (
            int rows )  [protected]
```

Scroll the display up(+) or down(-) the specified `rows`.

- Negative value scrolls "down", clearing top line, and history unaffected.
- Postive value scrolls "up", clearing bottom line, rotating top line into history.

13.134.5.53 scrollbar_actual_size()

```
int Fl_Terminal::scrollbar_actual_size (
            void  ) const
```

Returns the scrollbar's actual "trough size", which is the width of FL_VERTICAL scrollbars, or height of FL_↩ HORIZONTAL scrollbars.
If scrollbar_size() is zero (default), then the value of the global Fl::scrollbar_size() is returned, which is the default global scrollbar size for the entire application.

13.134.5.54 scrollbar_size() [1/2]

```
void Fl_Terminal::scrollbar_size (
            int val )
```

Set the pixel size of both horizontal and vertical scrollbar's "trough" to `val`.
Setting `val` to the special value 0 causes the widget to track the global Fl::scrollbar_size().
Use non-zero values *only* if you need to override the global Fl::scrollbar_size() size.

See also

Fl::scrollbar_size(), scrollbar_actual_size()

13.134.5.55 scrollbar_size() [2/2]

```
int Fl_Terminal::scrollbar_size (
            void  ) const
```

Get current pixel size of all the scrollbar's troughs for this widget, or zero if the global Fl::scrollbar_size() is being used (default).
If this value returns *zero*, this widget's scrollbars are using the global Fl::scrollbar_size(), in which case use scrollbar_actual_size() to get the actual (effective) pixel scrollbar size being used.

Returns

Scrollbar trough size in pixels, or 0 if the global Fl::scrollbar_size() is being used.

See also

Fl::scrollbar_size(int), scrollbar_actual_size()

13.134.5.56 selection_extend()

```
bool Fl_Terminal::selection_extend (
            int X,
            int Y )  [protected]
```

Extend selection to FLTK coords X,Y.
Returns true if extended, false if nothing done (X,Y offscreen)

13.134.5.57 selection_text()

```
const char * Fl_Terminal::selection_text (
            void  ) const
```

Return text selection (for copy()/paste() operations)

- Returns allocated NULL terminated string for entire selection.
- Caller must free() this memory when done.
- Unicode safe.

13.134.5.58 selection_text_len()

```
int Fl_Terminal::selection_text_len (
            void  ) const
```

Return byte length of all UTF-8 chars in selection, or 0 if no selection.
NOTE: Length includes trailing white on each line.

13.134.5.59 show_unknown() [1/2]

```
void Fl_Terminal::show_unknown (
            bool val )
```

Set the "show unknown" flag.
If true, invalid UTF-8 and invalid ANSI sequences will be shown with the error character "¿".
If false, error characters won't be shown.

See also

handle_unknown_char(), error_char(const char∗).

13.134.5.60 show_unknown() [2/2]

```
bool Fl_Terminal::show_unknown (
            void  ) const
```

Return the "show unknown" flag.

See also

show_unknown(bool), error_char(const char∗).

13.134.5.61 text()

```
const char * Fl_Terminal::text (
            bool lines_below_cursor = false ) const
```

Return a string copy of all lines in the terminal (including history).
The returned string is allocated with `strdup(3)`, which the caller must `free(3)`.

If `'lines_below_cursor'` is false (default), lines below the cursor on down to the bottom of the display are ignored, and not included in the returned string.
If `'lines_below_cursor'` is true, then all lines in the display are returned including any below the cursor, even if all are blank.
Example use:

```
Fl_Terminal *tty = new Fl_Terminal(..);
:
const char *s = tty->text();   // get a copy of the terminal's contents
printf("Terminal's contents is:\n%s\n", s);
free((void*)s);                // free() the copy when done!
```

Parameters

in	*lines_below_cursor*	include lines below cursor, default: false

Returns

A string allocated with strdup(3) which must be free()ed, text is UTF-8.

13.134.5.62 textattrib() [1/2]

`uchar Fl_Terminal::textattrib ( ) const`
Get text attribute bits (underline, inverse, etc).
This is the default attribute used for all newly printed text.

See also

textattrib(uchar), Fl_Terminal::Attrib

13.134.5.63 textattrib() [2/2]

```
void Fl_Terminal::textattrib (
            uchar val )
```
Set text attribute bits (underline, inverse, etc).
This will be the default attribute used for all newly printed text.

See also

Fl_Terminal::Attrib

13.134.5.64 textbgcolor()

```
void Fl_Terminal::textbgcolor (
            Fl_Color val )
```
Set text background color to fltk color `val` used by any new text added.
Use this for temporary color changes, similar to <ESC>[48;2;<R>;<G>;<B>m
Colors set this way will NOT be influenced by the xterm Dim/Bold color intensity attributes. For that, use textbgcolor_xterm(uchar) instead.
This setting does *not* affect the 'default' text colors used by <ESC>[0m, <ESC>c, reset_terminal(), etc. To set that too, also set textbgcolor_default(Fl_Color), e.g.

```
// Set both 'current' and 'default' colors
Fl_Color darkamber = 0x20100000;
tty->textbgcolor(darkamber);          // set 'current' bg color
tty->textbgcolor_default(darkamber); // set 'default' bg color used by ESC[0m reset
```

The special color value 0xffffffff (all ff's) is the "see through" color, which lets the widget's own Fl_Group::color() show through behind the text. This special text background color is the *default*, and is what most situations need.

See also

textbgcolor_default(Fl_Color), textbgcolor_xterm(uchar)

13.134.5.65 textbgcolor_default() [1/2]

```
void Fl_Terminal::textbgcolor_default (
            Fl_Color val )
```

Set the default text background color used by any new text added after a reset (<ESC>c, <ESC>[0m, or reset_terminal()).
Does not affect the 'current' text background color; use textbgcolor(Fl_Color) to set that.
The special color value 0xffffffff (all ff's) is the "see through" color, which lets the widget's own Fl_Group::color() show through behind the text. This special text background color is the *default*, and is what most situations need.

See also

textbgcolor(Fl_Color)

13.134.5.66 textbgcolor_default() [2/2]

```
Fl_Color Fl_Terminal::textbgcolor_default (
            void ) const [inline]
```

Return text's default background color.

See also

textbgcolor()

13.134.5.67 textbgcolor_xterm()

```
void Fl_Terminal::textbgcolor_xterm (
            uchar val )
```

Sets the background text color as one of the 8 'xterm color' values.
This will be the background color used for all newly printed text, similar to the <ESC>[#m escape sequence, where # is between 40 and 47.
This color will be reset to the default bg color if reset_terminal() is called, or by <ESC>c, <ESC>[0m, etc.
The xterm color intensity values can be influenced by the Dim/Bold/Normal modes (which can be set with e.g. <ESC>[1m, textattrib(), etc), so the actual RGB values of these colors allow room for Dim/Bold to influence their brightness. For instance, "Normal Red" is not full brightness to allow "Bold Red" to be brighter. This goes for all colors except 'Black', which is not influenced by Dim or Bold; Black is always Black.
The 8 color xterm values are:

- 0 = Black
- 1 = Red
- 2 = Green
- 3 = Yellow
- 4 = Blue
- 5 = Magenta
- 6 = Cyan
- 7 = White

See also

textbgcolor_default(Fl_Color)

13.134.5.68 textcolor()

```
void Fl_Terminal::textcolor (
            Fl_Color val )
```

Set the text color for the terminal.
This is a convenience method that sets *both* textfgcolor() and textfgcolor_default(), ensuring both are set to the same value.
Colors set this way will NOT be influenced by the xterm Dim/Bold color intensity attributes. For that, use textfgcolor_xterm() instead.

See also

textfgcolor(Fl_Color), textfgcolor_default(Fl_Color), textbgcolor_xterm(uchar)

13.134.5.69 textfgcolor()

```
void Fl_Terminal::textfgcolor (
            Fl_Color val )
```

Set text foreground drawing color to fltk color `val` used by any new text added.
Use this for temporary color changes, similar to <ESC>[38;2;<R>;<G>;<B>m
Colors set this way will NOT be influenced by the xterm Dim/Bold color intensity attributes. For that, use textfgcolor_xterm(uchar) instead.
This setting does *not* affect the 'default' text colors used by <ESC>[0m, <ESC>c, reset_terminal(), etc. To change both the current *and* default fg color, also use textfgcolor_default(Fl_Color). Example:

```
// Set both 'current' and 'default' colors
Fl_Color amber = 0xd0704000;
tty->textfgcolor(amber);         // set 'current' fg color
tty->textfgcolor_default(amber); // set 'default' fg color used by ESC[0m reset
```

See also

textfgcolor_default(Fl_Color), textfgcolor_xterm(uchar)

13.134.5.70 textfgcolor_default() [1/2]

```
void Fl_Terminal::textfgcolor_default (
            Fl_Color val )
```

Set the default text foreground color used by <ESC>c, <ESC>[0m, and reset_terminal().
Does not affect the 'current' text foreground color; use textfgcolor(Fl_Color) to set that.

See also

textfgcolor(Fl_Color)

13.134.5.71 textfgcolor_default() [2/2]

```
Fl_Color Fl_Terminal::textfgcolor_default (
            void ) const [inline]
```

Return text's default foreground color.

See also

textfgcolor()

13.134.5.72 textfgcolor_xterm()

```
void Fl_Terminal::textfgcolor_xterm (
            uchar val )
```

Sets the foreground text color as one of the 8 'xterm color' values.
This will be the foreground color used for all newly printed text, similar to the <ESC>[#m escape sequence, where # is between 30 and 37.
This color will be reset to the default fg color if reset_terminal() is called, or by <ESC>c, <ESC>[0m, etc.
The xterm color intensity values can be influenced by the Dim/Bold/Normal modes (which can be set with e.g. <ESC>[1m, textattrib(), etc), so the actual RGB values of these colors allow room for Dim/Bold to influence their brightness. For instance, "Normal Red" is not full brightness to allow "Bold Red" to be brighter. This goes for all colors except 'Black', which is not influenced by Dim or Bold; Black is always Black.
The 8 color xterm values are:

- 0 = Black
- 1 = Red
- 2 = Green
- 3 = Yellow
- 4 = Blue
- 5 = Magenta
- 6 = Cyan
- 7 = White

See also

textfgcolor_default(Fl_Color)

13.134.5.73 textfont()

```
void Fl_Terminal::textfont (
            Fl_Font val )
```

Sets the font used for all text displayed in the terminal.
This affects all existing text (in display and history) as well as any newly printed text.
Only monospace fonts are recommended, such as FL_COURIER or FL_SCREEN. Custom fonts configured with Fl::set_font() will also work, as long as they are monospace.

13.134.5.74 textsize()

```
void Fl_Terminal::textsize (
            Fl_Fontsize val )
```

Sets the font size used for all text displayed in the terminal.
This affects all existing text (in display and history) as well as any newly printed text.
Changing this will affect the display_rows() and display_columns().

13.134.5.75 u8c_disp_row()

```
Fl_Terminal::Utf8Char * Fl_Terminal::u8c_disp_row (
            int drow )  [protected]
```

Return pointer to the first u8c character in row `drow` of the display.

- 'drow' is indexed relative to the beginning of the display buffer.

- This can be used to walk all columns in the specified row, e.g.
```
// Print all chars in first row of display (ASCII and UTF-8)
Utf8Char *u8c = u8c_disp_row(0);            // first char of first display row
int scol = 0, ecol = disp_cols();           // start/end for column loop
for (int col=scol; col<ecol; col++,u8c++) { // loop from first char to last
    char *text = u8c->text_utf8();          // text string for char
    int len    = u8c->length();             // text string length for char
    ::printf("<%.*s>", len, text);          // print potentially multibyte char
}
```
- This can also be used to walk all rows on the display screen up to the cursor row, e.g.
```
// Write all chars in display up to cursor row to stdout
for (int row=0; row<disp_rows() && row<=cursor_row(); row++) {
    const Utf8Char *u8c = u8c_disp_row(row);                // first char in row
    for (int col=0; col<disp_cols(); col++,u8c++) {         // walk columns left-to-right
        // ..Do things here with each u8c char..
        ::printf("%.*s", u8c->length(), u8c->text_utf8());  // write each utf8 char to stdout
    }
    ::printf("\n");
}
```

See also

u8c_hist_use_row() for examples of walking the screen history

13.134.5.76 u8c_hist_row()

```
Fl_Terminal::Utf8Char * Fl_Terminal::u8c_hist_row (
            int hrow )  [protected]
```

Return u8c for beginning of a row inside the scrollback history.
'hrow' is indexed relative to the beginning of the scrollback history buffer.

See also

u8c_disp_row(int) for example use.

13.134.5.77 u8c_hist_use_row()

```
Fl_Terminal::Utf8Char * Fl_Terminal::u8c_hist_use_row (
            int hurow )  [protected]
```

Return u8c for beginning of row `hurow` inside the 'in use' part of the scrollback history.
'hurow' is indexed relative to the beginning of the 'in use' part of the scrollback history buffer. This may be different from u8c_hist_row(int) if the history was recently cleared, and there aren't many (or any) rows in the history buffer that have been populated with scrollback text yet.
Example to walk all "in use" lines of the history buffer:
```
// Walk the entire screen history ("in use") and display to stdout
for (int row=0; row<hist_use(); row++) {
    const Utf8Char *u8c = u8c_hist_use_row(row);            // first char in row
    for (int col=0; col<hist_cols(); col++,u8c++) {         // walk columns left-to-right
        // ..Do things here with each u8c char..
        ::printf("%.*s", u8c->length(), u8c->text_utf8());  // show each utf8 char to stdout
    }
    ::printf("\n"); // end of each line
}
```

See also

u8c_disp_row(int) for example use.

13.134.5.78 u8c_ring_row()

```
Fl_Terminal::Utf8Char * Fl_Terminal::u8c_ring_row (
            int grow )  [protected]
```

Return UTF-8 char for row `grow` in the ring buffer.
`grow` is globally indexed relative to the beginning of the ring buffer, so this method can access ANY character in the entire ring buffer (hist or disp) by the global index.
Scrolling offset is NOT applied; this is raw access to the ring's rows.
Should really ONLY be used for making a complete copy of the ring.

Example:

```
// Walk ALL rows and cols in the raw ring buffer..
//    These will not necessarily be in order as they appear
//    on-screen.
//    For walking the terminal lines in order, see examples
//    for u8c_hist_use_row() and u8c_disp_row().
//
for (int row=0; row<ring_rows(); row++) {
    Utf8Char *u8c = u8c_ring_row(row);
    for (int col=0; col<ring_cols(); col++,u8c++) {
        ..make use of u8c->xxx() methods..
    }
}
```

13.134.5.79 utf8_char_at_disp()

```
const Fl_Terminal::Utf8Char * Fl_Terminal::utf8_char_at_disp (
            int drow,
            int dcol ) const  [protected]
```

Return Utf8Char* for char at specified display row and column.
This accesses any character in the display part of the ring buffer.
No range checking done on drow,dcol:

- `drow` must be in range 0..(disp_rows()-1)
- `dcol` must be in range 0..(disp_cols()-1)

See also

u8c_disp_row()

13.134.5.80 utf8_char_at_glob()

```
const Fl_Terminal::Utf8Char * Fl_Terminal::utf8_char_at_glob (
            int grow,
            int gcol ) const  [protected]
```

Return Utf8Char* for char at specified global (grow,gcol).
This accesses any character in the ring buffer (history + display).
No range checking done on grow,gcol:

- `grow` must be in range 0..(ring_rows()-1)
- `gcol` must be in range 0..(ring_cols()-1)

See also

u8c_ring_row()

13.134.5.81 vprintf()

```
void Fl_Terminal::vprintf (
            const char * fmt,
            va_list ap )
```

Appends printf formatted messages to the terminal.
Subclasses can use this to implement their own printf() functionality.
The string can contain UTF-8, crlf's, and ANSI sequences are also supported when ansi(bool) is set to 'true'.

Note

The expanded string is currently limited to 1024 characters (including NULL). For longer strings use append() which has no string limits.

Parameters

fmt	is a printf format string for the message text.
ap	is a va_list created by va_start() and closed with va_end(), which the caller is responsible for handling.

13.134.5.82 w_to_col()

```
int Fl_Terminal::w_to_col (
            int W ) const  [protected]
```

Given a width in pixels, return number of columns that "fits" into that area.
This is used by the constructor to size the row/cols to fit the widget size.

13.134.5.83 walk_selection()

```
const Fl_Terminal::Utf8Char * Fl_Terminal::walk_selection (
            const Utf8Char * u8c,
            int & row,
            int & col ) const  [protected]
```

Walk the mouse selection one character at a time from beginning to end, returning a Utf8Char∗ to the next character in the selection, or NULL if the end was reached, or if there's no selection.
This is easier to use for walking the selection than get_selection().
`u8c` should start out as NULL, rewinding to the beginning of the selection. If the returned Utf8Char∗ is not NULL, `row` and `col` return the character's row/column position in the ring buffer.

```
// EXAMPLE: Walk the entire mouse selection, if any
int row,col;                                  // the returned row/col for each char
Utf8Char *u8c = NULL;                         // start with NULL to begin walk
while ((u8c = walk_selection(u8c, row, col))) { // loop until end char reached
  ..do something with *u8c..
}
```

See also

get_selection(), is_selection()

Parameters

u8c	NULL on first iter
row	returned row#
col	returned col#

13.134.6 Member Data Documentation

13.134.6.1 hscrollbar

```
Fl_Scrollbar* Fl_Terminal::hscrollbar
```

Horizontal scrollbar.
This is public so it can be accessed directly, e.g.

- hscrollbar->value() returns the column offset position from the left edge of the display; 0 being the left edge (default).
- hscrollbar->value(int) similarly sets the column offset, which should be in the range [0 .. Fl_Scrollbar::maximum()].
- hscrollbar->step(double) sets the smoothness of scrolling, default is 0.25 for 4 steps of motion per column.

13.134.6.2 scrollbar

`Fl_Scrollbar* Fl_Terminal::scrollbar`
Vertical scrollbar.
This is public so it can be accessed directly, e.g.

- scrollbar->value() returns the row offset from the bottom of the display, 0 being the bottom (default).
- scrollbar->value(int) similarly sets the row offset, which should be in the range [0 .. Fl_Scrollbar::maximum()].
- scrollbar->step(double) sets the smoothness of scrolling, default is 0.25 for 4 steps of motion per row.

Todo Support scrollbar_left/right() - See Fl_Browser_::scrollbar docs
Support new ScrollbarStyle

The documentation for this class was generated from the following files:

- Fl_Terminal.H
- Fl_Terminal.cxx

13.135 Fl_Text_Buffer Class Reference

This class manages Unicode text displayed in one or more Fl_Text_Display widgets.
`#include <Fl_Text_Buffer.H>`

Public Member Functions

- void add_modify_callback (Fl_Text_Modify_Cb bufModifiedCB, void *cbArg)

  Adds a callback function that is called whenever the text buffer is modified.
- void **add_predelete_callback** (Fl_Text_Predelete_Cb bufPredelCB, void *cbArg)

  Adds a callback routine to be called before text is deleted from the buffer.
- char * address (int pos)

  Convert a byte offset in buffer into a memory address.
- const char * address (int pos) const

  Convert a byte offset in buffer into a memory address.
- void append (const char *t, int addedLength=-1)

  Appends the text string to the end of the buffer.
- int appendfile (const char *file, int buflen=128 *1024)

  Appends the named file to the end of the buffer.
- char byte_at (int pos) const

  Returns the raw byte at the specified position pos in the buffer.
- void **call_modify_callbacks** ()

  Calls all modify callbacks that have been registered using the add_modify_callback() method.
- void **call_predelete_callbacks** ()

  Calls the stored pre-delete callback procedure(s) for this buffer to update the changed area(s) on the screen and any other listeners.
- bool can_redo () const

  Check if undo is enabled and if the last undo action can be redone.
- bool can_undo () const

  Check if undo is enabled and if the last action can be undone.
- void canUndo (char flag=1)

  Enable or disable undo actions for this text buffer.
- unsigned int char_at (int pos) const

  Returns the character at the specified position `pos` in the buffer.
- void copy (Fl_Text_Buffer *fromBuf, int fromStart, int fromEnd, int toPos)

Copies text from another Fl_Text_Buffer to this one.

- int count_displayed_characters (int lineStartPos, int targetPos) const

  Count the number of displayed characters between buffer position `lineStartPos` and `targetPos`.
- int count_lines (int startPos, int endPos) const

  Counts the number of newlines between `startPos` and `endPos` in buffer.
- int estimate_lines (int startPos, int endPos, int lineLen) const

  Estimate the number of newlines between `startPos` and `endPos` in buffer.
- int findchar_backward (int startPos, unsigned int searchChar, int *foundPos) const

  Search backwards in buffer `buf` for character `searchChar`, starting with the character before `startPos`, returning the result in `foundPos`.
- int findchar_forward (int startPos, unsigned searchChar, int *foundPos) const

  Finds the next occurrence of the specified character.
- Fl_Text_Buffer (int requestedSize=0, int preferredGapSize=1024)

  Create an empty text buffer of a pre-determined size.
- int **highlight** () const

  Returns a non-zero value if text has been highlighted, 0 otherwise.
- void **highlight** (int start, int end)

  Highlights the specified text within the buffer.
- int **highlight_position** (int *start, int *end) const

  Highlights the specified text between `start` and `end` within the buffer.
- const Fl_Text_Selection * **highlight_selection** () const

  Returns the current highlight selection.
- char * highlight_text () const

  Returns the highlighted text.
- void insert (int pos, const char *text, int insertedLength=-1)

  Inserts null-terminated string `text` at position `pos`.
- int insertfile (const char *file, int pos, int buflen=128 *1024)

  Inserts a file at the specified position.
- bool is_word_separator (int pos) const

  Returns whether character at position `pos` is a word separator.
- int length () const

  Returns the number of bytes in the buffer.
- int line_end (int pos) const

  Finds and returns the position of the end of the line containing position `pos` (which is either a pointer to the newline character ending the line or a pointer to one character beyond the end of the buffer).
- int line_start (int pos) const

  Returns the position of the start of the line containing position `pos`.
- char * line_text (int pos) const

  Returns the text from the entire line containing the specified character position.
- int loadfile (const char *file, int buflen=128 *1024)

  Loads a text file into the buffer.
- int next_char (int pos) const

  Returns the index of the next character.
- int **next_char_clipped** (int pos) const
- int outputfile (const char *file, int start, int end, int buflen=128 *1024)

  Writes the specified portions of the text buffer to a file.
- int prev_char (int pos) const

  Returns the index of the previous character.
- int **prev_char_clipped** (int pos) const

  As prev_char() but returns 0 if the beginning of the buffer is reached.
- Fl_Text_Selection * **primary_selection** ()

Returns the primary selection.

- const Fl_Text_Selection * **primary_selection** () const

  Returns the primary selection.
- void printf (const char *fmt,...)

  Appends printf formatted messages to the end of the buffer.
- int **redo** (int *cp=0)

  Redo previous undo action.
- void remove (int start, int end)

  Deletes a range of characters in the buffer.
- void **remove_modify_callback** (Fl_Text_Modify_Cb bufModifiedCB, void *cbArg)

  Removes a modify callback.
- void **remove_predelete_callback** (Fl_Text_Predelete_Cb predelCB, void *cbArg)

  Removes a callback routine `bufPreDeleteCB` associated with argument `cbArg` to be called before text is deleted from the buffer.
- void **remove_secondary_selection** ()

  Removes the text from the buffer corresponding to the secondary text selection object.
- void **remove_selection** ()

  Removes the text in the primary selection.
- void replace (int start, int end, const char *text, int insertedLength=-1)

  Deletes the characters between `start` and `end`, and inserts the null-terminated string `text` in their place in the buffer.
- void **replace_secondary_selection** (const char *text)

  Replaces the text from the buffer corresponding to the secondary text selection object with the new string `text`.
- void **replace_selection** (const char *text)

  Replaces the text in the primary selection.
- int rewind_lines (int startPos, int nLines) const

  Finds and returns the position of the first character of the line `nLines` backwards from `startPos` (not counting the character pointed to by `startpos` if that is a newline) in the buffer.
- int savefile (const char *file, int buflen=128 *1024)

  Saves a text file from the current buffer.
- int search_backward (int startPos, const char *searchString, int *foundPos, int matchCase=0) const

  Search backwards in buffer for string `searchString`, starting with the character at `startPos`, returning the result in `foundPos`.
- int search_forward (int startPos, const char *searchString, int *foundPos, int matchCase=0) const

  Search forwards in buffer for string `searchString`, starting with the character `startPos`, and returning the result in `foundPos`.
- void **secondary_select** (int start, int end)

  Selects a range of characters in the secondary selection.
- int **secondary_selected** () const

  Returns a non-zero value if text has been selected in the secondary text selection, 0 otherwise.
- const Fl_Text_Selection * **secondary_selection** () const

  Returns the secondary selection.
- int **secondary_selection_position** (int *start, int *end) const

  Returns the current selection in the secondary text selection object.
- char * secondary_selection_text () const

  Returns the text in the secondary selection.
- void **secondary_unselect** ()

  Clears any selection in the secondary text selection object.
- void **select** (int start, int end)

  Selects a range of characters in the buffer.
- int **selected** () const

  Returns a non-zero value if text has been selected, 0 otherwise.

- int **selection_position** (int *start, int *end) const

  Gets the selection position.
- char * selection_text () const

  Returns the currently selected text.
- int skip_displayed_characters (int lineStartPos, int nChars) const

  Count forward from buffer position `startPos` in displayed characters.
- int **skip_lines** (int startPos, int nLines) const

  Finds the first character of the line `nLines` forward from `startPos` in the buffer and returns its position.
- int tab_distance () const

  Gets the tab width.
- void **tab_distance** (int tabDist)

  Set the hardware tab distance (width) used by all displays for this buffer, and used in computing offsets for rectangular selection operations.
- char * text () const

  Get a copy of the entire contents of the text buffer.
- void text (const char *text)

  Replaces the entire contents of the text buffer.
- char * text_range (int start, int end) const

  Get a copy of a part of the text buffer.
- std::string text_str () const

  Get a copy of the entire contents of the text buffer.
- int undo (int *cp=0)

  Undo text modification according to the undo variables or insert text from the undo buffer.
- void **unhighlight** ()

  Unhighlights text in the buffer.
- void **unselect** ()

  Cancels any previous selection on the primary text selection object.
- int **utf8_align** (int) const

  Align an index into the buffer to the current or previous UTF-8 boundary.
- void vprintf (const char *fmt, va_list ap)

  Can be used by subclasses that need their own printf() style functionality.
- int word_end (int pos) const

  Returns the position corresponding to the end of the word.
- int word_start (int pos) const

  Returns the position corresponding to the start of the word.
- ~**Fl_Text_Buffer** ()

  Frees a text buffer.

Public Attributes

- int **input_file_was_transcoded**

  true if the loaded file has been transcoded to UTF-8.
- void(* transcoding_warning_action)(Fl_Text_Buffer *)

  Pointer to a function called after reading a non UTF-8 encoded file.

Static Public Attributes

- static const char * file_encoding_warning_message

  This message may be displayed using the fl_alert() function when a file which was not UTF-8 encoded is input.

Protected Member Functions

- int **apply_undo** (Fl_Text_Undo_Action *action, int *cursorPos)

  Apply the current undo/redo operation, called from undo() or redo().
- void **call_modify_callbacks** (int pos, int nDeleted, int nInserted, int nRestyled, const char *deletedText) const

  Calls the stored modify callback procedure(s) for this buffer to update the changed area(s) on the screen and any other listeners.
- void **call_predelete_callbacks** (int pos, int nDeleted) const

  Calls the stored pre-delete callback procedure(s) for this buffer to update the changed area(s) on the screen and any other listeners.
- int insert_ (int pos, const char *text, int insertedLength=-1)

  Internal (non-redisplaying) version of insert().
- void **move_gap** (int pos)

  Move the gap to start at a new position.
- void **reallocate_with_gap** (int newGapStart, int newGapLen)

  Reallocates the text storage in the buffer to have a gap starting at `newGapStart` and a gap size of `newGapLen`, preserving the buffer's current contents.
- void **redisplay_selection** (Fl_Text_Selection *oldSelection, Fl_Text_Selection *newSelection) const

  Calls the stored redisplay procedure(s) for this buffer to update the screen for a change in a selection.
- void remove_ (int start, int end)

  Internal (non-redisplaying) version of remove().
- void **remove_selection_** (Fl_Text_Selection *sel)

  Removes the text from the buffer corresponding to `sel`.
- void **replace_selection_** (Fl_Text_Selection *sel, const char *text)

  Replaces the `text` in selection `sel`.
- char * **selection_text_** (const Fl_Text_Selection *sel) const
- void **update_selections** (int pos, int nDeleted, int nInserted)

  Updates all of the selections in the buffer for changes in the buffer's text.

Protected Attributes

- char * **mBuf**

  allocated memory where the text is stored
- char **mCanUndo**

  if this buffer is used for attributes, it must not do any undo calls
- void ** **mCbArgs**

  caller arguments for modifyProcs above
- int **mCursorPosHint**

  hint for reasonable cursor position after a buffer modification operation
- int **mGapEnd**

  points to the first character after the gap
- int **mGapStart**

  points to the first character of the gap
- Fl_Text_Selection **mHighlight**

  highlighted areas
- int **mLength**

  length of the text in the buffer (the length of the buffer itself must be calculated: gapEnd - gapStart + length)
- Fl_Text_Modify_Cb * **mModifyProcs**

  procedures to call when buffer is modified to redisplay contents
- int **mNModifyProcs**

  number of modify-redisplay procs attached
- int **mNPredeleteProcs**

number of pre-delete procs attached

- void ** **mPredeleteCbArgs**

  caller argument for pre-delete proc above

- Fl_Text_Predelete_Cb * **mPredeleteProcs**

  procedure to call before text is deleted from the buffer; at most one is supported.

- int **mPreferredGapSize**

  the default allocation for the text gap is 1024 bytes and should only be increased if frequent and large changes in buffer size are expected

- Fl_Text_Selection **mPrimary**

  highlighted areas

- Fl_Text_Undo_Action_List * **mRedoList**

  List of redo event.

- Fl_Text_Selection **mSecondary**

  highlighted areas

- int mTabDist

  equiv.

- Fl_Text_Undo_Action * **mUndo**

  local undo event

- Fl_Text_Undo_Action_List * **mUndoList**

  List of undo event.

13.135.1 Detailed Description

This class manages Unicode text displayed in one or more Fl_Text_Display widgets.
All text in Fl_Text_Buffer must be encoded in UTF-8. All indices used in the function calls must be aligned to the start of a UTF-8 sequence. All indices and pointers returned will be aligned. All functions that return a single character will return that in an unsigned int in UCS-4 encoding.
The Fl_Text_Buffer class is used by the Fl_Text_Display and Fl_Text_Editor to manage complex text data and is based upon the excellent NEdit text editor engine - see https://sourceforge.↩net/projects/nedit/.

13.135.2 Constructor & Destructor Documentation

13.135.2.1 Fl_Text_Buffer()

```
Fl_Text_Buffer::Fl_Text_Buffer (
            int requestedSize = 0,
            int preferredGapSize = 1024 )
```

Create an empty text buffer of a pre-determined size.

Parameters

requestedSize	use this to avoid unnecessary re-allocation if you know exactly how much the buffer will need to hold
preferredGapSize	Initial size for the buffer gap (empty space in the buffer where text might be inserted if the user is typing sequential characters)

13.135.3 Member Function Documentation

13.135.3.1 add_modify_callback()

```
void Fl_Text_Buffer::add_modify_callback (
            Fl_Text_Modify_Cb bufModifiedCB,
            void * cbArg )
```

Adds a callback function that is called whenever the text buffer is modified.

The callback function is declared as follows:

```
typedef void (*Fl_Text_Modify_Cb)(int pos, int nInserted, int nDeleted,
   int nRestyled, const char* deletedText,
   void* cbArg);
```

13.135.3.2 address() [1/2]

```
char * Fl_Text_Buffer::address (
            int pos )  [inline]
```

Convert a byte offset in buffer into a memory address.

Parameters

pos	byte offset into buffer

Returns

byte offset converted to a memory address

13.135.3.3 address() [2/2]

```
const char * Fl_Text_Buffer::address (
            int pos ) const  [inline]
```

Convert a byte offset in buffer into a memory address.

Parameters

pos	byte offset into buffer

Returns

byte offset converted to a memory address

13.135.3.4 append()

```
void Fl_Text_Buffer::append (
            const char * t,
            int addedLength = -1 )  [inline]
```

Appends the text string to the end of the buffer.

Parameters

t	UTF-8 encoded text
addedLength	number of bytes to append, or -1 to indicate `t` is null-terminated

13.135.3.5 appendfile()

```
int Fl_Text_Buffer::appendfile (
            const char * file,
            int buflen = 128*1024 )  [inline]
```

Appends the named file to the end of the buffer.
See also insertfile().

13.135.3.6 byte_at()

```
char Fl_Text_Buffer::byte_at (
            int pos ) const
```

Returns the raw byte at the specified position pos in the buffer.
Positions start at 0.

Parameters

pos	byte offset into buffer

Returns

unencoded raw byte

13.135.3.7 can_redo()

```
bool Fl_Text_Buffer::can_redo ( ) const
```

Check if undo is enabled and if the last undo action can be redone.
Check if undo is anabled and if the last undo action can be redone.

See also

canUndo()

13.135.3.8 can_undo()

```
bool Fl_Text_Buffer::can_undo ( ) const
```

Check if undo is enabled and if the last action can be undone.

See also

canUndo()

13.135.3.9 canUndo()

```
void Fl_Text_Buffer::canUndo (
            char flag = 1 )
```

Enable or disable undo actions for this text buffer.
Undo actions are enable for text buffer by default. If used as a style buffer in Fl_Text_Display, undo actions are disabled as they are handled by the text buffer.

See also

can_undo()

13.135.3.10 char_at()

```
unsigned int Fl_Text_Buffer::char_at (
            int pos ) const
```

Returns the character at the specified position `pos` in the buffer.
Positions start at 0.

Parameters

pos	byte offset into buffer, `pos` must be at a UTF-8 character boundary

Returns

Unicode UCS-4 encoded character

13.135.3.11 copy()

```
void Fl_Text_Buffer::copy (
            Fl_Text_Buffer * fromBuf,
            int fromStart,
            int fromEnd,
            int toPos )
```

Copies text from another Fl_Text_Buffer to this one.

Parameters

fromBuf	source text buffer, may be the same as this
fromStart	byte offset into buffer
fromEnd	byte offset into buffer
toPos	destination byte offset into buffer

13.135.3.12 count_displayed_characters()

```
int Fl_Text_Buffer::count_displayed_characters (
            int lineStartPos,
            int targetPos ) const
```

Count the number of displayed characters between buffer position `lineStartPos` and `targetPos`.
Displayed characters are the characters shown on the screen to represent characters in the buffer, where tabs and control characters are expanded.

13.135.3.13 count_lines()

```
int Fl_Text_Buffer::count_lines (
            int startPos,
            int endPos ) const
```

Counts the number of newlines between `startPos` and `endPos` in buffer.
The character at position `endPos` is not counted.

13.135.3.14 estimate_lines()

```
int Fl_Text_Buffer::estimate_lines (
            int startPos,
            int endPos,
            int lineLen ) const
```

Estimate the number of newlines between `startPos` and `endPos` in buffer.
This call takes line wrapping into account. It assumes a line break at every `lineLen` characters after the beginning of a line.

13.135.3.15 findchar_backward()

```
int Fl_Text_Buffer::findchar_backward (
            int startPos,
            unsigned int searchChar,
            int * foundPos ) const
```

Search backwards in buffer `buf` for character `searchChar`, starting with the character *before* `startPos`, returning the result in `foundPos`.
Returns 1 if found, 0 if not. The difference between this and search_backward() is that it's optimized for single characters. The overall performance of the text widget is dependent on its ability to count lines quickly, hence searching for a single character: newline.

Parameters

startPos	byte offset to start position
searchChar	UCS-4 character that we want to find
foundPos	byte offset where the character was found

Returns

1 if found, 0 if not

13.135.3.16 findchar_forward()

```
int Fl_Text_Buffer::findchar_forward (
            int startPos,
            unsigned searchChar,
            int * foundPos ) const
```

Finds the next occurrence of the specified character.
Search forwards in buffer for character `searchChar`, starting with the character `startPos`, and returning the result in `foundPos`. Returns 1 if found, 0 if not. The difference between this and search_forward() is that it's optimized for single characters. The overall performance of the text widget is dependent on its ability to count lines quickly, hence searching for a single character: newline.

Parameters

startPos	byte offset to start position
searchChar	UCS-4 character that we want to find
foundPos	byte offset where the character was found

Returns

1 if found, 0 if not

13.135.3.17 highlight_text()

```
char * Fl_Text_Buffer::highlight_text ( ) const
```

Returns the highlighted text.
When you are done with the text, free it using the free() function.

13.135.3.18 insert()

```
void Fl_Text_Buffer::insert (
            int pos,
            const char * text,
            int insertedLength = -1 )
```

Inserts null-terminated string `text` at position `pos`.

Parameters

pos	insertion position as byte offset (must be UTF-8 character aligned)
text	UTF-8 encoded text
insertedLength	number of bytes to insert, or -1 to indicate `text` is null-terminated

13.135.3.19 insert_()

```
int Fl_Text_Buffer::insert_ (
            int pos,
            const char * text,
            int insertedLength = -1 )  [protected]
```

Internal (non-redisplaying) version of insert().
Returns the length of text inserted (this is just strlen(`text`) if `insertedLength` == -1, however this calculation can be expensive and the length will be required by any caller who will continue on to call redisplay). `pos` must be contiguous with the existing text in the buffer (i.e. not past the end).

Returns

the number of bytes inserted

13.135.3.20 insertfile()

```
int Fl_Text_Buffer::insertfile (
            const char * file,
            int pos,
            int buflen = 128*1024 )
```

Inserts a file at the specified position.
Returns

- 0 on success
- non-zero on error (strerror() contains reason)
- 1 indicates open for read failed (no data loaded)
- 2 indicates error occurred while reading data (data was partially loaded)

File can be UTF-8 or CP1252 encoded. If the input file is not UTF-8 encoded, the Fl_Text_Buffer widget will contain data transcoded to UTF-8. By default, the message Fl_Text_Buffer::file_encoding_warning_message will warn the user about this.

See also

input_file_was_transcoded and transcoding_warning_action.

13.135.3.21 is_word_separator()

```
bool Fl_Text_Buffer::is_word_separator (
            int pos ) const
```

Returns whether character at position `pos` is a word separator.
Pos must be at a character boundary.

13.135.3.22 length()

```
int Fl_Text_Buffer::length ( ) const  [inline]
```

Returns the number of bytes in the buffer.

Returns

size of text in bytes

13.135.3.23 line_end()

```
int Fl_Text_Buffer::line_end (
            int pos ) const
```

Finds and returns the position of the end of the line containing position `pos` (which is either a pointer to the newline character ending the line or a pointer to one character beyond the end of the buffer).

Parameters

pos	byte index into buffer

Returns

byte offset to line end

13.135.3.24 line_start()

```
int Fl_Text_Buffer::line_start (
            int pos ) const
```

Returns the position of the start of the line containing position `pos`.

Parameters

pos	byte index into buffer

Returns

byte offset to line start

13.135.3.25 line_text()

```
char * Fl_Text_Buffer::line_text (
            int pos ) const
```

Returns the text from the entire line containing the specified character position.
When you are done with the text, free it using the free() function.

Parameters

pos	byte index into buffer

Returns

copy of UTF-8 text, must be free'd

13.135.3.26 loadfile()

```
int Fl_Text_Buffer::loadfile (
            const char * file,
            int buflen = 128*1024 )  [inline]
```

Loads a text file into the buffer.
See also insertfile().

13.135.3.27 next_char()

```
int Fl_Text_Buffer::next_char (
            int pos ) const
```

Returns the index of the next character.
This function processes an emoji sequence (see fl_utf8_next_composed_char) as a single character. Returns length() if the end of the buffer is reached.

Parameters

in	*pos*	index to the current character

13.135.3.28 outputfile()

```
int Fl_Text_Buffer::outputfile (
            const char * file,
            int start,
            int end,
            int buflen = 128*1024 )
```

Writes the specified portions of the text buffer to a file.
Returns

- 0 on success
- non-zero on error (strerror() contains reason)
- 1 indicates open for write failed (no data saved)
- 2 indicates error occurred while writing data (data was partially saved)

See also

savefile(const char *file, int buflen)

13.135.3.29 prev_char()

```
int Fl_Text_Buffer::prev_char (
            int pos ) const
```

Returns the index of the previous character.
This function processes an emoji sequence (see fl_utf8_next_composed_char) as a single character. Returns -1 if the beginning of the buffer is reached.

Parameters

pos	index to the current character

13.135.3.30 printf()

```
void Fl_Text_Buffer::printf (
            const char * fmt,
             ...  )
```

Appends printf formatted messages to the end of the buffer.
Example:

```
#include <FL/Fl_Text_Display.H>
int main(..) {
    :
    // Create a text display widget and assign it a text buffer
    Fl_Text_Display *tdsp = new Fl_Text_Display(..);
    Fl_Text_Buffer *tbuf = new Fl_Text_Buffer();
    tdsp->buffer(tbuf);
    :
    // Append three lines of formatted text to the buffer
    tbuf->printf("The current date is: %s.\nThe time is: %s\n", date_str, time_str);
    tbuf->printf("The current PID is %ld.\n", (long)getpid());
    :
```

Note

The expanded string is currently limited to 1024 characters.

Parameters

in	*fmt*	is a printf format string for the message text.

13.135.3.31 remove()

```
void Fl_Text_Buffer::remove (
            int start,
            int end )
```

Deletes a range of characters in the buffer.

Parameters

start	byte offset to first character to be removed
end	byte offset to character after last character to be removed

13.135.3.32 remove_()

```
void Fl_Text_Buffer::remove_ (
            int start,
            int end )  [protected]
```

Internal (non-redisplaying) version of remove().
Removes the contents of the buffer between `start` and `end` (and moves the gap to the site of the delete).

13.135.3.33 replace()

```
void Fl_Text_Buffer::replace (
            int start,
            int end,
            const char * text,
            int insertedLength = -1 )
```

Deletes the characters between `start` and `end`, and inserts the null-terminated string `text` in their place in the buffer.

Parameters

start	byte offset to first character to be removed and new insert position
end	byte offset to character after last character to be removed
text	UTF-8 encoded text
insertedLength	number of bytes to insert, or -1 to indicate `text` is null-terminated

13.135.3.34 rewind_lines()

```
int Fl_Text_Buffer::rewind_lines (
            int startPos,
            int nLines ) const
```

Finds and returns the position of the first character of the line `nLines` backwards from `startPos` (not counting the character pointed to by `startpos` if that is a newline) in the buffer.
`nLines` == 0 means find the beginning of the line.

13.135.3.35 savefile()

```
int Fl_Text_Buffer::savefile (
            const char * file,
            int buflen = 128*1024 )  [inline]
```

Saves a text file from the current buffer.
Returns

- 0 on success
- non-zero on error (strerror() contains reason)

- 1 indicates open for write failed (no data saved)
- 2 indicates error occurred while writing data (data was partially saved)

See also

outputfile(const char *file, int start, int end, int buflen)

13.135.3.36 search_backward()

```
int Fl_Text_Buffer::search_backward (
            int startPos,
            const char * searchString,
            int * foundPos,
            int matchCase = 0 ) const
```

Search backwards in buffer for string `searchString`, starting with the character *at* `startPos`, returning the result in `foundPos`.
Returns 1 if found, 0 if not.

Parameters

startPos	byte offset to start position
searchString	UTF-8 string that we want to find
foundPos	byte offset where the string was found
matchCase	if set, match character case

Returns

1 if found, 0 if not

13.135.3.37 search_forward()

```
int Fl_Text_Buffer::search_forward (
            int startPos,
            const char * searchString,
            int * foundPos,
            int matchCase = 0 ) const
```

Search forwards in buffer for string `searchString`, starting with the character `startPos`, and returning the result in `foundPos`.
Returns 1 if found, 0 if not.

Parameters

startPos	byte offset to start position
searchString	UTF-8 string that we want to find
foundPos	byte offset where the string was found
matchCase	if set, match character case

Returns

1 if found, 0 if not

13.135.3.38 secondary_selection_text()

```
char * Fl_Text_Buffer::secondary_selection_text ( ) const
```

Returns the text in the secondary selection.
When you are done with the text, free it using the free() function.

13.135.3.39 selection_text()

```
char * Fl_Text_Buffer::selection_text ( ) const
```

Returns the currently selected text.
When you are done with the text, free it using the free() function.

13.135.3.40 skip_displayed_characters()

```
int Fl_Text_Buffer::skip_displayed_characters (
            int lineStartPos,
            int nChars ) const
```

Count forward from buffer position `startPos` in displayed characters.
Displayed characters are the characters shown on the screen to represent characters in the buffer, where tabs and control characters are expanded.

Parameters

lineStartPos	byte offset into buffer
nChars	number of bytes that are sent to the display

Returns

byte offset in input after all output bytes are sent

13.135.3.41 tab_distance()

```
int Fl_Text_Buffer::tab_distance ( ) const  [inline]
```

Gets the tab width.
The tab width is measured in characters. The pixel position is calculated using an average character width.

13.135.3.42 text() [1/2]

```
char * Fl_Text_Buffer::text ( ) const
```

Get a copy of the entire contents of the text buffer.
Memory is allocated to contain the returned string, which the caller must free.

Returns

newly allocated text buffer - must be free'd, text is UTF-8

13.135.3.43 text() [2/2]

```
void Fl_Text_Buffer::text (
            const char * text )
```

Replaces the entire contents of the text buffer.

Parameters

text	Text must be valid UTF-8. If null, an empty string is substituted.

13.135.3.44 text_range()

```
char * Fl_Text_Buffer::text_range (
            int start,
```

```
            int end ) const
```

Get a copy of a part of the text buffer.
Return a copy of the text between `start` and `end` character positions from text buffer `buf`. Positions start at 0, and the range does not include the character pointed to by `end`. When you are done with the text, free it using the free() function.

Parameters

start	byte offset to first character
end	byte offset after last character in range

Returns

newly allocated text buffer - must be free'd, text is UTF-8

13.135.3.45 text_str()

```
std::string Fl_Text_Buffer::text_str ( ) const
```

Get a copy of the entire contents of the text buffer.

Returns

text as a UTF-8 string

13.135.3.46 undo()

```
int Fl_Text_Buffer::undo (
            int * cursorPos = 0 )
```

Undo text modification according to the undo variables or insert text from the undo buffer.
Take the previous changes and undo them.
Return the previous cursor position in cursorPos. Returns 1 if the undo was applied. CursorPos will be at a character boundary.

13.135.3.47 vprintf()

```
void Fl_Text_Buffer::vprintf (
            const char * fmt,
            va_list ap )
```

Can be used by subclasses that need their own printf() style functionality.

Note

The expanded string is currently limited to 1024 characters.

Parameters

in	*fmt*	is a printf format string for the message text.
in	*ap*	is a va_list created by va_start() and closed with va_end(), which the caller is responsible for handling.

13.135.3.48 word_end()

```
int Fl_Text_Buffer::word_end (
            int pos ) const
```

Returns the position corresponding to the end of the word.

Parameters

pos	byte index into buffer

Returns

byte offset to word end

13.135.3.49 word_start()

```
int Fl_Text_Buffer::word_start (
            int pos ) const
```

Returns the position corresponding to the start of the word.

Parameters

pos	byte index into buffer

Returns

byte offset to word start

13.135.4 Member Data Documentation

13.135.4.1 file_encoding_warning_message

```
const char * Fl_Text_Buffer::file_encoding_warning_message [static]
```

Initial value:

```
=
    "Displayed text contains the UTF-8 transcoding\n"
    "of the input file which was not UTF-8 encoded.\n"
    "Some changes may have occurred."
```

This message may be displayed using the fl_alert() function when a file which was not UTF-8 encoded is input.

13.135.4.2 mTabDist

```
int Fl_Text_Buffer::mTabDist [protected]
```

equiv.
number of characters in a tab

13.135.4.3 transcoding_warning_action

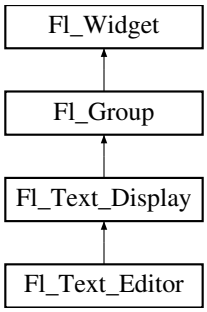
```
void(* Fl_Text_Buffer::transcoding_warning_action) (Fl_Text_Buffer *)
```

Pointer to a function called after reading a non UTF-8 encoded file.
This function is called after reading a file if the file content was transcoded to UTF-8. Its default implementation calls fl_alert() with the text of file_encoding_warning_message. No warning message is displayed if this pointer is set to NULL. Use input_file_was_transcoded to be informed if file input required transcoding to UTF-8.
The documentation for this class was generated from the following files:

- Fl_Text_Buffer.H
- Fl_Text_Buffer.cxx

13.136 Fl_Text_Display Class Reference

Rich text display widget.
`#include <Fl_Text_Display.H>`
Inheritance diagram for Fl_Text_Display:



Classes

- struct Style_Table_Entry

  This structure associates the color, font, and font size of a string to draw with an attribute mask matching attr.

Public Types

- enum {
  NORMAL_CURSOR , CARET_CURSOR , DIM_CURSOR , BLOCK_CURSOR ,
  HEAVY_CURSOR , SIMPLE_CURSOR }

  text display cursor shapes enumeration
- enum { **CURSOR_POS** , **CHARACTER_POS** }

  the character position is the left edge of a character, whereas the cursor is thought to be between the centers of two consecutive characters.
- enum {
  DRAG_NONE = -2 , **DRAG_START_DND** = -1 , **DRAG_CHAR** = 0 , **DRAG_WORD** = 1 ,
  DRAG_LINE = 2 }

  drag types - they match Fl::event_clicks() so that single clicking to start a collection selects by character, double clicking selects by word and triple clicking selects by line.
- enum { WRAP_NONE , WRAP_AT_COLUMN , WRAP_AT_PIXEL , WRAP_AT_BOUNDS }

  wrap types - used in wrap_mode()
- enum {
  ATTR_BGCOLOR = 0x0001 , ATTR_BGCOLOR_EXT_ = 0x0002 , ATTR_BGCOLOR_EXT = 0x0003 ,
  ATTR_UNDERLINE = 0x0004 ,
  ATTR_GRAMMAR = 0x0008 , ATTR_SPELLING = 0x000C , ATTR_STRIKE_THROUGH = 0x0010 ,
  ATTR_LINES_MASK = 0x001C }

  attribute flags in `Style_Table_Entry.attr`
- typedef void(∗ **Unfinished_Style_Cb**) (int, void ∗)

Public Member Functions

- Fl_Text_Buffer ∗ buffer () const

  Gets the current text buffer associated with the text widget.
- void buffer (Fl_Text_Buffer &buf)

  Sets the current text buffer associated with the text widget.
- void buffer (Fl_Text_Buffer ∗buf)

  Attach a text buffer to display, replacing the current buffer (if any).
- double col_to_x (double col) const

  Convert a column number into an x pixel position.
- int count_lines (int start, int end, bool start_pos_is_line_start) const

  Count the number of lines between two positions.
- Fl_Color cursor_color () const

  Gets the text cursor color.
- void cursor_color (Fl_Color n)

Sets the text cursor color.

- int **cursor_style** () const
- void cursor_style (int style)

  Sets the text cursor style.
- virtual void display_needs_recalc ()

  Schedule a recalc_display() to be done on next draw().
- Fl_Text_Display (int X, int Y, int W, int H, const char *l=0)

  Creates a new text display widget.
- int get_absolute_top_line_number () const

  Returns the absolute (non-wrapped) line number of the first line displayed.
- Fl_Color grammar_underline_color () const

  Gets the underline color for style attribute ATTR_GRAMMAR.
- void grammar_underline_color (Fl_Color color)

  Sets the underline color for style attribute ATTR_GRAMMAR.
- int handle (int e) override

  Event handling.
- void **hide_cursor** ()

  Hides the text cursor.
- void highlight_data (Fl_Text_Buffer *styleBuffer, const Style_Table_Entry *styleTable, int nStyles, char unfinishedStyle, Unfinished_Style_Cb unfinishedHighlightCB, void *cbArg)

  Attach (or remove) highlight information in text display and redisplay.
- int in_selection (int x, int y) const

  Check if a pixel position is within the primary selection.
- void insert (const char *text)

  Inserts "text" at the current cursor location.
- int insert_position () const

  Gets the position of the text insertion cursor for text display.
- void insert_position (int newPos)

  Sets the position of the text insertion cursor for text display.
- int line_end (int startPos, bool startPosIsLineStart) const

  Returns the end of a line.
- int line_start (int pos) const

  Return the beginning of a line.
- Fl_Align **linenumber_align** () const

  Returns the alignment used for line numbers (if enabled).
- void linenumber_align (Fl_Align val)

  Set alignment for line numbers (if enabled).
- Fl_Color **linenumber_bgcolor** () const

  Returns the background color used for line numbers (if enabled).
- void linenumber_bgcolor (Fl_Color val)

  Set the background color used for line numbers (if enabled).
- Fl_Color **linenumber_fgcolor** () const

  Return the foreground color used for line numbers (if enabled).
- void linenumber_fgcolor (Fl_Color val)

  Set the foreground color used for line numbers (if enabled).
- Fl_Font **linenumber_font** () const

  Return the font used for line numbers (if enabled).
- void linenumber_font (Fl_Font val)

  Set the font used for line numbers (if enabled).
- const char * **linenumber_format** () const

  Returns the line number printf() format string.

- void linenumber_format (const char *val)

  Sets the printf() style format string used for line numbers.
- Fl_Fontsize **linenumber_size** () const

  Return the font size used for line numbers (if enabled).
- void linenumber_size (Fl_Fontsize val)

  Set the font size used for line numbers (if enabled).
- int **linenumber_width** () const

  Return the screen area width provided for line numbers.
- void linenumber_width (int width)

  Set width of screen area for line numbers.
- int move_down ()

  Moves the current insert position down one line.
- int move_left ()

  Moves the current insert position left one character.
- int move_right ()

  Moves the current insert position right one character.
- int move_up ()

  Moves the current insert position up one line.
- void **next_word** (void)

  Moves the current insert position right one word.
- void overstrike (const char *text)

  Replaces text at the current insert position.
- int position_style (int lineStartPos, int lineLen, int lineIndex) const

  Find the correct style for a character.
- int position_to_xy (int pos, int *x, int *y) const

  Convert a character index into a pixel position.
- void **previous_word** (void)

  Moves the current insert position left one word.
- virtual void recalc_display ()

  Recalculate the display's visible lines and scrollbar sizes.
- void redisplay_range (int start, int end)

  Marks text from start to end as needing a redraw.
- void resize (int X, int Y, int W, int H) override

  Change the size of the displayed text area.
- int rewind_lines (int startPos, int nLines)

  Skip a number of lines back.
- void scroll (int topLineNum, int horizOffset)

  Scrolls the current buffer to start at the specified line and column.
- int **scroll_col** () const
- int **scroll_row** () const
- Fl_Align scrollbar_align () const

  Gets the scrollbar alignment type.
- void scrollbar_align (Fl_Align a)

  Sets the scrollbar alignment type.
- int scrollbar_size () const

  Gets the current size of the scrollbars' troughs, in pixels.
- void scrollbar_size (int newSize)

  Sets the pixel size of the scrollbars' troughs to `newSize`, in pixels.
- int scrollbar_width () const

  Returns the global value Fl::scrollbar_size() unless a specific scrollbar_width_ has been set.
- void scrollbar_width (int width)

Sets the global Fl::scrollbar_size(), and forces this instance of the widget to use it.

- Fl_Color secondary_selection_color () const

  Gets the background color for the secondary selection block.
- void secondary_selection_color (Fl_Color color)

  Sets the background color for the secondary selection block.
- int shortcut () const
- void shortcut (int s)
- void show_cursor (int b=1)

  Shows the text cursor.
- void show_insert_position ()

  Scrolls the text buffer to show the current insert position.
- int skip_lines (int startPos, int nLines, bool startPosIsLineStart)

  Skip a number of lines forward.
- Fl_Color spelling_underline_color () const

  Gets the underline color for style attribute ATTR_SPELLING.
- void spelling_underline_color (Fl_Color color)

  Sets the underline color for style attribute ATTR_SPELLING.
- Fl_Text_Buffer * style_buffer () const

  Gets the current style buffer associated with the text widget.
- Fl_Color textcolor () const

  Gets the default color of text in the widget.
- void textcolor (Fl_Color n)

  Sets the default color of text in the widget.
- Fl_Font textfont () const

  Gets the default font used when drawing text in the widget.
- void textfont (Fl_Font s)

  Sets the default font used when drawing text in the widget.
- Fl_Fontsize textsize () const

  Gets the default size of text in the widget.
- void textsize (Fl_Fontsize s)

  Sets the default size of text in the widget.
- int word_end (int pos) const

  Moves the insert position to the end of the current word.
- int word_start (int pos) const

  Moves the insert position to the beginning of the current word.
- void wrap_mode (int wrap, int wrap_margin)

  Set the new text wrap mode.
- int wrapped_column (int row, int column) const

  Nobody knows what this function does.
- int wrapped_row (int row) const

  Nobody knows what this function does.
- double x_to_col (double x) const

  Convert an x pixel position into a column number.
- ~Fl_Text_Display ()

  Free a text display and release its associated memory.

Public Member Functions inherited from Fl_Group

- Fl_Widget *& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget *o)

  See void Fl_Group::add(Fl_Widget &w)
- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget *const * array () const

  Returns a pointer to the internal array of children.
- Fl_Group const * as_group () const override
- Fl_Group * as_group () override

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const

  Returns the n'th child.
- int **children** () const

  Returns how many child widgets the group has.
- void clear ()

  Deletes all child widgets from memory recursively.
- unsigned int clip_children () const

  Returns the current clipping mode.
- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)

  Removes the widget at `index` from the group and deletes it.
- void end ()

  Exactly the same as current(this->parent()).
- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*
- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.
- void **forms_end** ()

  This is for forms compatibility only.
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

  This does insert(w, find(before)).
- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.
- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.

- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.

- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

Returns whether the inactive image is managed by the widget.

- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.

- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

Returns if the widget is able to take events.

- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Protected Types

- enum {
  DRAW_LINE , **FIND_INDEX** , **FIND_INDEX_FROM_ZERO** , **GET_WIDTH** ,
  FIND_CURSOR_INDEX }

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions

- void absolute_top_line_number (int oldFirstChar)

  Re-calculate absolute top line number for a change in scroll position.
- void calc_last_char ()

  Update last display character index.
- void calc_line_starts (int startLine, int endLine)

  Update the line starts array.
- void clear_rect (int style, int x, int y, int width, int height) const

  Clear a rectangle with the appropriate background color for `style`.
- void display_insert ()

  Scroll the display to bring insertion cursor into view.
- void draw () override

  Draw the widget.
- void draw_cursor (int, int)

  Draw a cursor with top center at `X`, `Y`.
- void draw_line_numbers (bool clearAll)

  Refresh the line number area.
- void draw_range (int start, int end)

  Draw a range of text.
- void draw_string (int style, int x, int y, int toX, const char *string, int nChars) const

  Draw a text segment in a single style.
- void draw_text (int X, int Y, int W, int H)

  Refresh a rectangle of the text display.
- void draw_vline (int visLineNum, int leftClip, int rightClip, int leftCharIndex, int rightCharIndex)

  Draw a single line of text.
- int empty_vlines () const

  Return true if there are lines visible with no corresponding buffer text.
- void extend_range_for_styles (int *start, int *end)

  I don't know what this does!

- void find_line_end (int pos, bool start_pos_is_line_start, int *lineEnd, int *nextLineStart) const

  Finds both the end of the current line and the start of the next line.
- void find_wrap_range (const char *deletedText, int pos, int nInserted, int nDeleted, int *modRangeStart, int *modRangeEnd, int *linesInserted, int *linesDeleted)

  Wrapping calculations.
- int find_x (const char *s, int len, int style, int x) const

  Find the index of the character that lies at the given x position / closest cursor position.
- int handle_rmb (int readonly)

  Handle right mouse button down events.
- int handle_vline (int mode, int lineStart, int lineLen, int leftChar, int rightChar, int topClip, int bottomClip, int leftClip, int rightClip) const

  Universal pixel machine.
- int longest_vline () const

  Find the longest line of all visible lines.
- void maintain_absolute_top_line_number (int state)

  Line numbering stuff, currently unused.
- int maintaining_absolute_top_line_number () const

  Returns true if a separate absolute top line number is being maintained.
- void measure_deleted_lines (int pos, int nDeleted)

  Wrapping calculations.
- double measure_proportional_character (const char *s, int colNum, int pos) const

  Wrapping calculations.
- int measure_vline (int visLineNum) const

  Returns the width in pixels of the displayed line pointed to by "visLineNum".
- void offset_line_starts (int newTopLineNum)

  Offset line start counters for a new vertical scroll position.
- int position_to_line (int pos, int *lineNum) const

  Convert a position index into a line number offset.
- int position_to_linecol (int pos, int *lineNum, int *column) const

  Find the line and column number of position `pos`.
- void reset_absolute_top_line_number ()

  Reestablish the absolute (non-wrapped) top line number.
- int scroll_ (int topLineNum, int horizOffset)

  Scrolls the current buffer to start at the specified line and column.
- double string_width (const char *string, int length, int style) const

  Find the width of a string in the font of a particular style.
- void update_h_scrollbar ()

  Update horizontal scrollbar.
- void update_line_starts (int pos, int charsInserted, int charsDeleted, int linesInserted, int linesDeleted, int *scrolled)

  Update line start arrays and variables.
- void update_v_scrollbar ()

  Update vertical scrollbar.
- int vline_length (int visLineNum) const

  Count number of bytes in a visible line.
- int wrap_uses_character (int lineEndPos) const

  Check if the line break is caused by a newline or by line wrapping.
- void wrapped_line_counter (Fl_Text_Buffer *buf, int startPos, int maxPos, int maxLines, bool startPosIs↩LineStart, int styleBufOffset, int *retPos, int *retLines, int *retLineStart, int *retLineEnd, bool countLast↩LineMissingNewLine=true) const

  Wrapping calculations.

- int xy_to_position (int x, int y, int PosType=CHARACTER_POS) const

  Translate a pixel position into a character index.
- void xy_to_rowcol (int x, int y, int ∗row, int ∗column, int PosType=CHARACTER_POS) const

  Translate pixel coordinates into row and column.

Protected Member Functions inherited from Fl_Group

- Fl_Rect ∗ bounds ()

  Returns the internal array of widget sizes and positions.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget ∗, int)

  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)

  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)

  Allow derived groups to act when a child widget is removed from the group.
- int ∗ sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

Gets the widget flags mask.

- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Static Protected Member Functions

- static void buffer_modified_cb (int pos, int nInserted, int nDeleted, int nRestyled, const char *deletedText, void *cbArg)

  This is called whenever the buffer is modified.
- static void buffer_predelete_cb (int pos, int nDeleted, void *cbArg)

  This is called before any characters are deleted.
- static void **h_scrollbar_cb** (Fl_Scrollbar *w, Fl_Text_Display *d)

  Callback for drag or valueChanged on horizontal scrollbar.
- static void scroll_timer_cb (void *)

  Timer callback for scroll events.
- static void **v_scrollbar_cb** (Fl_Scrollbar *w, Fl_Text_Display *d)

  Callback for drag or valueChanged on vertical scrollbar.

Protected Attributes

- int **damage_range1_end**
- int **damage_range1_start**
- int **damage_range2_end**
- int **damage_range2_start**
- int **display_insert_position_hint**
- bool **display_needs_recalc_**
- int **dragging**
- int **dragPos**
- int **dragType**
- Fl_Color **grammar_underline_color_**
- Fl_Align **linenumber_align_**
- Fl_Color **linenumber_bgcolor_**
- Fl_Color **linenumber_fgcolor_**
- Fl_Font **linenumber_font_**
- const char * **linenumber_format_**
- Fl_Fontsize **linenumber_size_**
- int **mAbsTopLineNum**
- Fl_Text_Buffer * **mBuffer**
- double **mColumnScale**
- int **mContinuousWrap**
- Fl_Color **mCursor_color**
- int **mCursorOldY**
- int **mCursorOn**
- int **mCursorPos**
- int **mCursorPreferredXPos**
- int **mCursorStyle**

- int **mCursorToHint**
- int **mFirstChar**
- void ∗ **mHighlightCBArg**
- int **mHorizOffset**
- int **mHorizOffsetHint**
- Fl_Scrollbar ∗ **mHScrollBar**
- int **mLastChar**
- int **mLineNumLeft**
- int **mLineNumWidth**
- int ∗ **mLineStarts**
- int **mMaxsize**
- int **mModifyingTabDistance**
- int **mNBufferLines**
- int **mNeedAbsTopLineNum**
- int **mNLinesDeleted**
- int **mNStyles**
- int **mNVisibleLines**
- Fl_Text_Buffer ∗ **mStyleBuffer**
- const Style_Table_Entry ∗ **mStyleTable**
- int **mSuppressResync**
- int **mTopLineNum**
- int **mTopLineNumHint**
- Unfinished_Style_Cb **mUnfinishedHighlightCB**
- char **mUnfinishedStyle**
- Fl_Scrollbar ∗ **mVScrollBar**
- int **mWrapMarginPix**
- Fl_Align **scrollbar_align_**
- int **scrollbar_width_**
- Fl_Color **secondary_selection_color_**
- int **shortcut_**
- Fl_Color **spelling_underline_color_**
- struct {
  int **h**
  int **w**
  int **x**
  int **y**
  } **text_area**

- Fl_Color **textcolor_**
- Fl_Font **textfont_**
- Fl_Fontsize **textsize_**

Friends

- void **fl_text_drag_me** (int pos, Fl_Text_Display ∗d)
- int **fl_text_drag_prepare** (int pos, int key, Fl_Text_Display ∗d)

Additional Inherited Members

Static Public Member Functions inherited from Fl_Group

- static Fl_Group ∗ current ()

  Returns the currently active group.
- static void current (Fl_Group ∗g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

13.136.1 Detailed Description

Rich text display widget.
This is the FLTK text display widget. It allows the user to view multiple lines of text and supports highlighting, word wrap, mixes of font faces and colors, line numbers and scrolling. The buffer that is displayed in the widget is managed by the Fl_Text_Buffer class. A single Text Buffer can be displayed by multiple Text Displays.

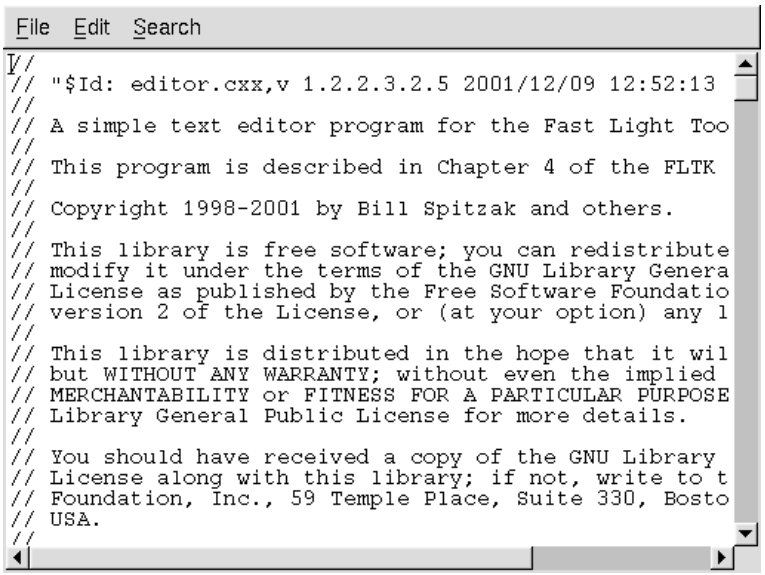


Figure 13.58 Fl_Text_Display widget

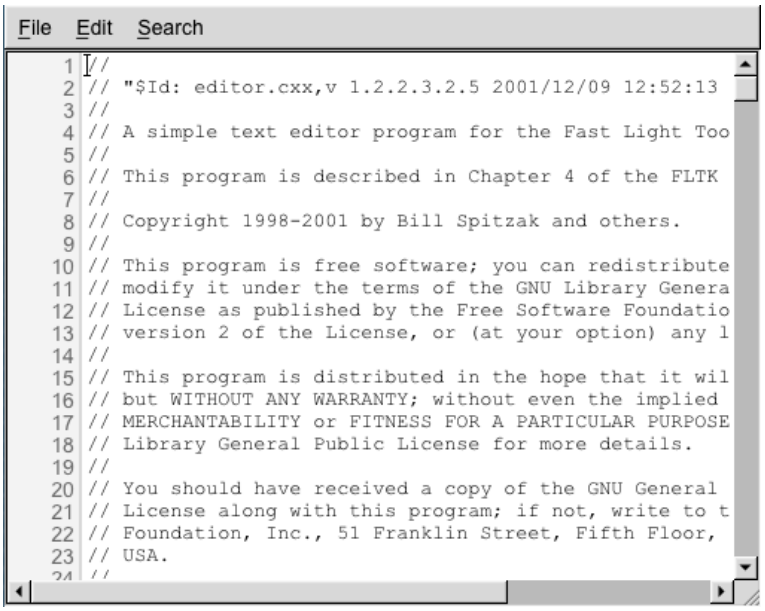


Figure 13.59 Fl_Text_Display widget with line numbers enabled

Example Use

```
#include <FL/Fl_Text_Display.H>
..
int main() {
    ..
    Fl_Text_Buffer *buff = new Fl_Text_Buffer();
    Fl_Text_Display *disp = new Fl_Text_Display(10, 10, 640, 480);
    disp->buffer(buff);                 // attach text buffer to display widget
    buff->text("line one\nline two");   // add some text to buffer
    ..
}
```

Features

- Word wrap: wrap_mode(), wrapped_column(), wrapped_row()
- Font control: textfont(), textsize(), textcolor()
- Font styling: highlight_data()
- Cursor: cursor_style(), show_cursor(), hide_cursor(), cursor_color()
- Line numbers: linenumber_width(), linenumber_font(), linenumber_size(), linenumber_fgcolor(), linenumber_bgcolor(), linenumber_align(), linenumber_format()

Note that other features may be available via Fl_Text_Editor and Fl_Text_Buffer classes.

Note

Line numbers were added in FLTK 1.3.3.

See also

Fl_Widget::shortcut_label(int)

13.136.2 Member Enumeration Documentation

13.136.2.1 anonymous enum

`anonymous enum`

text display cursor shapes enumeration

Enumerator

NORMAL_CURSOR	I-beam.

Enumerator

CARET_CURSOR	caret under the text
DIM_CURSOR	dim I-beam
BLOCK_CURSOR	unfilled box under the current character
HEAVY_CURSOR	thick I-beam
SIMPLE_CURSOR	as cursor as Fl_Input cursor

13.136.2.2 anonymous enum

`anonymous enum`

wrap types - used in wrap_mode()

Enumerator

WRAP_NONE	don't wrap text at all
WRAP_AT_COLUMN	wrap text at the given text column
WRAP_AT_PIXEL	wrap text at a pixel position
WRAP_AT_BOUNDS	wrap text so that it fits into the widget width

13.136.2.3 anonymous enum

`anonymous enum`

attribute flags in `Style_Table_Entry.attr`

Enumerator

ATTR_BGCOLOR	use the background color in the `bgcolor` field
ATTR_BGCOLOR_EXT_	(internal use)
ATTR_BGCOLOR_EXT	extend background color to the end of the line
ATTR_UNDERLINE	a single underline, underline types are mutually exclusive
ATTR_GRAMMAR	grammar suggestion (blue dotted underline)
ATTR_SPELLING	spelling suggestion (red dotted underline)
ATTR_STRIKE_THROUGH	line through the middle of the text
ATTR_LINES_MASK	the mask for all underline and strike through types

13.136.3 Constructor & Destructor Documentation

13.136.3.1 Fl_Text_Display()

```
Fl_Text_Display::Fl_Text_Display (
            int X,
            int Y,
            int W,
            int H,
            const char * l = 0 )
```

Creates a new text display widget.

Parameters

X,Y,W,H	position and size of widget
l	label text, defaults to none

13.136.3.2 ~Fl_Text_Display()

```
Fl_Text_Display::~Fl_Text_Display ( )
```

Free a text display and release its associated memory.

Note

The text buffer that the text display displays is a separate entity and is not freed, nor are the style buffer or style table.

See also

Fl_Text_Display::buffer(Fl_Text_Buffer* buf)

13.136.4 Member Function Documentation

13.136.4.1 absolute_top_line_number()

```
void Fl_Text_Display::absolute_top_line_number (
            int oldFirstChar )  [protected]
```

Re-calculate absolute top line number for a change in scroll position.
Does nothing if the absolute top line number is not being maintained.

13.136.4.2 buffer() [1/3]

```
Fl_Text_Buffer * Fl_Text_Display::buffer ( ) const  [inline]
```

Gets the current text buffer associated with the text widget.
Multiple text widgets can be associated with the same text buffer.

Returns

current text buffer

See also

Fl_Text_Display::buffer(Fl_Text_Buffer* buf)

Fl_Text_Display::buffer(Fl_Text_Buffer& buf)

13.136.4.3 buffer() [2/3]

```
void Fl_Text_Display::buffer (
            Fl_Text_Buffer & buf )  [inline]
```

Sets the current text buffer associated with the text widget.
Multiple text widgets can be associated with the same text buffer.

Parameters

buf	new text buffer

See also

Fl_Text_Display::buffer(Fl_Text_Buffer* buf)

13.136.4.4 buffer() [3/3]

```
void Fl_Text_Display::buffer (
            Fl_Text_Buffer * buf )
```

Attach a text buffer to display, replacing the current buffer (if any).
Multiple text widgets can be associated with the same text buffer.

Note

The caller is responsible for the old (replaced) buffer (if any). This method does not delete the old buffer.

Parameters

buf	attach this text buffer

13.136.4.5 buffer_modified_cb()

```
void Fl_Text_Display::buffer_modified_cb (
            int pos,
            int nInserted,
            int nDeleted,
            int nRestyled,
            const char * deletedText,
            void * cbArg )  [static], [protected]
```

This is called whenever the buffer is modified.
Callback attached to the text buffer to receive modification information.
This callback can be used to adjust the display or update other setting. It is not advisable to change any buffers or text in this callback, or line counting may get out of sync.

Parameters

pos	starting index of modification
nInserted	number of bytes we inserted (must be UTF-8 aligned!)
nDeleted	number of bytes deleted (must be UTF-8 aligned!)
nRestyled	??
deletedText	this is what was removed, must not be NULL if nDeleted is set
cbArg	"this" pointer for static callback function

13.136.4.6 buffer_predelete_cb()

```
void Fl_Text_Display::buffer_predelete_cb (
            int pos,
            int nDeleted,
            void * cbArg )  [static], [protected]
```

This is called before any characters are deleted.
Callback attached to the text buffer to receive delete information before the modifications are actually made.
This callback can be used to adjust the display or update other setting. It is not advisable to change any buffers or text in this callback, or line counting may get out of sync.

Parameters

pos	starting index of deletion
nDeleted	number of bytes we will delete (must be UTF-8 aligned!)
cbArg	"this" pointer for static callback function

13.136.4.7 calc_last_char()

```
void Fl_Text_Display::calc_last_char ( )  [protected]
```

Update last display character index.
Given a Fl_Text_Display with a complete, up-to-date lineStarts array, update the lastChar entry to point to the last

buffer position displayed.

13.136.4.8 calc_line_starts()

```
void Fl_Text_Display::calc_line_starts (
            int startLine,
            int endLine )  [protected]
```

Update the line starts array.

Scan through the text in the Text Display's buffer and recalculate the line starts array values beginning at index "startLine" and continuing through (including) "endLine". It assumes that the line starts entry preceding "startLine" (or mFirstChar if startLine is 0) is good, and re-counts newlines to fill in the requested entries. Out of range values for "startLine" and "endLine" are acceptable.

Parameters

startLine,endLine	range of lines to scan as line numbers

13.136.4.9 clear_rect()

```
void Fl_Text_Display::clear_rect (
            int style,
            int X,
            int Y,
            int width,
            int height ) const  [protected]
```

Clear a rectangle with the appropriate background color for `style`.

Parameters

style	index into style table
X,Y,width,height	size and position of background area

13.136.4.10 col_to_x()

```
double Fl_Text_Display::col_to_x (
            double col ) const
```

Convert a column number into an x pixel position.

Parameters

col	an approximate column number based on the main font

Returns

number of pixels from the left margin to the left of an average sized character

See also

x_to_col()

13.136.4.11 count_lines()

```
int Fl_Text_Display::count_lines (
            int startPos,
            int endPos,
            bool startPosIsLineStart ) const
```

Count the number of lines between two positions.
Same as Fl_Text_Buffer::count_lines(), but takes into account wrapping if wrapping is turned on. If the caller knows that `startPos` is at a line start, it can pass `startPosIsLineStart` as True to make the call more efficient by avoiding the additional step of scanning back to the last newline.

Parameters

startPos	index to first character
endPos	index after last character
startPosIsLineStart	avoid scanning back to the line start

Returns

number of lines

13.136.4.12 cursor_color() [1/2]

```
Fl_Color Fl_Text_Display::cursor_color ( ) const  [inline]
```

Gets the text cursor color.

Returns

cursor color

13.136.4.13 cursor_color() [2/2]

```
void Fl_Text_Display::cursor_color (
            Fl_Color n )  [inline]
```

Sets the text cursor color.

Parameters

n	new cursor color

13.136.4.14 cursor_style()

```
void Fl_Text_Display::cursor_style (
            int style )
```

Sets the text cursor style.
Sets the text cursor style to one of the following:

- Fl_Text_Display::NORMAL_CURSOR - Shows an I beam.
- Fl_Text_Display::CARET_CURSOR - Shows a caret under the text.
- Fl_Text_Display::DIM_CURSOR - Shows a dimmed I beam.
- Fl_Text_Display::BLOCK_CURSOR - Shows an unfilled box around the current character.
- Fl_Text_Display::HEAVY_CURSOR - Shows a thick I beam.

This call also switches the cursor on and may trigger a redraw.

Parameters

style	new cursor style

13.136.4.15 display_insert()

```
void Fl_Text_Display::display_insert ( ) [protected]
```

Scroll the display to bring insertion cursor into view.
Note: it would be nice to be able to do this without counting lines twice (scroll_() counts them too) and/or to count from the most efficient starting point, but the efficiency of this routine is not as important to the overall performance of the text display.

13.136.4.16 display_needs_recalc()

```
void Fl_Text_Display::display_needs_recalc ( ) [virtual]
```

Schedule a recalc_display() to be done on next draw().
Call this from methods that might be called repeatedly, to defers potentially CPU intensive recalc_display() until it's actually needed just before draw().

13.136.4.17 draw()

```
void Fl_Text_Display::draw (
            void ) [override], [protected], [virtual]
```

Draw the widget.
This function tries to limit drawing to smaller areas if possible.
Reimplemented from Fl_Group.

13.136.4.18 draw_cursor()

```
void Fl_Text_Display::draw_cursor (
            int X,
            int Y ) [protected]
```

Draw a cursor with top center at `X`, `Y`.

Parameters

X,Y	cursor position in pixels

13.136.4.19 draw_line_numbers()

```
void Fl_Text_Display::draw_line_numbers (
            bool clearAll ) [protected]
```

Refresh the line number area.

Parameters

clearAll	– (currently unused) If False, only draws the line number text, does not clear the area behind it. If True, clears the area and redraws the text. Use False to avoid a 'flash' for single buffered windows.

13.136.4.20 draw_range()

```
void Fl_Text_Display::draw_range (
            int startpos,
            int endpos ) [protected]
```

Draw a range of text.
Refresh all of the text between buffer positions `startpos` and `endpos` not including the character at the position `endpos`.
If `endpos` points beyond the end of the buffer, refresh the whole display after `startpos`, including blank lines which are not technically part of any range of characters.

Parameters

startpos	index of first character to draw
endpos	index after last character to draw

13.136.4.21 draw_string()

```
void Fl_Text_Display::draw_string (
            int style,
            int X,
            int Y,
            int toX,
            const char * string,
            int nChars ) const  [protected]
```

Draw a text segment in a single style.

Draw a string or blank area according to parameter `style`, using the appropriate colors and drawing method for that style, with top left corner at `X`, `Y`. If style says to draw text, use `string` as source of characters, and draw `nChars`, if style is FILL, erase rectangle where text would have drawn from `X` to `toX` and from `Y` to the maximum y extent of the current font(s).

Parameters

style	index into style lookup table
X,Y	drawing origin
toX	rightmost position if this is a fill operation
string	text if this is a drawing operation
nChars	number of characters to draw

13.136.4.22 draw_text()

```
void Fl_Text_Display::draw_text (
            int left,
            int top,
            int width,
            int height )  [protected]
```

Refresh a rectangle of the text display.

Parameters

left,top	are in coordinates of the text drawing window.
width,height	size in pixels

13.136.4.23 draw_vline()

```
void Fl_Text_Display::draw_vline (
            int visLineNum,
            int leftClip,
            int rightClip,
            int leftCharIndex,
            int rightCharIndex )  [protected]
```

Draw a single line of text.

Draw the text on a single line represented by `visLineNum` (the number of lines down from the top of the display), limited by `leftClip` and `rightClip` window coordinates and `leftCharIndex` and `rightCharIndex` character positions (not including the character at position `rightCharIndex`).

Parameters

visLineNum	index of line in the visible line number lookup
leftClip,rightClip	pixel position of clipped area
leftCharIndex,rightCharIndex	index into line of segment that we want to draw

13.136.4.24 empty_vlines()

```
int Fl_Text_Display::empty_vlines ( ) const  [protected]
```

Return true if there are lines visible with no corresponding buffer text.

Returns

1 if there are empty lines

13.136.4.25 extend_range_for_styles()

```
void Fl_Text_Display::extend_range_for_styles (
            int * startpos,
            int * endpos )  [protected]
```

I don't know what this does!
Extend the range of a redraw request (from ∗start to ∗end) with additional redraw requests resulting from changes to the attached style buffer (which contains auxiliary information for coloring or styling text).

Parameters

startpos	??
endpos	??

Todo Unicode?

13.136.4.26 find_line_end()

```
void Fl_Text_Display::find_line_end (
            int startPos,
            bool startPosIsLineStart,
            int * lineEnd,
            int * nextLineStart ) const  [protected]
```

Finds both the end of the current line and the start of the next line.
Why? In continuous wrap mode, if you need to know both, figuring out one from the other can be expensive or error prone. The problem comes when there's a trailing space or tab just before the end of the buffer. To translate an end of line value to or from the next lines start value, you need to know whether the trailing space or tab is being used as a line break or just a normal character, and to find that out would otherwise require counting all the way back to the beginning of the line.

Parameters

	startPos	
	startPosIsLineStart	
out	*lineEnd*	
out	*nextLineStart*	

13.136.4.27 find_wrap_range()

```
void Fl_Text_Display::find_wrap_range (
            const char * deletedText,
            int pos,
            int nInserted,
            int nDeleted,
            int * modRangeStart,
            int * modRangeEnd,
            int * linesInserted,
            int * linesDeleted )  [protected]
```

Wrapping calculations.

When continuous wrap is on, and the user inserts or deletes characters, wrapping can happen before and beyond the changed position. This routine finds the extent of the changes, and counts the deleted and inserted lines over that range. It also attempts to minimize the size of the range to what has to be counted and re-displayed, so the results can be useful both for delimiting where the line starts need to be recalculated, and for deciding what part of the text to redisplay.

Parameters

deletedText	
pos	
nInserted	
nDeleted	
modRangeStart	
modRangeEnd	
linesInserted	
linesDeleted	

13.136.4.28 find_x()

```
int Fl_Text_Display::find_x (
            const char * s,
            int len,
            int style,
            int x ) const  [protected]
```

Find the index of the character that lies at the given x position / closest cursor position.

Parameters

s	UTF-8 text string
len	length of string
style	index into style lookup table
x	position in pixels - negative returns closest cursor position

Returns

index into buffer

13.136.4.29 get_absolute_top_line_number()

```
int Fl_Text_Display::get_absolute_top_line_number ( ) const
```

Returns the absolute (non-wrapped) line number of the first line displayed.

Returns 0 if the absolute top line number is not being maintained.

13.136.4.30 grammar_underline_color() [1/2]

```
Fl_Color Fl_Text_Display::grammar_underline_color ( ) const  [inline]
```

Gets the underline color for style attribute ATTR_GRAMMAR.

Returns

underline color

13.136.4.31 grammar_underline_color() [2/2]

```
void Fl_Text_Display::grammar_underline_color (
            Fl_Color color )  [inline]
```

Sets the underline color for style attribute ATTR_GRAMMAR.

Parameters

color	underline color

13.136.4.32 handle()

```
int Fl_Text_Display::handle (
            int e )  [override], [virtual]
```

Event handling.
Reimplemented from Fl_Group.
Reimplemented in Fl_Text_Editor.

13.136.4.33 handle_rmb()

```
int Fl_Text_Display::handle_rmb (
            int readonly )  [protected]
```

Handle right mouse button down events.

Returns

0 for no op, 1 to cut, 2 to copy, 3 to paste

13.136.4.34 handle_vline()

```
int Fl_Text_Display::handle_vline (
            int mode,
            int lineStartPos,
            int lineLen,
            int leftChar,
            int rightChar,
            int Y,
            int bottomClip,
            int leftClip,
            int rightClip ) const  [protected]
```

Universal pixel machine.
We use a single function that handles all line layout, measuring, and drawing

- draw a text range
- return the width of a text range in pixels
- return the index of a character that is at a pixel position

Parameters

in	*mode*	DRAW_LINE, GET_WIDTH, FIND_INDEX, FIND_INDEX_FROM_ZERO, or FIND_CURSOR_INDEX
in	*lineStartPos*	index of first character
in	*lineLen*	size of string in bytes
in	*leftChar,rightChar*	
in	*Y*	drawing position
in	*bottomClip,leftClip,rightClip*	stop work when we reach the clipped area. rightClip is the X position that we search in FIND_INDEX.

Return values

DRAW_LINE	index of last drawn character
GET_WIDTH	width in pixels of text segment if we would draw it
FIND_INDEX	index of character at given x position in window coordinates
FIND_INDEX_FROM_ZERO	index of character at given x position without scrolling and widget offsets

Todo we need to handle hidden hyphens and tabs here!

we handle all styles and selections

we must provide code to get pixel positions of the middle of a character as well

13.136.4.35 highlight_data()

```
void Fl_Text_Display::highlight_data (
            Fl_Text_Buffer * styleBuffer,
            const Style_Table_Entry * styleTable,
            int nStyles,
            char unfinishedStyle,
            Unfinished_Style_Cb unfinishedHighlightCB,
            void * cbArg )
```

Attach (or remove) highlight information in text display and redisplay.
Highlighting information consists of a style buffer which parallels the normal text buffer, but codes font and color information for the display; a style table which translates style buffer codes (indexed by buffer character - 'A') into fonts and colors; and a callback mechanism for as-needed highlighting, triggered by a style buffer entry of "unfinished↩ Style". Style buffer can trigger additional redisplay during a normal buffer modification if the buffer contains a primary Fl_Text_Selection (see extend_range_for_styles() for more information on this protocol).
Style buffers, tables and their associated memory are managed by the caller.
Styles are ranged from 65 ('A') to 126.

Note

Style information in the style buffer must have the same byte offset as the corresponding character in the text buffer. UTF-8 characters can have a maximum length of four bytes. Style information must take this into account and fill the unused bytes with 0. See `fl_utf8len()`.

Text: "∗g∗ r ü ∗n∗" , where normal style is 'A', and bold is 'B'

```
Text Buffer(hex):  67 72 c3 bc 6e : gr..n
Style Buffer(hex): 42 41 41 00 42 : BAA.B
```

Parameters

styleBuffer	this buffer works in parallel to the text buffer. For every character in the text buffer, the style buffer has a byte at the same offset that contains an index into an array of possible styles.

Parameters

styleTable	a list of styles indexed by the style buffer
nStyles	number of styles in the style table
unfinishedStyle	if this style is found, the callback below is called
unfinishedHighlightCB	if a character with an unfinished style is found, this callback will be called
cbArg	an optional argument for the callback above, usually a pointer to the Text Display.

See also

Fl_Text_Display::style_buffer()

13.136.4.36 in_selection()

```
int Fl_Text_Display::in_selection (
            int X,
            int Y ) const
```

Check if a pixel position is within the primary selection.

Parameters

X,Y	pixel position to test

Returns

1 if position (X, Y) is inside of the primary Fl_Text_Selection

13.136.4.37 insert()

```
void Fl_Text_Display::insert (
            const char * text )
```

Inserts "text" at the current cursor location.
This has the same effect as inserting the text into the buffer using insert(insert_position(),text) and then moving the insert position after the newly inserted text, except that it's optimized to do less redrawing.

Parameters

text	new text in UTF-8 encoding.

13.136.4.38 insert_position() [1/2]

```
int Fl_Text_Display::insert_position ( ) const  [inline]
```

Gets the position of the text insertion cursor for text display.
The insert position is the byte count (offset) from the beginning of the text buffer (starting with 0). Returns 0 (zero) if no buffer is associated to the text display. Returns buffer()->length() if the insert position is at the end of the buffer.

Returns

insert position index into text buffer

See also

insert_position(int)

13.136.4.39 insert_position() [2/2]

```
void Fl_Text_Display::insert_position (
            int newPos )
```

Sets the position of the text insertion cursor for text display.
Moves the insertion cursor in front of the character at `newPos`. This function may trigger a redraw.

Parameters

newPos	new caret position

13.136.4.40 line_end()

```
int Fl_Text_Display::line_end (
            int startPos,
            bool startPosIsLineStart ) const
```

Returns the end of a line.
Same as buffer()->line_end(startPos), but takes into account line breaks when wrapping is turned on. If the caller knows that `startPos` is at a line start, it can pass `startPosIsLineStart` as True to make the call more efficient by avoiding the additional step of scanning back to the last newline.
Note that the definition of the end of a line is less clear when continuous wrap is on. With continuous wrap off, it's just a pointer to the newline that ends the line. When it's on, it's the character beyond the last **displayable** character on the line, where a whitespace character which has been "converted" to a newline for wrapping is not considered displayable. Also note that a line can be wrapped at a non-whitespace character if the line had no whitespace. In this case, this routine returns a pointer to the start of the next line. This is also consistent with the model used by visLineLength.

Parameters

startPos	index to starting character
startPosIsLineStart	avoid scanning back to the line start

Returns

new position as index

13.136.4.41 line_start()

```
int Fl_Text_Display::line_start (
            int pos ) const
```

Return the beginning of a line.
Same as buffer()->line_start(pos), but returns the character after last wrap point rather than the last newline.

Parameters

pos	index to starting character

Returns

new position as index

13.136.4.42 linenumber_align()

```
void Fl_Text_Display::linenumber_align (
            Fl_Align val )
```

Set alignment for line numbers (if enabled).

Valid values are FL_ALIGN_LEFT, FL_ALIGN_CENTER or FL_ALIGN_RIGHT.

Version

1.3.3

13.136.4.43 linenumber_bgcolor()

```
void Fl_Text_Display::linenumber_bgcolor (
            Fl_Color val )
```

Set the background color used for line numbers (if enabled).

Version

1.3.3

13.136.4.44 linenumber_fgcolor()

```
void Fl_Text_Display::linenumber_fgcolor (
            Fl_Color val )
```

Set the foreground color used for line numbers (if enabled).

Version

1.3.3

13.136.4.45 linenumber_font()

```
void Fl_Text_Display::linenumber_font (
            Fl_Font val )
```

Set the font used for line numbers (if enabled).

Version

1.3.3

13.136.4.46 linenumber_format()

```
void Fl_Text_Display::linenumber_format (
            const char * val )
```

Sets the printf() style format string used for line numbers.
Default is "%d" for normal unpadded decimal integers.
An internal copy of `val` is allocated and managed; it is automatically freed whenever a new value is assigned, or when the widget is destroyed.
The value of `val` must *not* be NULL.
Example values:

```
- "%d"   -- For normal line numbers without padding (Default)
- "%03d" -- For 000 padding
- "%x"   -- For hexadecimal line numbers
- "%o"   -- For octal line numbers
```

Version

1.3.3

13.136.4.47 linenumber_size()

```
void Fl_Text_Display::linenumber_size (
            Fl_Fontsize val )
```

Set the font size used for line numbers (if enabled).

Version

1.3.3

13.136.4.48 linenumber_width()

```
void Fl_Text_Display::linenumber_width (
            int width )
```

Set width of screen area for line numbers.
Use to also enable/disable line numbers. A value of 0 disables line numbering, values >0 enable the line number display.

Parameters

width	The new width of the area for line numbers to appear, in pixels. 0 disables line numbers (default)

13.136.4.49 longest_vline()

```
int Fl_Text_Display::longest_vline ( ) const  [protected]
```

Find the longest line of all visible lines.

Returns

the width of the longest visible line in pixels

13.136.4.50 maintain_absolute_top_line_number()

```
void Fl_Text_Display::maintain_absolute_top_line_number (
            int state )  [protected]
```

Line numbering stuff, currently unused.
In continuous wrap mode, internal line numbers are calculated after wrapping. A separate non-wrapped line count is maintained when line numbering is turned on. There is some performance cost to maintaining this line count, so normally absolute line numbers are not tracked if line numbering is off. This routine allows callers to specify that they still want this line count maintained (for use via Fl_Text_Display::position_to_linecol()). More specifically, this allows the line number reported in the statistics line to be calibrated in absolute lines, rather than post-wrapped lines.

13.136.4.51 maintaining_absolute_top_line_number()

```
int Fl_Text_Display::maintaining_absolute_top_line_number ( ) const  [protected]
```

Returns true if a separate absolute top line number is being maintained.
The absolute top line number is used for displaying line numbers in continuous wrap mode or showing in the statistics line (the latter is currently not available in FLTK).

13.136.4.52 measure_deleted_lines()

```
void Fl_Text_Display::measure_deleted_lines (
            int pos,
            int nDeleted )  [protected]
```

Wrapping calculations.
This is a stripped-down version of the findWrapRange() function above, intended to be used to calculate the number of "deleted" lines during a buffer modification. It is called *before* the modification takes place.
This function should only be called in continuous wrap mode with a non-fixed font width. In that case, it is impossible to calculate the number of deleted lines, because the necessary style information is no longer available *after* the modification. In other cases, we can still perform the calculation afterwards (possibly even more efficiently).

Parameters

pos	
nDeleted	

13.136.4.53 measure_proportional_character()

```
double Fl_Text_Display::measure_proportional_character (
            const char * s,
            int xPix,
            int pos ) const  [protected]
```

Wrapping calculations.

Measure the width in pixels of the first character of string "s" at a particular column "colNum" and buffer position "pos". This is for measuring characters in proportional or mixed-width highlighting fonts.

A note about proportional and mixed-width fonts: the mixed width and proportional font code in nedit does not get much use in general editing, because nedit doesn't allow per-language-mode fonts, and editing programs in a proportional font is usually a bad idea, so very few users would choose a proportional font as a default. There are still probably mixed- width syntax highlighting cases where things don't redraw properly for insertion/deletion, though static display and wrapping and resizing should now be solid because they are now used for online help display.

Parameters

s	text string
xPix	x pixel position needed for calculating tab widths
pos	offset within string

Returns

width of character in pixels

13.136.4.54 measure_vline()

```
int Fl_Text_Display::measure_vline (
            int visLineNum ) const  [protected]
```

Returns the width in pixels of the displayed line pointed to by "visLineNum".

Parameters

visLineNum	index into visible lines array

Returns

width of line in pixels

13.136.4.55 move_down()

```
int Fl_Text_Display::move_down ( )
```

Moves the current insert position down one line.

Returns

1 if the cursor moved, 0 if the beginning of the text was reached

13.136.4.56 move_left()

```
int Fl_Text_Display::move_left ( )
```

Moves the current insert position left one character.

Returns

1 if the cursor moved, 0 if the beginning of the text was reached

13.136.4.57 move_right()

```
int Fl_Text_Display::move_right ( )
```

Moves the current insert position right one character.

Returns

1 if the cursor moved, 0 if the end of the text was reached

13.136.4.58 move_up()

```
int Fl_Text_Display::move_up ( )
```

Moves the current insert position up one line.

Returns

1 if the cursor moved, 0 if the beginning of the text was reached

13.136.4.59 offset_line_starts()

```
void Fl_Text_Display::offset_line_starts (
            int newTopLineNum )  [protected]
```

Offset line start counters for a new vertical scroll position.
Offset the line starts array, mTopLineNum, mFirstChar and lastChar, for a new vertical scroll position given by new↩ TopLineNum. If any currently displayed lines will still be visible, salvage the line starts values, otherwise, count lines from the nearest known line start (start or end of buffer, or the closest value in the mLineStarts array)

Parameters

newTopLineNum	index into buffer

13.136.4.60 overstrike()

```
void Fl_Text_Display::overstrike (
            const char * text )
```

Replaces text at the current insert position.

Parameters

text	new text in UTF-8 encoding

Todo Unicode? Find out exactly what we do here and simplify.

13.136.4.61 position_style()

```
int Fl_Text_Display::position_style (
            int lineStartPos,
            int lineLen,
            int lineIndex ) const
```

Find the correct style for a character.
Determine the drawing method to use to draw a specific character from "buf".
`lineStartPos` gives the character index where the line begins, `lineIndex`, the number of characters past the beginning of the line, and `lineLen` the number of displayed characters past the beginning of the line. Passing `lineStartPos` of -1 returns the drawing style for "no text".
Why not just: position_style(pos)? Because style applies to blank areas of the window beyond the text boundaries, and because this routine must also decide whether a position is inside of a rectangular Fl_Text_Selection, and do so efficiently, without re-counting character positions from the start of the line.

Note that style is a somewhat incorrect name, drawing method would be more appropriate.
If lineIndex is pointing to the last character in a line, and the second to last character has the ATTR_BGCOLOR_EXT set, the background color will extend into the remaining line.

Parameters

lineStartPos	beginning of this line
lineLen	number of bytes in line
lineIndex	position of character within line

Returns

style for the given character

13.136.4.62 position_to_line()

```
int Fl_Text_Display::position_to_line (
            int pos,
            int * lineNum ) const  [protected]
```

Convert a position index into a line number offset.
Find the line number of position `pos` relative to the first line of displayed text, counting from 0 to *visible lines - 1*.
The line number is returned in `lineNum`.
Returns 0 if the line is not displayed. In this case `lineNum` is 0 as well.
Returns 1 if the line is displayed. In this case `lineNum` is the relative line number.

Parameters

in	*pos*	byte position in buffer
out	*lineNum*	relative line number of byte `pos` in buffer

Returns

whether the character at byte position `pos` is currently displayed

Return values

0	`pos` is not displayed; `lineNum` is invalid (zero)
1	`pos` is displayed; `lineNum` is valid

13.136.4.63 position_to_linecol()

```
int Fl_Text_Display::position_to_linecol (
            int pos,
            int * lineNum,
            int * column ) const  [protected]
```

Find the line and column number of position `pos`.
This only works for displayed lines. If the line is not displayed, the function returns 0 (without the mLineStarts array it could turn in to very long calculation involving scanning large amounts of text in the buffer). If continuous wrap mode is on, returns the absolute line number (as opposed to the wrapped line number which is used for scrolling).

Parameters

	pos	character index
out	*lineNum*	absolute (unwrapped) line number

Parameters

out	*column*	character offset to the beginning of the line

Returns

0 if `pos` is off screen, line number otherwise

Todo a column number makes little sense in the UTF-8/variable font width environment. We will have to further define what exactly we want to return. Please check the functions that call this particular function.

13.136.4.64 position_to_xy()

```
int Fl_Text_Display::position_to_xy (
            int pos,
            int * X,
            int * Y ) const
```

Convert a character index into a pixel position.
Translate a buffer text position to the XY location where the top left of the cursor would be positioned to point to that character. Returns 0 if the position is not displayed because it is ***vertically*** *out* of view. If the position is horizontally out of view, returns the X coordinate where the position would be if it were visible.

Parameters

	pos	character index
out	*X,Y*	pixel position of character on screen

Returns

0 if character vertically out of view, X & Y positions otherwise

13.136.4.65 recalc_display()

```
void Fl_Text_Display::recalc_display ( ) [virtual]
```

Recalculate the display's visible lines and scrollbar sizes.
Beware calling this directly may cause a lot of CPU if called repeatedly (issue 300). Better to call display_needs_recalc() to flag a recalc to be done during next draw().

13.136.4.66 redisplay_range()

```
void Fl_Text_Display::redisplay_range (
            int startpos,
            int endpos )
```

Marks text from start to end as needing a redraw.
This function will trigger a damage event and later a redraw of parts of the widget.

Parameters

startpos	index of first character needing redraw
endpos	index after last character needing redraw

13.136.4.67 reset_absolute_top_line_number()

```
void Fl_Text_Display::reset_absolute_top_line_number ( ) [protected]
```

Reestablish the absolute (non-wrapped) top line number.

Count lines from the beginning of the buffer to reestablish the absolute (non-wrapped) top line number. If mode is not continuous wrap, or the number is not being maintained, does nothing.

13.136.4.68 resize()

```
void Fl_Text_Display::resize (
            int X,
            int Y,
            int W,
            int H )  [override], [virtual]
```

Change the size of the displayed text area.
Calling this function will trigger a recalculation of all visible lines and of all scrollbar sizes.

Parameters

X,Y,W,H	new position and size of this widget

Reimplemented from Fl_Group.

13.136.4.69 rewind_lines()

```
int Fl_Text_Display::rewind_lines (
            int startPos,
            int nLines )
```

Skip a number of lines back.
Same as buffer()->rewind_lines(startPos, nLines), but takes into account line breaks when wrapping is turned on.

Parameters

startPos	index to starting character
nLines	number of lines to skip back

Returns

new position as index

13.136.4.70 scroll()

```
void Fl_Text_Display::scroll (
            int topLineNum,
            int horizOffset )
```

Scrolls the current buffer to start at the specified line and column.

Parameters

topLineNum	top line number
horizOffset	column number

Todo Column numbers make little sense here.

13.136.4.71 scroll_()

```
int Fl_Text_Display::scroll_ (
            int topLineNum,
            int horizOffset )  [protected]
```

Scrolls the current buffer to start at the specified line and column.

Parameters

topLineNum	top line number
horizOffset	in pixels

Returns

0 if nothing changed, 1 if we scrolled

13.136.4.72 scroll_timer_cb()

```
void Fl_Text_Display::scroll_timer_cb (
            void * user_data )  [static], [protected]
```

Timer callback for scroll events.
This timer event scrolls the text view proportionally to how far the mouse pointer has left the text area. This allows for smooth scrolling without "wiggeling" the mouse.

13.136.4.73 scrollbar_align() [1/2]

```
Fl_Align Fl_Text_Display::scrollbar_align ( ) const  [inline]
```

Gets the scrollbar alignment type.

Returns

scrollbar alignment

13.136.4.74 scrollbar_align() [2/2]

```
void Fl_Text_Display::scrollbar_align (
            Fl_Align a )  [inline]
```

Sets the scrollbar alignment type.

Parameters

a	new scrollbar alignment

13.136.4.75 scrollbar_size() [1/2]

```
int Fl_Text_Display::scrollbar_size ( ) const  [inline]
```

Gets the current size of the scrollbars' troughs, in pixels.
If this value is zero (default), this widget will use the Fl::scrollbar_size() value as the scrollbar's width.

Returns

Scrollbar size in pixels, or 0 if the global Fl::scrollbar_size() is being used.

See also

Fl::scrollbar_size(int)

13.136.4.76 scrollbar_size() [2/2]

```
void Fl_Text_Display::scrollbar_size (
            int newSize )  [inline]
```

Sets the pixel size of the scrollbars' troughs to `newSize`, in pixels.
Normally you should not need this method, and should use Fl::scrollbar_size(int) instead to manage the size of ALL your widgets' scrollbars. This ensures your application has a consistent UI, is the default behavior, and is normally what you want.

Only use THIS method if you really need to override the global scrollbar size. The need for this should be rare. Setting `newSize` to the special value of 0 causes the widget to track the global Fl::scrollbar_size(), which is the default.

Parameters

in	*newSize*	Sets the scrollbar size in pixels. If 0 (default), scrollbar size tracks the global Fl::scrollbar_size()

See also

Fl::scrollbar_size()

13.136.4.77 scrollbar_width() [1/2]

```
int Fl_Text_Display::scrollbar_width ( ) const  [inline]
```

Returns the global value Fl::scrollbar_size() unless a specific scrollbar_width_ has been set.

Deprecated Use scrollbar_size() instead.

Todo This method should eventually be removed.

Deprecated "since 1.4.0 - use scrollbar_size() instead"

13.136.4.78 scrollbar_width() [2/2]

```
void Fl_Text_Display::scrollbar_width (
            int width )  [inline]
```

Sets the global Fl::scrollbar_size(), and forces this instance of the widget to use it.

Deprecated Use scrollbar_size() instead.

Todo This method should eventually be removed

Deprecated "since 1.4.0 - use scrollbar_size() instead"

13.136.4.79 secondary_selection_color() [1/2]

```
Fl_Color Fl_Text_Display::secondary_selection_color ( ) const  [inline]
```

Gets the background color for the secondary selection block.

Returns

background color color

13.136.4.80 secondary_selection_color() [2/2]

```
void Fl_Text_Display::secondary_selection_color (
            Fl_Color color )  [inline]
```

Sets the background color for the secondary selection block.

Parameters

color	background color

13.136.4.81 shortcut() [1/2]

```
int Fl_Text_Display::shortcut ( ) const  [inline]
```

Todo FIXME : get set methods pointing on shortcut_ have no effects as shortcut_ is unused in this class and derived!

Returns

the current shortcut key

13.136.4.82 shortcut() [2/2]

```
void Fl_Text_Display::shortcut (
            int s )  [inline]
```

Todo FIXME : get set methods pointing on shortcut_ have no effects as shortcut_ is unused in this class and derived!

Parameters

s	the new shortcut key

13.136.4.83 show_cursor()

```
void Fl_Text_Display::show_cursor (
            int b = 1 )
```

Shows the text cursor.
This function may trigger a redraw.

Parameters

b	show(1) or hide(0) the text cursor (caret).

13.136.4.84 show_insert_position()

```
void Fl_Text_Display::show_insert_position ( )
```

Scrolls the text buffer to show the current insert position.
This function triggers a complete recalculation, ending in a call to Fl_Text_Display::display_insert()

13.136.4.85 skip_lines()

```
int Fl_Text_Display::skip_lines (
            int startPos,
            int nLines,
            bool startPosIsLineStart )
```

Skip a number of lines forward.
Same as Fl_Text_Buffer::skip_lines(startPos, nLines), but takes into account line breaks when wrapping is turned on. If the caller knows that `startPos` is at a line start, it can pass `startPosIsLineStart` as True to make the call more efficient by avoiding the additional step of scanning back to the last newline.

Parameters

startPos	index to starting character
nLines	number of lines to skip ahead
startPosIsLineStart	avoid scanning back to the line start

Returns

new position as index

13.136.4.86 spelling_underline_color() [1/2]

```
Fl_Color Fl_Text_Display::spelling_underline_color ( ) const  [inline]
```

Gets the underline color for style attribute ATTR_SPELLING.

Returns

underline color

13.136.4.87 spelling_underline_color() [2/2]

```
void Fl_Text_Display::spelling_underline_color (
            Fl_Color color )  [inline]
```

Sets the underline color for style attribute ATTR_SPELLING.

Parameters

color	underline color

13.136.4.88 string_width()

```
double Fl_Text_Display::string_width (
            const char * string,
            int length,
            int style ) const  [protected]
```

Find the width of a string in the font of a particular style.

Parameters

string	the text
length	number of bytes in string
style	index into style table

Returns

width of text segment in pixels

13.136.4.89 style_buffer()

```
Fl_Text_Buffer * Fl_Text_Display::style_buffer ( ) const  [inline]
```

Gets the current style buffer associated with the text widget.
Multiple text widgets can be associated with the same style buffer.

Returns

current style buffer

See also

Fl_Text_Display::highlight_data()

13.136.4.90 textcolor() [1/2]

```
Fl_Color Fl_Text_Display::textcolor ( ) const  [inline]
```

Gets the default color of text in the widget.

Returns

text color unless overridden by a style

13.136.4.91 textcolor() [2/2]

```
void Fl_Text_Display::textcolor (
            Fl_Color n )  [inline]
```

Sets the default color of text in the widget.

Parameters

n	new text color

13.136.4.92 textfont() [1/2]

```
Fl_Font Fl_Text_Display::textfont ( ) const  [inline]
```

Gets the default font used when drawing text in the widget.

Returns

current text font face unless overridden by a style

13.136.4.93 textfont() [2/2]

```
void Fl_Text_Display::textfont (
            Fl_Font s )  [inline]
```

Sets the default font used when drawing text in the widget.

Parameters

s	default text font face

13.136.4.94 textsize() [1/2]

```
Fl_Fontsize Fl_Text_Display::textsize ( ) const  [inline]
```

Gets the default size of text in the widget.

Returns

current text height unless overridden by a style

13.136.4.95 textsize() [2/2]

```
void Fl_Text_Display::textsize (
            Fl_Fontsize s )  [inline]
```

Sets the default size of text in the widget.

Parameters

s	new text size

13.136.4.96 update_h_scrollbar()

```
void Fl_Text_Display::update_h_scrollbar ( ) [protected]
```

Update horizontal scrollbar.
Update the minimum, maximum, slider size, page increment, and value for the horizontal scrollbar.

13.136.4.97 update_line_starts()

```
void Fl_Text_Display::update_line_starts (
            int pos,
            int charsInserted,
            int charsDeleted,
            int linesInserted,
            int linesDeleted,
            int * scrolled ) [protected]
```

Update line start arrays and variables.
Update the line starts array, mTopLineNum, mFirstChar and lastChar for this text display after a modification to the text buffer, given by the position `pos` where the change began, and the numbers of characters and lines inserted and deleted.

Parameters

	pos	index into buffer of recent changes
	charsInserted	number of bytes(!) inserted
	charsDeleted	number of bytes(!) deleted
	linesInserted	number of lines
	linesDeleted	number of lines
`out`	*scrolled*	set to 1 if the text display needs to be scrolled

13.136.4.98 update_v_scrollbar()

```
void Fl_Text_Display::update_v_scrollbar ( ) [protected]
```

Update vertical scrollbar.
Update the minimum, maximum, slider size, page increment, and value for the vertical scrollbar.

13.136.4.99 vline_length()

```
int Fl_Text_Display::vline_length (
            int visLineNum ) const [protected]
```

Count number of bytes in a visible line.
Return the length of a line (number of bytes) by examining entries in the line starts array rather than by scanning for newlines.

Parameters

visLineNum	index of line in visible line array

Returns

number of bytes in this line

13.136.4.100 word_end()

```
int Fl_Text_Display::word_end (
            int pos ) const [inline]
```

Moves the insert position to the end of the current word.

Parameters

pos	start calculation at this index

Returns

index of first character after the end of the word

13.136.4.101 word_start()

```
int Fl_Text_Display::word_start (
            int pos ) const  [inline]
```

Moves the insert position to the beginning of the current word.

Parameters

pos	start calculation at this index

Returns

beginning of the words

13.136.4.102 wrap_mode()

```
void Fl_Text_Display::wrap_mode (
            int wrap,
            int wrapMargin )
```

Set the new text wrap mode.
If `wrap` mode is not zero, this call enables automatic word wrapping at column `wrapMargin`. Word-wrapping does not change the text buffer itself, only the way the text is displayed. Different Text Displays can have different wrap modes, even if they share the same Text Buffer.
Valid wrap modes are:

- WRAP_NONE : don't wrap text at all
- WRAP_AT_COLUMN : wrap text at the given text column
- WRAP_AT_PIXEL : wrap text at a pixel position
- WRAP_AT_BOUNDS : wrap text so that it fits into the widget width

Parameters

wrap	new wrap mode (see above)
wrapMargin	in WRAP_AT_COLUMN mode, text will wrap at the n'th character. For variable width fonts, an average character width is calculated. The column width is calculated using the current textfont or the first style when this function is called. If the font size changes, this function must be called again. In WRAP_AT_PIXEL mode, this is the pixel position.

13.136.4.103 wrap_uses_character()

```
int Fl_Text_Display::wrap_uses_character (
            int lineEndPos ) const  [protected]
```

Check if the line break is caused by a newline or by line wrapping.
Line breaks in continuous wrap mode usually happen at newlines (\n) or whitespace. This line-terminating character

is not included in line width measurements and has a special status as a non-visible character. However, lines with no whitespace are wrapped without the benefit of a line terminating character, and this distinction causes endless trouble with all of the text display code which was originally written without continuous wrap mode and always expects to wrap at a newline character.
Given the position of the end of the line, as returned by Fl_Text_Display::line_end() or Fl_Text_Buffer::line_end(), this returns true if there is a line terminating character, and false if there's not. On the last character in the buffer, this function can't tell for certain whether a trailing space was used as a wrap point, and just guesses that it wasn't. So if an exact accounting is necessary, don't use this function.

Parameters

lineEndPos	index of character where the line wraps

Returns

1 if a \n character causes the line wrap

13.136.4.104 wrapped_column()

```
int Fl_Text_Display::wrapped_column (
            int row,
            int column ) const
```

Nobody knows what this function does.
Correct a column number based on an unconstrained position (as returned by TextDXYToUnconstrainedPosition) to be relative to the last actual newline in the buffer before the row and column position given, rather than the last line start created by line wrapping. This is an adapter for rectangular selections and code written before continuous wrap mode, which thinks that the unconstrained column is the number of characters from the last newline. Obviously this is time consuming, because it invloves character re-counting.

Parameters

row	
column	

Returns

something unknown

Todo What does this do and how is it useful? Column numbers mean little in this context. Which functions depend on this one? Function TextDXYToUnconstrainedPosition does not exist (nedit port?)

Todo Unicode?

13.136.4.105 wrapped_line_counter()

```
void Fl_Text_Display::wrapped_line_counter (
            Fl_Text_Buffer * buf,
            int startPos,
            int maxPos,
            int maxLines,
            bool startPosIsLineStart,
            int styleBufOffset,
            int * retPos,
            int * retLines,
            int * retLineStart,
            int * retLineEnd,
            bool countLastLineMissingNewLine = true ) const  [protected]
```

Wrapping calculations.
Count forward from startPos to either maxPos or maxLines (whichever is reached first), and return all relevant positions and line count. The provided textBuffer may differ from the actual text buffer of the widget. In that case it must be a (partial) copy of the actual text buffer and the styleBufOffset argument must indicate the starting position of the copy, to take into account the correct style information.

Parameters

in	*buf*	The text buffer to operate on
in	*startPos*	Starting index position into the buffer
in	*maxPos*	Maximum index position into the buffer we'll reach
in	*maxLines*	Maximum number of lines we'll reach
in	*startPosIsLineStart*	Flag indicating if startPos is start of line. (If set, prevents our having to find the line start)
in	*styleBufOffset*	Offset index position into style buffer.
out	*retPos*	Position where counting ended. When counting lines, the position returned is the start of the line "maxLines" lines beyond "startPos".
out	*retLines*	Number of line breaks counted
out	*retLineStart*	Start of the line where counting ended
out	*retLineEnd*	End position of the last line traversed
out	*countLastLineMissingNewLine*	

13.136.4.106 wrapped_row()

```
int Fl_Text_Display::wrapped_row (
            int row ) const
```

Nobody knows what this function does.
Correct a row number from an unconstrained position (as returned by TextDXYToUnconstrainedPosition) to a straight number of newlines from the top line of the display. Because rectangular selections are based on newlines, rather than display wrapping, and anywhere a rectangular selection needs a row, it needs it in terms of un-wrapped lines.

Parameters

row	

Returns

something unknown

Todo What does this do and how is it useful? Column numbers mean little in this context. Which functions depend on this one? Function TextDXYToUnconstrainedPosition does not exist (nedit port?)

13.136.4.107 x_to_col()

```
double Fl_Text_Display::x_to_col (
            double x ) const
```

Convert an x pixel position into a column number.

Parameters

x	number of pixels from the left margin

Returns

an approximate column number based on the main font

13.136.4.108 xy_to_position()

```
int Fl_Text_Display::xy_to_position (
            int X,
            int Y,
            int posType = CHARACTER_POS ) const  [protected]
```

Translate a pixel position into a character index.
Translate window coordinates to the nearest (insert cursor or character cell) text position. The parameter `posType` specifies how to interpret the position: CURSOR_POS means translate the coordinates to the nearest cursor position, and CHARACTER_POS means return the position of the character closest to (`X`, `Y`).

Parameters

X,Y	pixel position
posType	CURSOR_POS or CHARACTER_POS

Returns

index into text buffer

13.136.4.109 xy_to_rowcol()

```
void Fl_Text_Display::xy_to_rowcol (
            int X,
            int Y,
            int * row,
            int * column,
            int posType = CHARACTER_POS ) const  [protected]
```

Translate pixel coordinates into row and column.
Translate window coordinates to the nearest row and column number for positioning the cursor. This, of course, makes no sense when the font is proportional, since there are no absolute columns. The parameter posType specifies how to interpret the position: CURSOR_POS means translate the coordinates to the nearest position between characters, and CHARACTER_POS means translate the position to the nearest character cell.

Parameters

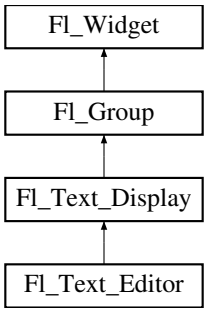
	X,Y	pixel coordinates
`out`	*row,column*	neares row and column
	posType	CURSOR_POS or CHARACTER_POS

The documentation for this class was generated from the following files:

- Fl_Text_Display.H
- Fl_Text_Display.cxx

13.137 Fl_Text_Editor Class Reference

This is the FLTK text editor widget.
`#include <Fl_Text_Editor.H>`
Inheritance diagram for Fl_Text_Editor:



Classes

- struct Key_Binding

  Simple linked list item associating a key/state to a function.

Public Types

- typedef int(∗ **Key_Func**) (int key, Fl_Text_Editor ∗editor)

  Key function binding callback type.

Public Types inherited from Fl_Text_Display

- enum {
  NORMAL_CURSOR , CARET_CURSOR , DIM_CURSOR , BLOCK_CURSOR ,
  HEAVY_CURSOR , SIMPLE_CURSOR }

  text display cursor shapes enumeration
- enum { **CURSOR_POS** , **CHARACTER_POS** }

  the character position is the left edge of a character, whereas the cursor is thought to be between the centers of two consecutive characters.
- enum {
  DRAG_NONE = -2 , **DRAG_START_DND** = -1 , **DRAG_CHAR** = 0 , **DRAG_WORD** = 1 ,
  DRAG_LINE = 2 }

  drag types - they match Fl::event_clicks() so that single clicking to start a collection selects by character, double clicking selects by word and triple clicking selects by line.
- enum { WRAP_NONE , WRAP_AT_COLUMN , WRAP_AT_PIXEL , WRAP_AT_BOUNDS }

  wrap types - used in wrap_mode()
- enum {
  ATTR_BGCOLOR = 0x0001 , ATTR_BGCOLOR_EXT_ = 0x0002 , ATTR_BGCOLOR_EXT = 0x0003 ,
  ATTR_UNDERLINE = 0x0004 ,
  ATTR_GRAMMAR = 0x0008 , ATTR_SPELLING = 0x000C , ATTR_STRIKE_THROUGH = 0x0010 ,
  ATTR_LINES_MASK = 0x001C }

  attribute flags in `Style_Table_Entry.attr`
- typedef void(∗ **Unfinished_Style_Cb**) (int, void ∗)

Public Member Functions

- void **add_default_key_bindings** (Key_Binding ∗∗list)

  Adds all of the default editor key bindings to the specified key binding list.
- void **add_key_binding** (int key, int state, Key_Func f)

  Adds a `key` of state `state` with the function `f`.
- void add_key_binding (int key, int state, Key_Func f, Key_Binding ∗∗list)

  Adds a `key` of state `state` with the function `function` to an arbitrary key binding list `list`.
- Key_Func **bound_key_function** (int key, int state) const

  Returns the function associated with a key binding.

- Key_Func **bound_key_function** (int key, int state, Key_Binding *list) const

  Returns the function associated with a key binding.
- void **default_key_function** (Key_Func f)

  Sets the default key function for unassigned keys.
- **Fl_Text_Editor** (int X, int Y, int W, int H, const char *l=0)

  The constructor creates a new text editor widget.
- int handle (int e) override

  Event handling.
- int insert_mode () const

  Gets the current insert mode; if non-zero, new text is inserted before the current cursor position.
- void insert_mode (int b)

  Sets the current insert mode; if non-zero, new text is inserted before the current cursor position.
- void **remove_all_key_bindings** ()

  Removes all of the key bindings associated with the text editor or list.
- void **remove_all_key_bindings** (Key_Binding **list)

  Removes all of the key bindings associated with the text editor or list.
- void **remove_key_binding** (int key, int state)

  Removes the key binding associated with the key "key" of state "state".
- void remove_key_binding (int key, int state, Key_Binding **list)

  Removes the key binding associated with the key `key` of state `state` from the Key_Binding list `list`.
- int tab_nav () const

  Check if Tab focus navigation is enabled.
- void tab_nav (int val)

  Enables or disables Tab key focus navigation.

Public Member Functions inherited from Fl_Text_Display

- Fl_Text_Buffer * buffer () const

  Gets the current text buffer associated with the text widget.
- void buffer (Fl_Text_Buffer &buf)

  Sets the current text buffer associated with the text widget.
- void buffer (Fl_Text_Buffer *buf)

  Attach a text buffer to display, replacing the current buffer (if any).
- double col_to_x (double col) const

  Convert a column number into an x pixel position.
- int count_lines (int start, int end, bool start_pos_is_line_start) const

  Count the number of lines between two positions.
- Fl_Color cursor_color () const

  Gets the text cursor color.
- void cursor_color (Fl_Color n)

  Sets the text cursor color.
- int **cursor_style** () const
- void cursor_style (int style)

  Sets the text cursor style.
- virtual void display_needs_recalc ()

  Schedule a recalc_display() to be done on next draw().
- Fl_Text_Display (int X, int Y, int W, int H, const char *l=0)

  Creates a new text display widget.
- int get_absolute_top_line_number () const

  Returns the absolute (non-wrapped) line number of the first line displayed.
- Fl_Color grammar_underline_color () const

Gets the underline color for style attribute ATTR_GRAMMAR.

- void grammar_underline_color (Fl_Color color)

  Sets the underline color for style attribute ATTR_GRAMMAR.
- void **hide_cursor** ()

  Hides the text cursor.
- void highlight_data (Fl_Text_Buffer *styleBuffer, const Style_Table_Entry *styleTable, int nStyles, char unfinishedStyle, Unfinished_Style_Cb unfinishedHighlightCB, void *cbArg)

  Attach (or remove) highlight information in text display and redisplay.
- int in_selection (int x, int y) const

  Check if a pixel position is within the primary selection.
- void insert (const char *text)

  Inserts "text" at the current cursor location.
- int insert_position () const

  Gets the position of the text insertion cursor for text display.
- void insert_position (int newPos)

  Sets the position of the text insertion cursor for text display.
- int line_end (int startPos, bool startPosIsLineStart) const

  Returns the end of a line.
- int line_start (int pos) const

  Return the beginning of a line.
- Fl_Align **linenumber_align** () const

  Returns the alignment used for line numbers (if enabled).
- void linenumber_align (Fl_Align val)

  Set alignment for line numbers (if enabled).
- Fl_Color **linenumber_bgcolor** () const

  Returns the background color used for line numbers (if enabled).
- void linenumber_bgcolor (Fl_Color val)

  Set the background color used for line numbers (if enabled).
- Fl_Color **linenumber_fgcolor** () const

  Return the foreground color used for line numbers (if enabled).
- void linenumber_fgcolor (Fl_Color val)

  Set the foreground color used for line numbers (if enabled).
- Fl_Font **linenumber_font** () const

  Return the font used for line numbers (if enabled).
- void linenumber_font (Fl_Font val)

  Set the font used for line numbers (if enabled).
- const char * **linenumber_format** () const

  Returns the line number printf() format string.
- void linenumber_format (const char *val)

  Sets the printf() style format string used for line numbers.
- Fl_Fontsize **linenumber_size** () const

  Return the font size used for line numbers (if enabled).
- void linenumber_size (Fl_Fontsize val)

  Set the font size used for line numbers (if enabled).
- int **linenumber_width** () const

  Return the screen area width provided for line numbers.
- void linenumber_width (int width)

  Set width of screen area for line numbers.
- int move_down ()

  Moves the current insert position down one line.
- int move_left ()

Moves the current insert position left one character.

- int move_right ()

  Moves the current insert position right one character.

- int move_up ()

  Moves the current insert position up one line.

- void **next_word** (void)

  Moves the current insert position right one word.

- void overstrike (const char *text)

  Replaces text at the current insert position.

- int position_style (int lineStartPos, int lineLen, int lineIndex) const

  Find the correct style for a character.

- int position_to_xy (int pos, int *x, int *y) const

  Convert a character index into a pixel position.

- void **previous_word** (void)

  Moves the current insert position left one word.

- virtual void recalc_display ()

  Recalculate the display's visible lines and scrollbar sizes.

- void redisplay_range (int start, int end)

  Marks text from start to end as needing a redraw.

- void resize (int X, int Y, int W, int H) override

  Change the size of the displayed text area.

- int rewind_lines (int startPos, int nLines)

  Skip a number of lines back.

- void scroll (int topLineNum, int horizOffset)

  Scrolls the current buffer to start at the specified line and column.

- int **scroll_col** () const
- int **scroll_row** () const
- Fl_Align scrollbar_align () const

  Gets the scrollbar alignment type.

- void scrollbar_align (Fl_Align a)

  Sets the scrollbar alignment type.

- int scrollbar_size () const

  Gets the current size of the scrollbars' troughs, in pixels.

- void scrollbar_size (int newSize)

  Sets the pixel size of the scrollbars' troughs to `newSize`, in pixels.

- int scrollbar_width () const

  Returns the global value Fl::scrollbar_size() unless a specific scrollbar_width_ has been set.

- void scrollbar_width (int width)

  Sets the global Fl::scrollbar_size(), and forces this instance of the widget to use it.

- Fl_Color secondary_selection_color () const

  Gets the background color for the secondary selection block.

- void secondary_selection_color (Fl_Color color)

  Sets the background color for the secondary selection block.

- int shortcut () const
- void shortcut (int s)
- void show_cursor (int b=1)

  Shows the text cursor.

- void show_insert_position ()

  Scrolls the text buffer to show the current insert position.

- int skip_lines (int startPos, int nLines, bool startPosIsLineStart)

  Skip a number of lines forward.

- Fl_Color spelling_underline_color () const

  Gets the underline color for style attribute ATTR_SPELLING.
- void spelling_underline_color (Fl_Color color)

  Sets the underline color for style attribute ATTR_SPELLING.
- Fl_Text_Buffer * style_buffer () const

  Gets the current style buffer associated with the text widget.
- Fl_Color textcolor () const

  Gets the default color of text in the widget.
- void textcolor (Fl_Color n)

  Sets the default color of text in the widget.
- Fl_Font textfont () const

  Gets the default font used when drawing text in the widget.
- void textfont (Fl_Font s)

  Sets the default font used when drawing text in the widget.
- Fl_Fontsize textsize () const

  Gets the default size of text in the widget.
- void textsize (Fl_Fontsize s)

  Sets the default size of text in the widget.
- int word_end (int pos) const

  Moves the insert position to the end of the current word.
- int word_start (int pos) const

  Moves the insert position to the beginning of the current word.
- void wrap_mode (int wrap, int wrap_margin)

  Set the new text wrap mode.
- int wrapped_column (int row, int column) const

  Nobody knows what this function does.
- int wrapped_row (int row) const

  Nobody knows what this function does.
- double x_to_col (double x) const

  Convert an x pixel position into a column number.
- ~Fl_Text_Display ()

  Free a text display and release its associated memory.

Public Member Functions inherited from Fl_Group

- Fl_Widget *& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget *o)

  See void Fl_Group::add(Fl_Widget &w)
- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget *const * array () const

  Returns a pointer to the internal array of children.
- Fl_Group const * as_group () const override
- Fl_Group * as_group () override

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const

Returns the n'th child.

- int **children** () const

  Returns how many child widgets the group has.
- void clear ()

  Deletes all child widgets from memory recursively.
- unsigned int clip_children () const

  Returns the current clipping mode.
- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)

  Removes the widget at `index` from the group and deletes it.
- void end ()

  Exactly the same as current(this->parent()).
- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*
- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.
- void **forms_end** ()

  This is for forms compatibility only.
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

  This does insert(w, find(before)).
- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.
- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.
- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

Returns whether the widget and all of its parents are active.

- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ **as_gl_window** () const
- virtual Fl_Window ∗ as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const ∗ **as_window** () const
- void bind_deimage (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗cb, void ∗p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.

- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

Gets the image that is used as part of the widget label when in the active state.

- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".

- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

Gets the user data for this widget.

- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Static Public Member Functions

- static int kf_backspace (int c, Fl_Text_Editor *e)

  Does a backspace for key `'c'` in the current buffer of editor `'e'`.
- static int kf_c_s_move (int c, Fl_Text_Editor *e)

  Extends the current selection in the direction indicated by control key `'c'` in editor `'e'`.
- static int kf_copy (int c, Fl_Text_Editor *e)

  Does a copy of selected text or the current character in the current buffer of editor `'e'`.
- static int kf_ctrl_move (int c, Fl_Text_Editor *e)

  Moves the current text cursor in the direction indicated by control key `'c'` in editor `'e'`.
- static int kf_cut (int c, Fl_Text_Editor *e)

  Does a cut of selected text in the current buffer of editor `'e'`.
- static int kf_default (int c, Fl_Text_Editor *e)

  Inserts the text associated with key `'c'` in editor `'e'`.
- static int kf_delete (int c, Fl_Text_Editor *e)

  Does a delete of selected text or the current character in the current buffer of editor `'e'`.
- static int kf_down (int c, Fl_Text_Editor *e)

  Moves the text cursor one line down for editor `'e'`.
- static int kf_end (int c, Fl_Text_Editor *e)

Moves the text cursor to the end of the current line in editor `'e'`.

- static int kf_enter (int c, Fl_Text_Editor *e)

  Inserts a newline for key `'c'` at the current cursor position in editor `'e'`.
- static int kf_home (int, Fl_Text_Editor *e)

  Moves the text cursor to the beginning of the current line in editor `'e'`.
- static int kf_ignore (int c, Fl_Text_Editor *e)

  Ignores the key `'c'` in editor `'e'`.
- static int kf_insert (int c, Fl_Text_Editor *e)

  Toggles the insert mode for editor `'e'`.
- static int kf_left (int c, Fl_Text_Editor *e)

  Moves the text cursor one character to the left in editor `'e'`.
- static int kf_m_s_move (int c, Fl_Text_Editor *e)

  Extends the current selection in the direction indicated by meta key `'c'` in editor `'e'`.
- static int kf_meta_move (int c, Fl_Text_Editor *e)

  Moves the current text cursor in the direction indicated by meta key `'c'` in editor `'e'`.
- static int kf_move (int c, Fl_Text_Editor *e)

  Moves the text cursor in the direction indicated by key `'c'` in editor `'e'`.
- static int kf_page_down (int c, Fl_Text_Editor *e)

  Moves the text cursor down one page for editor `'e'`.
- static int kf_page_up (int c, Fl_Text_Editor *e)

  Moves the text cursor up one page for editor `'e'`.
- static int kf_paste (int c, Fl_Text_Editor *e)

  Does a paste of selected text in the current buffer of editor `'e'`.
- static int kf_redo (int c, Fl_Text_Editor *e)

  Redo last undo action.
- static int kf_right (int c, Fl_Text_Editor *e)

  Moves the text cursor one character to the right for editor `'e'`.
- static int kf_select_all (int c, Fl_Text_Editor *e)

  Selects all text in the current buffer in editor `'e'`.
- static int kf_shift_move (int c, Fl_Text_Editor *e)

  Extends the current selection in the direction of key `'c'` in editor `'e'`.
- static int kf_undo (int c, Fl_Text_Editor *e)

  Undo last edit in the current buffer of editor `'e'`.
- static int kf_up (int c, Fl_Text_Editor *e)

  Moves the text cursor one line up for editor `'e'`.

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()

  Returns the currently active group.
- static void current (Fl_Group *g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Member Functions

- int **handle_key** ()

  Handles a key press in the editor.
- void **maybe_do_callback** (Fl_Callback_Reason reason=FL_REASON_CHANGED)

  does or does not a callback according to changed() and when() settings

Protected Member Functions inherited from Fl_Text_Display

- void absolute_top_line_number (int oldFirstChar)

  Re-calculate absolute top line number for a change in scroll position.
- void calc_last_char ()

  Update last display character index.
- void calc_line_starts (int startLine, int endLine)

  Update the line starts array.
- void clear_rect (int style, int x, int y, int width, int height) const

  Clear a rectangle with the appropriate background color for `style`.
- void display_insert ()

  Scroll the display to bring insertion cursor into view.
- void draw () override

  Draw the widget.
- void draw_cursor (int, int)

  Draw a cursor with top center at `X`, `Y`.
- void draw_line_numbers (bool clearAll)

  Refresh the line number area.
- void draw_range (int start, int end)

  Draw a range of text.
- void draw_string (int style, int x, int y, int toX, const char ∗string, int nChars) const

  Draw a text segment in a single style.
- void draw_text (int X, int Y, int W, int H)

  Refresh a rectangle of the text display.
- void draw_vline (int visLineNum, int leftClip, int rightClip, int leftCharIndex, int rightCharIndex)

  Draw a single line of text.
- int empty_vlines () const

  Return true if there are lines visible with no corresponding buffer text.
- void extend_range_for_styles (int ∗start, int ∗end)

  I don't know what this does!
- void find_line_end (int pos, bool start_pos_is_line_start, int ∗lineEnd, int ∗nextLineStart) const

  Finds both the end of the current line and the start of the next line.
- void find_wrap_range (const char ∗deletedText, int pos, int nInserted, int nDeleted, int ∗modRangeStart, int ∗modRangeEnd, int ∗linesInserted, int ∗linesDeleted)

  Wrapping calculations.
- int find_x (const char ∗s, int len, int style, int x) const

  Find the index of the character that lies at the given x position / closest cursor position.
- int handle_rmb (int readonly)

  Handle right mouse button down events.
- int handle_vline (int mode, int lineStart, int lineLen, int leftChar, int rightChar, int topClip, int bottomClip, int leftClip, int rightClip) const

  Universal pixel machine.
- int longest_vline () const

  Find the longest line of all visible lines.
- void maintain_absolute_top_line_number (int state)

Line numbering stuff, currently unused.

- int maintaining_absolute_top_line_number () const

  Returns true if a separate absolute top line number is being maintained.
- void measure_deleted_lines (int pos, int nDeleted)

  Wrapping calculations.
- double measure_proportional_character (const char ∗s, int colNum, int pos) const

  Wrapping calculations.
- int measure_vline (int visLineNum) const

  Returns the width in pixels of the displayed line pointed to by "visLineNum".
- void offset_line_starts (int newTopLineNum)

  Offset line start counters for a new vertical scroll position.
- int position_to_line (int pos, int ∗lineNum) const

  Convert a position index into a line number offset.
- int position_to_linecol (int pos, int ∗lineNum, int ∗column) const

  Find the line and column number of position `pos`.
- void reset_absolute_top_line_number ()

  Reestablish the absolute (non-wrapped) top line number.
- int scroll_ (int topLineNum, int horizOffset)

  Scrolls the current buffer to start at the specified line and column.
- double string_width (const char ∗string, int length, int style) const

  Find the width of a string in the font of a particular style.
- void update_h_scrollbar ()

  Update horizontal scrollbar.
- void update_line_starts (int pos, int charsInserted, int charsDeleted, int linesInserted, int linesDeleted, int ∗scrolled)

  Update line start arrays and variables.
- void update_v_scrollbar ()

  Update vertical scrollbar.
- int vline_length (int visLineNum) const

  Count number of bytes in a visible line.
- int wrap_uses_character (int lineEndPos) const

  Check if the line break is caused by a newline or by line wrapping.
- void wrapped_line_counter (Fl_Text_Buffer ∗buf, int startPos, int maxPos, int maxLines, bool startPosIs↩LineStart, int styleBufOffset, int ∗retPos, int ∗retLines, int ∗retLineStart, int ∗retLineEnd, bool countLast↩LineMissingNewLine=true) const

  Wrapping calculations.
- int xy_to_position (int x, int y, int PosType=CHARACTER_POS) const

  Translate a pixel position into a character index.
- void xy_to_rowcol (int x, int y, int ∗row, int ∗column, int PosType=CHARACTER_POS) const

  Translate pixel coordinates into row and column.

Protected Member Functions inherited from Fl_Group

- Fl_Rect ∗ bounds ()

  Returns the internal array of widget sizes and positions.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget ∗, int)

Allow derived groups to act when a widget is added as a child.

- virtual int on_move (int, int)

  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)

  Allow derived groups to act when a child widget is removed from the group.
- int * sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Static Protected Attributes

- static Key_Binding * global_key_bindings

  Global key binding list.

Additional Inherited Members

Protected Types inherited from Fl_Text_Display

- enum {
  DRAW_LINE , **FIND_INDEX** , **FIND_INDEX_FROM_ZERO** , **GET_WIDTH** ,
  FIND_CURSOR_INDEX }

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Static Protected Member Functions inherited from Fl_Text_Display

- static void buffer_modified_cb (int pos, int nInserted, int nDeleted, int nRestyled, const char *deletedText, void *cbArg)

  This is called whenever the buffer is modified.
- static void buffer_predelete_cb (int pos, int nDeleted, void *cbArg)

  This is called before any characters are deleted.
- static void **h_scrollbar_cb** (Fl_Scrollbar *w, Fl_Text_Display *d)

  Callback for drag or valueChanged on horizontal scrollbar.
- static void scroll_timer_cb (void *)

  Timer callback for scroll events.
- static void **v_scrollbar_cb** (Fl_Scrollbar *w, Fl_Text_Display *d)

  Callback for drag or valueChanged on vertical scrollbar.

Protected Attributes inherited from Fl_Text_Display

- int **damage_range1_end**
- int **damage_range1_start**
- int **damage_range2_end**
- int **damage_range2_start**
- int **display_insert_position_hint**
- bool **display_needs_recalc_**
- int **dragging**
- int **dragPos**
- int **dragType**
- Fl_Color **grammar_underline_color_**
- Fl_Align **linenumber_align_**
- Fl_Color **linenumber_bgcolor_**
- Fl_Color **linenumber_fgcolor_**
- Fl_Font **linenumber_font_**

- const char * **linenumber_format_**
- Fl_Fontsize **linenumber_size_**
- int **mAbsTopLineNum**
- Fl_Text_Buffer * **mBuffer**
- double **mColumnScale**
- int **mContinuousWrap**
- Fl_Color **mCursor_color**
- int **mCursorOldY**
- int **mCursorOn**
- int **mCursorPos**
- int **mCursorPreferredXPos**
- int **mCursorStyle**
- int **mCursorToHint**
- int **mFirstChar**
- void * **mHighlightCBArg**
- int **mHorizOffset**
- int **mHorizOffsetHint**
- Fl_Scrollbar * **mHScrollBar**
- int **mLastChar**
- int **mLineNumLeft**
- int **mLineNumWidth**
- int * **mLineStarts**
- int **mMaxsize**
- int **mModifyingTabDistance**
- int **mNBufferLines**
- int **mNeedAbsTopLineNum**
- int **mNLinesDeleted**
- int **mNStyles**
- int **mNVisibleLines**
- Fl_Text_Buffer * **mStyleBuffer**
- const Style_Table_Entry * **mStyleTable**
- int **mSuppressResync**
- int **mTopLineNum**
- int **mTopLineNumHint**
- Unfinished_Style_Cb **mUnfinishedHighlightCB**
- char **mUnfinishedStyle**
- Fl_Scrollbar * **mVScrollBar**
- int **mWrapMarginPix**
- Fl_Align **scrollbar_align_**
- int **scrollbar_width_**
- Fl_Color **secondary_selection_color_**
- int **shortcut_**
- Fl_Color **spelling_underline_color_**
- struct {
  - int **h**
  - int **w**
  - int **x**
  - int **y**

  } **text_area**

- Fl_Color **textcolor_**
- Fl_Font **textfont_**
- Fl_Fontsize **textsize_**

13.137.1 Detailed Description

This is the FLTK text editor widget.
It allows the user to edit multiple lines of text and supports highlighting and scrolling. The buffer that is displayed in the widget is managed by the Fl_Text_Buffer class.

13.137.2 Member Function Documentation

13.137.2.1 add_key_binding()

```
void Fl_Text_Editor::add_key_binding (
            int key,
            int state,
            Key_Func function,
            Key_Binding ** list )
```

Adds a `key` of state `state` with the function `function` to an arbitrary key binding list `list`.
This can be used in derived classes to add global key bindings by using the global (static) Key_Binding list Fl_Text_Editor::global_key_bindings.

13.137.2.2 handle()

```
int Fl_Text_Editor::handle (
            int e )  [override], [virtual]
```

Event handling.
Reimplemented from Fl_Text_Display.

13.137.2.3 insert_mode() [1/2]

```
int Fl_Text_Editor::insert_mode ( ) const  [inline]
```

Gets the current insert mode; if non-zero, new text is inserted before the current cursor position.
Otherwise, new text replaces text at the current cursor position.

13.137.2.4 insert_mode() [2/2]

```
void Fl_Text_Editor::insert_mode (
            int b )  [inline]
```

Sets the current insert mode; if non-zero, new text is inserted before the current cursor position.
Otherwise, new text replaces text at the current cursor position.

13.137.2.5 kf_backspace()

```
int Fl_Text_Editor::kf_backspace (
            int c,
            Fl_Text_Editor * e )  [static]
```

Does a backspace for key `'c'` in the current buffer of editor `'e'`.
Any current selection is deleted. Otherwise, the character left is deleted and the cursor moved. The key value `'c'` is currently unused.

13.137.2.6 kf_c_s_move()

```
int Fl_Text_Editor::kf_c_s_move (
            int c,
            Fl_Text_Editor * e )  [static]
```

Extends the current selection in the direction indicated by control key `'c'` in editor `'e'`.

See also

kf_ctrl_move().

13.137.2.7 kf_copy()

```
int Fl_Text_Editor::kf_copy (
            int c,
            Fl_Text_Editor * e )  [static]
```

Does a copy of selected text or the current character in the current buffer of editor `'e'`.
The key value `'c'` is currently unused.

13.137.2.8 kf_ctrl_move()

```
int Fl_Text_Editor::kf_ctrl_move (
            int c,
            Fl_Text_Editor * e )  [static]
```

Moves the current text cursor in the direction indicated by control key `'c'` in editor `'e'`.
Supported values for 'c' are currently:

```
FL_Home      -- moves the cursor to the beginning of the document
FL_End       -- moves the cursor to the end of the document
FL_Left      -- moves the cursor left one word
FL_Right     -- moves the cursor right one word
FL_Up        -- scrolls up one line, without moving cursor
FL_Down      -- scrolls down one line, without moving cursor
FL_Page_Up   -- moves the cursor to the beginning of the top line on the current page
FL_Page_Down -- moves the cursor to the beginning of the last line on the current page
```

13.137.2.9 kf_cut()

```
int Fl_Text_Editor::kf_cut (
            int c,
            Fl_Text_Editor * e )  [static]
```

Does a cut of selected text in the current buffer of editor `'e'`.
The key value `'c'` is currently unused.

13.137.2.10 kf_default()

```
int Fl_Text_Editor::kf_default (
            int c,
            Fl_Text_Editor * e )  [static]
```

Inserts the text associated with key `'c'` in editor `'e'`.
Honors the current selection and insert/overstrike mode.

13.137.2.11 kf_delete()

```
int Fl_Text_Editor::kf_delete (
            int c,
            Fl_Text_Editor * e )  [static]
```

Does a delete of selected text or the current character in the current buffer of editor `'e'`.
The key value `'c'` is currently unused.

13.137.2.12 kf_down()

```
int Fl_Text_Editor::kf_down (
            int c,
            Fl_Text_Editor * e )  [static]
```

Moves the text cursor one line down for editor `'e'`.
Same as kf_move(FL_Down, e). The key value `'c'` is currently unused.

13.137.2.13 kf_end()

```
int Fl_Text_Editor::kf_end (
            int c,
            Fl_Text_Editor * e )  [static]
```

Moves the text cursor to the end of the current line in editor `'e'`.
Same as kf_move(FL_End, e). The key value `'c'` is currently unused.

13.137.2.14 kf_enter()

```
int Fl_Text_Editor::kf_enter (
            int c,
            Fl_Text_Editor * e )  [static]
```

Inserts a newline for key `'c'` at the current cursor position in editor `'e'`.
The key value `'c'` is currently unused.

13.137.2.15 kf_home()

```
int Fl_Text_Editor::kf_home (
            int ,
            Fl_Text_Editor * e )  [static]
```

Moves the text cursor to the beginning of the current line in editor `'e'`.
Same as kf_move(FL_Home, e). The key value `'c'` is currently unused.

13.137.2.16 kf_ignore()

```
int Fl_Text_Editor::kf_ignore (
            int c,
            Fl_Text_Editor * e )  [static]
```

Ignores the key `'c'` in editor `'e'`.
This method can be used as a keyboard binding to disable a key that might otherwise be handled or entered as text. An example would be disabling FL_Escape, so that it isn't added to the buffer when invoked by the user.

13.137.2.17 kf_insert()

```
int Fl_Text_Editor::kf_insert (
            int c,
            Fl_Text_Editor * e )  [static]
```

Toggles the insert mode for editor `'e'`.
The key value `'c'` is currently unused.

13.137.2.18 kf_left()

```
int Fl_Text_Editor::kf_left (
            int c,
            Fl_Text_Editor * e )  [static]
```

Moves the text cursor one character to the left in editor `'e'`.
Same as kf_move(FL_Left, e). The key value `'c'` is currently unused.

13.137.2.19 kf_m_s_move()

```
int Fl_Text_Editor::kf_m_s_move (
            int c,
            Fl_Text_Editor * e )  [static]
```

Extends the current selection in the direction indicated by meta key `'c'` in editor `'e'`.

See also

kf_meta_move().

13.137.2.20 kf_meta_move()

```
int Fl_Text_Editor::kf_meta_move (
            int c,
            Fl_Text_Editor * e )  [static]
```

Moves the current text cursor in the direction indicated by meta key `'c'` in editor `'e'`.
Supported values for 'c' are currently:

```
FL_Up        -- moves cursor to the beginning of the current document
```

```
FL_Down      -- moves cursor to the end of the current document
FL_Left      -- moves the cursor to the beginning of the current line
FL_Right     -- moves the cursor to the end of the current line
```

13.137.2.21 kf_move()

```
int Fl_Text_Editor::kf_move (
            int c,
            Fl_Text_Editor * e )  [static]
```

Moves the text cursor in the direction indicated by key `'c'` in editor `'e'`.
Supported values for 'c' are currently:

```
FL_Home      -- moves the cursor to the beginning of the current line
FL_End       -- moves the cursor to the end of the current line
FL_Left      -- moves the cursor left one character
FL_Right     -- moves the cursor right one character
FL_Up        -- moves the cursor up one line
FL_Down      -- moves the cursor down one line
FL_Page_Up   -- moves the cursor up one page
FL_Page_Down -- moves the cursor down one page
```

13.137.2.22 kf_page_down()

```
int Fl_Text_Editor::kf_page_down (
            int c,
            Fl_Text_Editor * e )  [static]
```

Moves the text cursor down one page for editor `'e'`.
Same as kf_move(FL_Page_Down, e). The key value `'c'` is currently unused.

13.137.2.23 kf_page_up()

```
int Fl_Text_Editor::kf_page_up (
            int c,
            Fl_Text_Editor * e )  [static]
```

Moves the text cursor up one page for editor `'e'`.
Same as kf_move(FL_Page_Up, e). The key value `'c'` is currently unused.

13.137.2.24 kf_paste()

```
int Fl_Text_Editor::kf_paste (
            int c,
            Fl_Text_Editor * e )  [static]
```

Does a paste of selected text in the current buffer of editor `'e'`.
Any current selection is replaced with the pasted content. The key value `'c'` is currently unused.

13.137.2.25 kf_redo()

```
int Fl_Text_Editor::kf_redo (
            int c,
            Fl_Text_Editor * e )  [static]
```

Redo last undo action.
Also deselects previous selection. The key value `'c'` is currently unused.

13.137.2.26 kf_right()

```
int Fl_Text_Editor::kf_right (
            int c,
            Fl_Text_Editor * e )  [static]
```

Moves the text cursor one character to the right for editor `'e'`.
Same as kf_move(FL_Right, e). The key value `'c'` is currently unused.

13.137.2.27 kf_select_all()

```
int Fl_Text_Editor::kf_select_all (
            int c,
            Fl_Text_Editor * e )  [static]
```

Selects all text in the current buffer in editor `'e'`.
The key value `'c'` is currently unused.

13.137.2.28 kf_shift_move()

```
int Fl_Text_Editor::kf_shift_move (
            int c,
            Fl_Text_Editor * e )  [static]
```

Extends the current selection in the direction of key `'c'` in editor `'e'`.

See also

kf_move()

13.137.2.29 kf_undo()

```
int Fl_Text_Editor::kf_undo (
            int c,
            Fl_Text_Editor * e )  [static]
```

Undo last edit in the current buffer of editor `'e'`.
Also deselects previous selection. The key value `'c'` is currently unused.

13.137.2.30 kf_up()

```
int Fl_Text_Editor::kf_up (
            int c,
            Fl_Text_Editor * e )  [static]
```

Moves the text cursor one line up for editor `'e'`.
Same as kf_move(FL_Up, e). The key value `'c'` is currently unused.

13.137.2.31 remove_key_binding()

```
void Fl_Text_Editor::remove_key_binding (
            int key,
            int state,
            Key_Binding ** list )
```

Removes the key binding associated with the key `key` of state `state` from the Key_Binding list `list`.
This can be used in derived classes to remove global key bindings by using the global (static) Key_Binding list Fl_Text_Editor::global_key_bindings.

13.137.2.32 tab_nav() [1/2]

```
int Fl_Text_Editor::tab_nav ( ) const
```

Check if Tab focus navigation is enabled.
If disabled (default), hitting Tab inserts a tab character into the editor buffer.
If enabled, hitting Tab navigates focus to the next widget, and Shift-Tab navigates focus to the previous widget.

Returns

if Tab inserts tab characters or moves the focus

Return values

0	Tab inserts tab characters (default)
1	Tab navigation is enabled.

See also

tab_nav(int), Fl::OPTION_ARROW_FOCUS.

Version

1.3.4 ABI feature

13.137.2.33 tab_nav() [2/2]

```
void Fl_Text_Editor::tab_nav (
            int val )
```

Enables or disables Tab key focus navigation.
When disabled (default), tab characters are inserted into Fl_Text_Editor. Only the mouse can change focus. This behavior is desireable when Fl_Text_Editor is used, e.g. in a source code editor.
When enabled, Tab navigates focus to the next widget, and Shift-Tab navigates focus to the previous widget. This behavior is desireable when Fl_Text_Editor is used e.g. in a database input form.
Currently, this method is implemented as a convenience method that adjusts the key bindings for the Tab key. This implementation detail may change in the future. Know that changing the editor's key bindings for Tab and Shift-Tab may affect tab navigation.

Parameters

in	*val*	If `val` is 0, Tab inserts a tab character (default). If `val` is 1, Tab navigates widget focus.

See also

tab_nav(), Fl::OPTION_ARROW_FOCUS.

Version

1.3.4 ABI feature

13.137.3 Member Data Documentation

13.137.3.1 global_key_bindings

```
Key_Binding* Fl_Text_Editor::global_key_bindings  [static], [protected]
```

Global key binding list.
Derived classes can add key bindings for all Fl_Text_Editor widgets by adding a Key_Binding to this list.

See also

add_key_binding(int key, int state, Key_Func f, Key_Binding** list);

The documentation for this class was generated from the following files:

- Fl_Text_Editor.H
- Fl_Text_Editor.cxx

13.138 Fl_Text_Selection Class Reference

This is an internal class for Fl_Text_Buffer to manage text selections.
`#include <Fl_Text_Buffer.H>`

Public Member Functions

- int end () const

Returns the byte offset to the character after the last selected character.

- int includes (int pos) const

  Returns true if position `pos` is in the Fl_Text_Selection.
- int length () const

  Returns the size in bytes of the selection.
- int position (int *startpos, int *endpos) const
- bool selected () const

  Returns true if any text is selected.
- void selected (bool b)

  Modifies the 'selected' flag.
- int selected (int *startpos, int *endpos) const

  Returns the status and the positions of this selection.
- void set (int startpos, int endpos)

  Sets the selection range.
- int start () const

  Returns the byte offset to the first selected character.
- void update (int pos, int nDeleted, int nInserted)

  Updates a selection after text was modified.

Protected Attributes

- int **mEnd**

  byte offset to the character after the last selected character
- bool **mSelected**

  this flag is set if any text is selected
- int **mStart**

  byte offset to the first selected character

Friends

- class **Fl_Text_Buffer**

13.138.1 Detailed Description

This is an internal class for Fl_Text_Buffer to manage text selections.
All methods use byte (not UTF-8 character) offsets and start at 0. This class works correctly with UTF-8 strings assuming that the parameters for all calls are on character boundaries.
If the selection is inactive (not currently used), then selected() returns `false` and start() and end() return 0 (zero).
The stored offsets are in ascending order, hence the following conditions are true (pseudo code):

```
if ( !selected() ) :  (start() == 0) && (end() == 0) && (start() == end())
if (  selected() ) :  start() < end()
always             :  0 <= start() <= end()
always             :  length() == end() - start()
```

The selection size in bytes can always (unconditionally) be computed by

```
int size = sel->end() - sel->start();
```

See also

length()

Note

The **protected** member variables `mStart` and `mEnd` are not necessarily 0 (zero) if mSelected == `false` because they are not cleared when `selected(false)` is called (as of Jul 2017). This may be changed in the future.

13.138.2 Member Function Documentation

13.138.2.1 end()

```
int Fl_Text_Selection::end ( ) const  [inline]
```

Returns the byte offset to the character after the last selected character.
The returned offset is only valid if selected() returns true (non-zero). The offset is 0 if no text is selected (since FLTK 1.4.0).

Note

In FLTK 1.3.x the returned offset could be non-zero even if selected() would have returned 0.

Returns

byte offset or 0 if not selected.

13.138.2.2 includes()

```
int Fl_Text_Selection::includes (
            int pos ) const
```

Returns true if position `pos` is in the Fl_Text_Selection.
`pos` must be at a character boundary.

13.138.2.3 length()

```
int Fl_Text_Selection::length ( ) const  [inline]
```

Returns the size in bytes of the selection.
This is a convenience method. It always returns the same as
`end() - start()`
and it returns 0 if selected() == false.

Returns

size in bytes or 0 if not selected.

Since

FLTK 1.4.0

13.138.2.4 position()

```
int Fl_Text_Selection::position (
            int * startpos,
            int * endpos ) const  [inline]
```

Deprecated "since 1.4.0 - use selected(startpos, endpos) instead"

13.138.2.5 selected() [1/3]

```
bool Fl_Text_Selection::selected ( ) const  [inline]
```

Returns true if any text is selected.

Returns

`true` if any text has been selected, or `false` if no text is selected.

13.138.2.6 selected() [2/3]

```
void Fl_Text_Selection::selected (
            bool b )  [inline]
```

Modifies the 'selected' flag.

Parameters

b	new flag

13.138.2.7 selected() [3/3]

```
int Fl_Text_Selection::selected (
            int * startpos,
            int * endpos ) const
```

Returns the status and the positions of this selection.
This method returns the same as `selected()` as an `int` (0 or 1) in its return value and the offsets to the start of the selection in `startpos` and to the byte after the last selected character in `endpos`, if selected() is `true`.
If selected() is `false`, both offsets are set to 0.

Note

In FLTK 1.3.x `startpos` and `endpos` were **not modified** if selected() was false.

Parameters

startpos	return byte offset to first selected character
endpos	return byte offset pointing after last selected character

Returns

whether the selection is active (selected()) or not

Return values

0	if not selected
1	if selected

See also

selected(), start(), end()

13.138.2.8 set()

```
void Fl_Text_Selection::set (
            int startpos,
            int endpos )
```

Sets the selection range.
`startpos` and `endpos` must be at a character boundary.
If `startpos` != `endpos` selected() is set to true, else to false.
If `startpos` is greater than `endpos` they are swapped so that `startpos` <= `endpos`.

Parameters

in	*startpos*	byte offset to first selected character
in	*endpos*	byte offset pointing after last selected character

13.138.2.9 start()

```
int Fl_Text_Selection::start ( ) const  [inline]
```

Returns the byte offset to the first selected character.
The returned offset is only valid if selected() returns true. If the selection is not valid the returned offset is 0 since FLTK 1.4.0.

Note

In FLTK 1.3.x the returned offset could be non-zero even if selected() would have returned 0.

Returns

byte offset or 0 if not selected.

13.138.2.10 update()

```
void Fl_Text_Selection::update (
            int pos,
            int nDeleted,
            int nInserted )
```

Updates a selection after text was modified.
Updates an individual selection for changes in the corresponding text.

Parameters

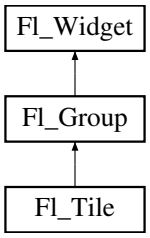
pos	byte offset into text buffer at which the change occurred
nDeleted	number of bytes deleted from the buffer
nInserted	number of bytes inserted into the buffer

The documentation for this class was generated from the following files:

- Fl_Text_Buffer.H
- Fl_Text_Buffer.cxx

13.139 Fl_Tile Class Reference

The Fl_Tile class lets you resize its children by dragging the border between them.
Inheritance diagram for Fl_Tile:



Classes

- struct Size_Range

Public Member Functions

- virtual void drag_intersection (int oldx, int oldy, int newx, int newy)

  Drags the intersection at (`oldx`,`oldy`) to (`newx`,`newy`).
- Fl_Tile (int X, int Y, int W, int H, const char *L=0)

  Creates a new Fl_Tile widget using the given position, size, and label string.
- int handle (int event) override

Handles the specified event.

- void init_size_range (int default_min_w=-1, int default_min_h=-1)

  Initialize the size range mode of Fl_Tile and set the default minimum width and height.
- virtual void move_intersection (int oldx, int oldy, int newx, int newy)

  Drags the intersection at (`oldx`,`oldy`) to (`newx`,`newy`).
- void position (int oldx, int oldy, int newx, int newy)
- void **position** (int x, int y)
- void resize (int X, int Y, int W, int H) override

  Resizes the Fl_Tile widget and its children.
- void size_range (Fl_Widget *w, int minw, int minh, int maxw=0x7FFFFFFF, int maxh=0x7FFFFFFF)

  Set the allowed size range for the give child widget.
- void size_range (int index, int minw, int minh, int maxw=0x7FFFFFFF, int maxh=0x7FFFFFFF)

  Set the allowed size range for the child at the given index.
- ~**Fl_Tile** () override

  Destructor.

Public Member Functions inherited from Fl_Group

- Fl_Widget *& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget *o)

  See void Fl_Group::add(Fl_Widget &w)
- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget *const * array () const

  Returns a pointer to the internal array of children.
- Fl_Group const * as_group () const override
- Fl_Group * as_group () override

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const

  Returns the n'th child.
- int **children** () const

  Returns how many child widgets the group has.
- void clear ()

  Deletes all child widgets from memory recursively.
- unsigned int clip_children () const

  Returns the current clipping mode.
- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)

  Removes the widget at `index` from the group and deletes it.
- void end ()

  Exactly the same as current(this->parent()).
- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*
- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.

- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.
- void **forms_end** ()

  This is for forms compatibility only.
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

  This does insert(w, find(before)).
- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.
- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.
- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.

- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

Sets the damage bits for the widget.

- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.

- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

Sets a widget to output only.

- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.

- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- Fl_Cursor cursor (int n) const

  Returns the cursor for cursor index n.
- int on_insert (Fl_Widget ∗, int) override

  Insert a new entry in the size range list.
- int on_move (int, int) override

  Move the entry in the size range list.
- void on_remove (int) override

  Remove the entry from the size range list.
- void request_grow_b (int old_b, int &new_b, Fl_Rect ∗final_size)

  Request for children to change their layout.
- void request_grow_l (int old_l, int &new_l, Fl_Rect ∗final_size)

  Request for children to change their layout.
- void request_grow_r (int old_r, int &new_r, Fl_Rect ∗final_size)

  Request for children to change their layout.
- void request_grow_t (int old_t, int &new_t, Fl_Rect ∗final_size)

  Request for children to change their layout.
- void request_shrink_b (int old_b, int &new_b, Fl_Rect ∗final_size)

  Request for children to change their layout.
- void request_shrink_l (int old_l, int &new_l, Fl_Rect ∗final_size)

  Request for children to change their layout.
- void request_shrink_r (int old_r, int &new_r, Fl_Rect ∗final_size)

  Request for children to change their layout.
- void request_shrink_t (int old_t, int &new_t, Fl_Rect ∗final_size)

  Request for children to change their layout.
- void set_cursor (int n)

  Set one of four cursors used for dragging etc. . .

Protected Member Functions inherited from Fl_Group

- Fl_Rect ∗ bounds ()

  Returns the internal array of widget sizes and positions.
- void draw () override

  Draws the widget.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.

- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- int * sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Protected Attributes

- int **cursor_**

  current cursor index (0..3)
- Fl_Cursor * **cursors_**

  points at the array of 4 cursors (may be overridden)
- int **default_min_h_**
- int **default_min_w_**
- Size_Range * **size_range_**
- int **size_range_capacity_**
- int **size_range_size_**

Additional Inherited Members

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()

  Returns the currently active group.
- static void current (Fl_Group *g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.139.1 Detailed Description

The Fl_Tile class lets you resize its children by dragging the border between them.

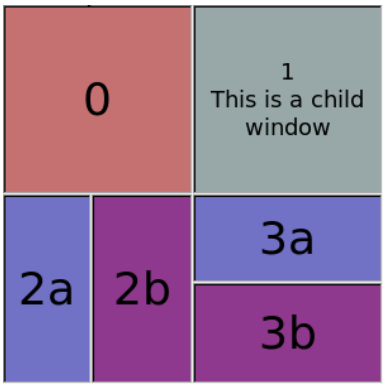


Figure 13.60 Fl_Tile

For the tiling to work correctly, the children of an Fl_Tile **must** cover the entire area of the widget, but **must not** overlap. This means that all children must touch each other at their edges, and no gaps can be left inside the Fl_Tile.
Fl_Tile does not normally draw any graphics of its own. The "borders" which can be seen in the snapshot above are actually part of the children. Their boxtypes have been set to FL_DOWN_BOX creating the impression of "ridges" where the boxes touch. What you see are actually two adjacent FL_DOWN_BOX's drawn next to each other. All neighboring widgets share the same edge - the widget's thick borders make it appear as though the widgets aren't actually touching, but they are. If the edges of adjacent widgets do not touch, then it will be impossible to drag the corresponding edges.

Note

Fl_Tile works in two distinctive modes. In classic mode, the range of motion for edges and intersections is controlled using an invisible child that is marked as the `resizable()` widget of the tile group. Classic mode is described in detail a few paragraphs down.

Fl_Tile size_range mode

By assigning a default minimum size to all children with `Fl_Tile::init_size_range(int default_minimum_width,` or by assigning minimal sizes to individual children with `size_range(Fl_Widget *child, int minimum_width, int` the tile group is put into size_range operation mode.
In this mode, the child that is marked resizable() will behave as it would in a regular Fl_Group widget. When dragging edges or intersections with the mouse, Fl_Tile will ensure that none of the children shrinks to a size that is smaller than requested. When resizing the Fl_Tile group, size ranges are not enforced by the tile. Instead, the size range of the enclosing window should be limited to a valid range.
Tile does not differentiate between visible and invisible children. If children are created smaller than their assigned minimum size, dragging intersections may cause unexpected jumps in size. Zero width or height widget are not harmful, but should be avoided.
Example for a center document tile and two tool boxes on the left and right

```
Fl_Window win(400, 300, "My App");

Fl_Tile tile(0, 0, 400, 300);

Fl_Box left_tool_box(0, 0, 100, 300, "Tools");
left_tool_box.box(FL_DOWN_BOX);
tile.size_range(&left_tool_box, 50, 50);

Fl_Box document(100, 0, 200, 300, "Document");
document.box(FL_DOWN_BOX);
tile.size_range(&document, 100, 50);

Fl_Box right_tool_box(300, 0, 100, 300, "More\nTools");
right_tool_box.box(FL_DOWN_BOX);
tile.size_range(&right_tool_box, 50, 50);

tile.end();
tile.resizable(document);

win.end();
win.resizable(tile);
win.show(argc,argv);
win.size_range(200, 50);
```

Fl_Tile classic mode

Fl_Tile allows objects to be resized to zero dimensions. To prevent this you can use the resizable() to limit where corners can be dragged to. For more information see note below.
Even though objects can be resized to zero sizes, they must initially have non-zero sizes so the Fl_Tile can figure out their layout. If desired, call position() after creating the children but before displaying the window to set the borders where you want.
Note on resizable(Fl_Widget &w): The "resizable" child widget (which should be invisible) limits where the borders can be dragged to. All dragging will be limited inside the resizable widget's borders. If you don't set it, it will be possible to drag the borders right to the edges of the Fl_Tile widget, and thus resize objects on the edges to zero width or height. When the entire Fl_Tile widget is resized, the resizable() widget will keep its border distance to all borders the same (this is normal resize behavior), so that you can effectively set a border width that will never change. To ensure correct event delivery to all child widgets the resizable() widget must be the first child of the Fl_Tile widget group. Otherwise some events (e.g. FL_MOVE and FL_ENTER) might be consumed by the resizable() widget so that they are lost for widgets covered (overlapped) by the resizable() widget.

Note

You can still resize widgets **inside** the resizable() to zero width and/or height, i.e. box **2b** above to zero width and box **3a** to zero height.

See also

void Fl_Group::resizable(Fl_Widget &w)

Example for resizable with 20 pixel border distance:

```
int dx = 20, dy = dx;
Fl_Tile tile(50,50,300,300);
// create resizable() box first
Fl_Box r(tile.x()+dx,tile.y()+dy,tile.w()-2*dx,tile.h()-2*dy);
tile.resizable(r);
// ... create widgets inside tile (see test/tile.cxx) ...
tile.end();
```

See also the complete example program in test/tile.cxx.

13.139.2 Constructor & Destructor Documentation

13.139.2.1 Fl_Tile()

```
Fl_Tile::Fl_Tile (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

Creates a new Fl_Tile widget using the given position, size, and label string.
The default boxtype is FL_NO_BOX.
The destructor *also deletes all the children*. This allows a whole tree to be deleted at once, without having to keep a pointer to all the children in the user code. A kludge has been done so the Fl_Tile and all of its children can be automatic (local) variables, but you must declare the Fl_Tile *first*, so that it is destroyed last.

See also

class Fl_Group

13.139.3 Member Function Documentation

13.139.3.1 cursor()

```
Fl_Cursor Fl_Tile::cursor (
            int n ) const  [inline], [protected]
```

Returns the cursor for cursor index n.

See also

Fl_Tile::set_cursor(int)

13.139.3.2 drag_intersection()

```
void Fl_Tile::drag_intersection (
            int oldx,
            int oldy,
            int newx,
            int newy )  [virtual]
```

Drags the intersection at (`oldx`,`oldy`) to (`newx`,`newy`).

See also

Fl_Tile::move_intersection(int oldx, int oldy, int newx, int newy) , but this method does not call init_sizes() and is used for interactive children layout using the mouse.

Parameters

in	*oldx,oldy*	move the intersection at this coordinate, pass zero to disable drag in that direction.
in	*newx,newy*	move the intersection as close to this new coordinate as possible

13.139.3.3 handle()

```
int Fl_Tile::handle (
            int event )  [override], [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to *override* the behavior of the base class. This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to *filter* event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you *consumed* the event by returning 1 even if you didn't *do* anything with it.

Parameters

in	*event*	the kind of event received

Return values

0	if the event was not used or understood
1	if the event was used and can be deleted

See also

Fl_Event

Reimplemented from Fl_Group.

13.139.3.4 init_size_range()

```
void Fl_Tile::init_size_range (
            int default_min_w = -1,
            int default_min_h = -1 )
```

Initialize the size range mode of Fl_Tile and set the default minimum width and height.

The default minimum width and height is the size of the mouse pointer grab area at about 4 pixel units.

Parameters

in	*default_min_w,default_min↩ _h*	default size range for widgets that don't have an individual range assigned

13.139.3.5 move_intersection()

```
void Fl_Tile::move_intersection (
            int oldx,
            int oldy,
            int newx,
            int newy )  [virtual]
```

Drags the intersection at (`oldx`,`oldy`) to (`newx`,`newy`).
This redraws all the necessary children.
If no size ranges are set, the new intersection position is limited to the size of the tile group. The resizable() option is not taken into account here.
If size ranges are set, the actual new position of the intersection will depend on the size range of every individual child. No child will be smaller than their minw and minh. After the new position is found, move_intersection() will call init_sizes(). The resizable() range is ignored.

Parameters

in	*oldx,oldy*	move the intersection at this coordinate, pass zero to disable drag in that direction.
in	*newx,newy*	move the intersection as close to this new coordinate as possible

13.139.3.6 on_insert()

```
int Fl_Tile::on_insert (
            Fl_Widget * candidate,
            int index )  [override], [protected], [virtual]
```

Insert a new entry in the size range list.
Reimplemented from Fl_Group.

13.139.3.7 on_move()

```
int Fl_Tile::on_move (
            int oldIndex,
            int newIndex )  [override], [protected], [virtual]
```

Move the entry in the size range list.
Reimplemented from Fl_Group.

13.139.3.8 on_remove()

```
void Fl_Tile::on_remove (
            int index )  [override], [protected], [virtual]
```

Remove the entry from the size range list.
Reimplemented from Fl_Group.

13.139.3.9 position()

```
void Fl_Tile::position (
            int oldx,
            int oldy,
            int newx,
            int newy )  [inline]
```

Deprecated "since 1.4.0 - use move_intersection(p) instead"

13.139.3.10 request_grow_b()

```
void Fl_Tile::request_grow_b (
            int old_b,
            int & new_b,
            Fl_Rect * final_size )  [protected]
```

Request for children to change their layout.

Parameters

in	*old_b*	grow all children with this current bottom edge toward the bottom edge of this tile
in,out	*new_b*	try to grow to this coordinate, return the maximum possible growth (currently maxh is ignored, so we always grow to new_b)
in,out	*final_size*	write the new position and size of all affected children into this list of Fl_Rect

13.139.3.11 request_grow_l()

```
void Fl_Tile::request_grow_l (
            int old_l,
            int & new_l,
            Fl_Rect * final_size )  [protected]
```

Request for children to change their layout.

Parameters

in	*old_l*	grow all children with this current left edge toward the left edge of this tile
in,out	*new_l*	try to grow to this coordinate, return the maximum possible growth (currently maxw is ignored, so we always grow to new_l)
in,out	*final_size*	write the new position and size of all affected children into this list of Fl_Rect

13.139.3.12 request_grow_r()

```
void Fl_Tile::request_grow_r (
            int old_r,
            int & new_r,
            Fl_Rect * final_size )  [protected]
```

Request for children to change their layout.

Parameters

in	*old_r*	grow all children with this current right edge toward the right edge of this tile
in,out	*new_r*	try to grow to this coordinate, return the maximum possible growth (currently maxw is ignored, so we always grow to new_r)
in,out	*final_size*	write the new position and size of all affected children into this list of Fl_Rect

13.139.3.13 request_grow_t()

```
void Fl_Tile::request_grow_t (
            int old_t,
            int & new_t,
```

```
            Fl_Rect * final_size )  [protected]
```

Request for children to change their layout.

Parameters

in	*old_t*	grow all children with this current top edge toward the top edge of this tile
in,out	*new_t*	try to grow to this coordinate, return the maximum possible growth (currently maxh is ignored, so we always grow to new_t)
in,out	*final_size*	write the new position and size of all affected children into this list of Fl_Rect

13.139.3.14 request_shrink_b()

```
void Fl_Tile::request_shrink_b (
            int old_b,
            int & new_b,
            Fl_Rect * final_size )  [protected]
```

Request for children to change their layout.

See also

Fl_Tile::request_shrink_l(int old_l, int &new_l, Fl_Rect *final_size)

Parameters

in	*old_b*	shrink all children with this current bottoom edge toward the top edge of this tile
in,out	*new_b*	try to shrink to this coordinate, return the maximum possible shrinkage
in,out	*final_size*	if not NULL, write the new position and size of all affected children into this list of Fl_Rect

13.139.3.15 request_shrink_l()

```
void Fl_Tile::request_shrink_l (
            int old_l,
            int & new_l,
            Fl_Rect * final_size )  [protected]
```

Request for children to change their layout.
drag_intersection requests that all children with the left edge at old_l to shrink to new_l towards the right side of the tile. If the child can not shrink by that amount, it will ask all other children that touch its right side to shrink by the remainder (recursion). new_l will return the the maximum possible value while maintaining minimum width for all children involved.
request_shrink_r asks children to shrink toward the left, so that their right edge is as close as possible to new_r. request_shrink_t and request_shrink_b provide the same functionality for vertical resizing.

Parameters

in	*old_l*	shrink all children with this current left edge
in,out	*new_l*	try to shrink to this coordinate, return the maximum possible shrinkage
in,out	*final_size*	if not NULL, write the new position and size of all affected children into this list of Fl_Rect

13.139.3.16 request_shrink_r()

```
void Fl_Tile::request_shrink_r (
            int old_r,
            int & new_r,
            Fl_Rect * final_size )  [protected]
```

Request for children to change their layout.

See also

Fl_Tile::request_shrink_l(int old_l, int &new_l, Fl_Rect *final_size)

Parameters

in	*old_r*	shrink all children with this current right edge toward the left edge of this tile
in,out	*new_r*	try to shrink to this coordinate, return the maximum possible shrinkage
in,out	*final_size*	if not NULL, write the new position and size of all affected children into this list of Fl_Rect

13.139.3.17 request_shrink_t()

```
void Fl_Tile::request_shrink_t (
            int old_t,
            int & new_t,
            Fl_Rect * final_size )  [protected]
```

Request for children to change their layout.

See also

Fl_Tile::request_shrink_l(int old_l, int &new_l, Fl_Rect *final_size)

Parameters

in	*old_t*	shrink all children with this current top edge toward the bottom edge of this tile
in,out	*new_t*	try to shrink to this coordinate, return the maximum possible shrinkage
in,out	*final_size*	if not NULL, write the new position and size of all affected children into this list of Fl_Rect

13.139.3.18 resize()

```
void Fl_Tile::resize (
            int X,
            int Y,
            int W,
            int H )  [override], [virtual]
```

Resizes the Fl_Tile widget and its children.
Fl_Tile implements its own resize() method. It does not use Fl_Group::resize() to resize itself and its children.
In size_range mode, the child marked resizable() is resized first. Only if its minimum size is reached, other widgets in the tile will resize too.
In classic mode or when no resizable child is set, enlarging works by moving the lower-right corner and resizing the bottom and right border widgets accordingly.
Shrinking the Fl_Tile works in the opposite way by shrinking the bottom and right border widgets, unless they are reduced to zero width or height, resp. or to their minimal sizes defined by the resizable() widget. In this case other widgets will be shrunk as well.
See the Fl_Tile class documentation about how the resizable() works.
Reimplemented from Fl_Group.

13.139.3.19 set_cursor()

```
void Fl_Tile::set_cursor (
            int n )  [protected]
```

Set one of four cursors used for dragging etc...
Fl_Tile uses an array of four cursors that are set depending on user actions:

- 0: normal cursor
- 1: horizontal dragging
- 2: vertical dragging
- 3: dragging an intersection

This method sets the window cursor for the given index `n`.

13.139.3.20 size_range() [1/2]

```
void Fl_Tile::size_range (
            Fl_Widget * w,
            int minw,
            int minh,
            int maxw = 0x7FFFFFFF,
            int maxh = 0x7FFFFFFF )
```

Set the allowed size range for the give child widget.
Fl_Tile currently supports only the minimal width and height setting.

Parameters

in	*w*	set the range for this child widget
in	*minw,minh*	minimum width and height for that child
in	*maxw,maxh*	maximum size, defaults to infinite, currently ignored

13.139.3.21 size_range() [2/2]

```
void Fl_Tile::size_range (
            int index,
            int minw,
            int minh,
            int maxw = 0x7FFFFFFF,
            int maxh = 0x7FFFFFFF )
```

Set the allowed size range for the child at the given index.
Fl_Tile currently supports only the minimal width and height setting.

Parameters

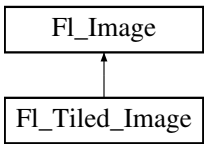
in	*index*	set the range for the child at this index
in	*minw,minh*	minimum width and height for that child
in	*maxw,maxh*	maximum size, defaults to infinite, currently ignored

The documentation for this class was generated from the following files:

- Fl_Tile.H
- Fl_Tile.cxx

13.140 Fl_Tiled_Image Class Reference

This class supports tiling of images over a specified area.
`#include <Fl_Tiled_Image.H>`
Inheritance diagram for Fl_Tiled_Image:



Public Member Functions

- void color_average (Fl_Color c, float i) override

  The color_average() method averages the colors in the image with the provided FLTK color value.
- Fl_Image ∗ **copy** () const
- Fl_Image ∗ copy (int W, int H) const override

  Creates a resized copy of the image.
- void desaturate () override

  The desaturate() method converts an image to grayscale.
- void **draw** (int X, int Y)
- void draw (int X, int Y, int W, int H, int cx=0, int cy=0) override

  Draws a tiled image.
- Fl_Tiled_Image (Fl_Image ∗i, int W=0, int H=0)

  The constructors create a new tiled image containing the specified image.
- Fl_Image ∗ **image** () const

  Gets The image that is tiled.
- virtual ∼**Fl_Tiled_Image** ()

  The destructor frees all memory and server resources that are used by the tiled image.

Public Member Functions inherited from Fl_Image

- virtual class Fl_Shared_Image ∗ as_shared_image ()

  Returns whether an image is an Fl_Shared_Image or not.
- Fl_Image ∗ copy () const

  Creates a copy of the image in the same size.
- int count () const

  Returns the number of data values associated with the image.
- int d () const

  Returns the image depth.
- const char ∗const ∗ data () const

  Returns a pointer to the current image data array.
- int **data_h** () const

  Returns the height of the image data.
- int **data_w** () const

  Returns the width of the image data.
- void draw (int X, int Y)

  Draws the image to the current drawing surface.
- int fail () const

  Returns a value that is not 0 if there is currently no image available.
- Fl_Image (int W, int H, int D)

  The constructor creates an empty image with the specified width, height, and depth.
- int h () const

  Returns the current image drawing height in FLTK units.
- void inactive ()

  The inactive() method calls color_average(FL_BACKGROUND_COLOR, 0.33f) to produce an image that appears grayed out.

- virtual void label (Fl_Menu_Item *m)

  This method is an obsolete way to set the image attribute of a menu item.
- virtual void label (Fl_Widget *w)

  This method is an obsolete way to set the image attribute of a widget or menu item.
- int ld () const

  Returns the current line data size in bytes.
- virtual void release ()

  Releases an Fl_Image - the same as '`delete this`'.
- virtual void scale (int width, int height, int proportional=1, int can_expand=0)

  Sets the drawing size of the image.
- virtual void uncache ()

  If the image has been cached for display, delete the cache data.
- int w () const

  Returns the current image drawing width in FLTK units.
- virtual ~**Fl_Image** ()

  The destructor is a virtual method that frees all memory used by the image.

Protected Attributes

- int **alloc_image_**
- Fl_Image * **image_**

Additional Inherited Members

Static Public Member Functions inherited from Fl_Image

- static Fl_Labeltype **define_FL_IMAGE_LABEL** ()
- static Fl_RGB_Scaling **RGB_scaling** ()

  Returns the currently used RGB image scaling method.
- static void RGB_scaling (Fl_RGB_Scaling)

  Sets the RGB image scaling method used for copy(int, int).
- static Fl_RGB_Scaling **scaling_algorithm** ()

  Gets what algorithm is used when resizing a source image to draw it.
- static void scaling_algorithm (Fl_RGB_Scaling algorithm)

  Sets what algorithm is used when resizing a source image to draw it.

Static Public Attributes inherited from Fl_Image

- static const int **ERR_FILE_ACCESS** = -2
- static const int **ERR_FORMAT** = -3
- static const int **ERR_MEMORY_ACCESS** = -4
- static const int **ERR_NO_IMAGE** = -1
- static bool **register_images_done** = false

  True after fl_register_images() was called, false before.

Protected Member Functions inherited from Fl_Image

- void **d** (int D)

  Sets the current image depth.
- void data (const char *const *p, int c)

  Sets the current data pointer and count of pointers in the array.
- void draw_empty (int X, int Y)

  The protected method draw_empty() draws a box with an X in it.
- int draw_scaled (int X, int Y, int W, int H)

Draw the image to the current drawing surface rescaled to a given width and height.

- void h (int H)

  Sets the height of the image data.

- void ld (int LD)

  Sets the current line data size in bytes.

- void w (int W)

  Sets the width of the image data.

Static Protected Member Functions inherited from Fl_Image

- static void **labeltype** (const Fl_Label *lo, int lx, int ly, int lw, int lh, Fl_Align la)
- static void **measure** (const Fl_Label *lo, int &lw, int &lh)

13.140.1 Detailed Description

This class supports tiling of images over a specified area.
The source (tile) image is **not** copied unless you call the color_average(), desaturate(), or inactive() methods.

13.140.2 Constructor & Destructor Documentation

13.140.2.1 Fl_Tiled_Image()

```
Fl_Tiled_Image::Fl_Tiled_Image (
            Fl_Image * i,
            int W = 0,
            int H = 0 )
```

The constructors create a new tiled image containing the specified image.
Use a width and height of 0 to tile the whole window/widget.

Note

Due to implementation constraints in FLTK 1.3.3 and later width and height of 0 may not work as expected when used as background image in widgets other than windows. You may need to center and clip the image (label) and set the label type to FL_NORMAL_LABEL. Doing so will let the tiled image fill the whole widget as its background image. Other combinations of label flags may or may not work.

```
#include "bg.xpm"
Fl_Pixmap *bg_xpm = new Fl_Pixmap(bg_xpm);
Fl_Tiled_Image *bg_tiled = new Fl_Tiled_Image(bg_xpm,0,0);

Fl_Box *box = new Fl_Box(40,40,300,100,"");
box->box(FL_UP_BOX);
box->labeltype(FL_NORMAL_LABEL);
box->align(FL_ALIGN_INSIDE | FL_ALIGN_CENTER | FL_ALIGN_CLIP);
box->image(bg_tiled);
```

Note

Setting an image (label) for a window may not work as expected due to implementation constraints in FLTK 1.3.x and maybe later. The reason is the way Fl::scheme() initializes the window's label type and image. A possible workaround is to use another Fl_Group as the only child widget and to set the background image for this group as described above.

Todo Fix Fl_Tiled_Image as background image for widgets and windows and fix the implementation of Fl::scheme(const char *).

13.140.3 Member Function Documentation

13.140.3.1 color_average()

```
void Fl_Tiled_Image::color_average (
            Fl_Color c,
            float i )  [override], [virtual]
```

The color_average() method averages the colors in the image with the provided FLTK color value.
The first argument specifies the FLTK color to be used.
The second argument specifies the amount of the original image to combine with the color, so a value of 1.0 results in no color blend, and a value of 0.0 results in a constant image of the specified color.
An internal copy is made of the original image data before changes are applied, to avoid modifying the original image data in memory.
Reimplemented from Fl_Image.

13.140.3.2 copy()

```
Fl_Image * Fl_Tiled_Image::copy (
            int W,
            int H ) const  [override], [virtual]
```

Creates a resized copy of the image.
It is recommended not to call this member function to reduce the size of an image to the size of the area where this image will be drawn, and to use Fl_Image::scale() instead.
The new image should be released when you are done with it.
Note: since FLTK 1.4.0 you can use Fl_Image::release() for all types of images (i.e. all subclasses of Fl_Image) instead of operator *delete* for Fl_Image's and Fl_Image::release() for Fl_Shared_Image's.
The new image data will be converted to the requested size. RGB images are resized using the algorithm set by Fl_Image::RGB_scaling().
For the new image the following equations are true:

- w() == data_w() == `W`
- h() == data_h() == `H`

Parameters

in	*W,H*	Requested width and height of the new image

Note

The returned image can be safely cast to the same image type as that of the source image provided this type is one of Fl_RGB_Image, Fl_SVG_Image, Fl_Pixmap, Fl_Bitmap, Fl_Tiled_Image, Fl_Anim_GIF_Image and Fl_Shared_Image. Returned objects copied from images of other, derived, image classes belong to the parent class appearing in this list. For example, the copy of an Fl_GIF_Image is an object of class Fl_Pixmap.

Since FLTK 1.4.0 this method is 'const'. If you derive your own class from Fl_Image or any subclass your overridden methods of 'Fl_Image::copy() const' and 'Fl_Image::copy(int, int) const' **must** also be 'const' for inheritance to work properly. This is different than in FLTK 1.3.x and earlier where these methods have not been 'const'.

Reimplemented from Fl_Image.

13.140.3.3 desaturate()

```
void Fl_Tiled_Image::desaturate ( )  [override], [virtual]
```

The desaturate() method converts an image to grayscale.
If the image contains an alpha channel (depth = 4), the alpha channel is preserved.
An internal copy is made of the original image data before changes are applied, to avoid modifying the original image data in memory.
Reimplemented from Fl_Image.

13.140.3.4 draw()

```
void Fl_Tiled_Image::draw (
            int X,
            int Y,
            int W,
```

```
int H,
int cx = 0,
int cy = 0 )  [override], [virtual]
```

Draws a tiled image.

Tiled images can be used as background images for widgets and windows. However, due to implementation constraints, you must take care when setting label types and alignment flags. Only certain combinations work as expected, others may yield unexpected results and undefined behavior.

This draw method can draw multiple copies of one image in an area given by `X`, `Y`, `W`, `H`.

The optional arguments `cx` and `cy` can be used to crop the image starting at offsets (cx, cy). `cx` and `cy` must be >= 0 (negative values are ignored). If one of the values is greater than the image width or height resp. (`cx` >= image()->w() or `cy` >= image()->h()) nothing is drawn, because the resulting image would be empty.

After calculating the resulting image size the image is drawn as often as necessary to fill the given area, starting at the top left corner.

If both `W` and `H` are 0 the image is repeated as often as necessary to fill the entire window, unless there is a valid clip region. If you want to fill only one particular widget's background, then you should either set a clip region in your draw() method or use the label alignment flags `FL_ALIGN_INSIDE|FL_ALIGN_CLIP` to make sure the image is clipped.

This may be improved in a later version of the library.

Reimplemented from Fl_Image.

The documentation for this class was generated from the following files:

- Fl_Tiled_Image.H
- Fl_Tiled_Image.cxx

13.141 Fl_Timeout Class Reference

The internal class Fl_Timeout handles all timeout related functions.

`#include <Fl_Timeout.h>`

Static Public Member Functions

- static void add_timeout (double time, Fl_Timeout_Handler cb, void *data)

  Adds a one-shot timeout callback.
- static void **do_timeouts** ()

  Elapse timers and call their callbacks if any timers are expired.
- static void elapse_timeouts ()

  Elapse all timers w/o calling their callbacks.
- static int has_timeout (Fl_Timeout_Handler cb, void *data)

  Returns true if the timeout exists and has not been called yet.
- static int remove_next_timeout (Fl_Timeout_Handler cb, void *data=NULL, void **data_return=NULL)

  Remove the next matching timeout callback and return its `data` pointer.
- static void remove_timeout (Fl_Timeout_Handler cb, void *data)

  Remove a timeout callback.
- static void repeat_timeout (double time, Fl_Timeout_Handler cb, void *data)

  Repeats a timeout callback from the expiration of the previous timeout, allowing for more accurate timing.
- static double time_to_wait (double ttw)

  Returns the delay in seconds until the next timer expires, limited by `ttw`.
- static std::vector< Fl::TimeoutData > **timeout_list** ()

Protected Member Functions

- double **delay** ()

  Get the timer's delay in seconds.
- void **delay** (double t)

  Set the timer's delay in seconds.

- void insert ()

  Insert this timer entry into the active timer queue.
- void make_current ()

  Remove the timeout from the active timer queue and push it onto the stack of currently running callbacks.
- void release ()

  Remove the top-most timeout from the stack of currently running timeout callbacks and insert it into the list of free timers.

Static Protected Member Functions

- static Fl_Timeout ∗ current ()

  Returns the first (top-most) timeout from the current timeout stack.
- static Fl_Timeout ∗ get (double time, Fl_Timeout_Handler cb, void ∗data)

  Get an Fl_Timeout instance for further handling.

Protected Attributes

- Fl_Timeout_Handler **callback**
- void ∗ **data**
- Fl_Timeout ∗ **next**
- int **skip**
- double **time**

Static Protected Attributes

- static Fl_Timeout ∗ current_timeout = 0

  The list of current timeouts is used to store the timeout whose callback is called while the callback is executed.
- static Fl_Timeout ∗ first_timeout = 0

  List of active timeouts.
- static Fl_Timeout ∗ free_timeout = 0

  List of free timeouts after use.

13.141.1 Detailed Description

The internal class Fl_Timeout handles all timeout related functions.
All code is platform independent except retrieving a timestamp which requires calling a system driver function and potentially results in different timer resolutions (from milliseconds to microseconds).
Related user documentation:

- Fl_Timeout_Handler
- Fl::add_timeout(double time, Fl_Timeout_Handler cb, void ∗data)
- Fl::repeat_timeout(double time, Fl_Timeout_Handler cb, void ∗data)
- Fl::has_timeout(Fl_Timeout_Handler cb, void ∗data)
- Fl::remove_timeout(Fl_Timeout_Handler cb, void ∗data)
- Fl::remove_next_timeout(Fl_Timeout_Handler cb, void ∗data, void ∗∗data_return)

13.141.2 Member Function Documentation

13.141.2.1 add_timeout()

```
void Fl_Timeout::add_timeout (
            double time,
            Fl_Timeout_Handler cb,
            void * data )  [static]
```

Adds a one-shot timeout callback.

The callback function `cb` will be called by Fl::wait() at `time` seconds after this function is called.

Parameters

in	*time*	delta time in seconds until the timer expires
in	*cb*	callback function
in	*data*	optional user data (default: NULL)

Implements:

`void Fl::add_timeout(double time, Fl_Timeout_Handler cb, void *data)`

See also

Fl::add_timeout(double time, Fl_Timeout_Handler cb, void *data)

13.141.2.2 current()

`Fl_Timeout * Fl_Timeout::current ( ) [static], [protected]`

Returns the first (top-most) timeout from the current timeout stack.

This returns a pointer to the timeout but does not remove it from the list of current timeouts. This should be the timeout that is currently executing its callback.

Returns

Fl_Timeout* The current timeout whose callback is running.

Return values

NULL	if no callback is currently running.

13.141.2.3 elapse_timeouts()

`void Fl_Timeout::elapse_timeouts ( ) [static]`

Elapse all timers w/o calling their callbacks.

All timer values are adjusted by the delta time since the last call. This method does **NOT** call timer callbacks if timers are expired.

This must be called before new timers are added to the timer queue to make sure that the next timer decrement does not count down too much time.

See also

Fl_Timeout::do_timeouts()

13.141.2.4 get()

```
Fl_Timeout * Fl_Timeout::get (
            double time,
            Fl_Timeout_Handler cb,
            void * data ) [static], [protected]
```

Get an Fl_Timeout instance for further handling.

The timer object will be initialized with the input parameters as given by Fl::add_timeout() or Fl::repeat_timeout().

Fl_Timeout objects are maintained in three queues:

- active timer queue
- list (stack, i.e. LIFO) of currently executing timer callbacks
- free timer entries.

When the FLTK program is launched all queues are empty. Whenever a new timer object is required the get() method is called and a timer object is either found in the queue of free timer entries or a new timer object is created (operator new).
Active timer entries are inserted into the "active timer queue" until they expire and their callback is called.
Before the callback is called the timer entry is inserted into the list of current timers, i.e. it becomes the Fl_Timeout::current() timeout. This can be used in Fl::repeat_timeout() to find out if and how long the current timeout has been delayed.
When a timer is no longer used it is popped from the `current` list and inserted into the "free timer" list so it can be reused later.
Timer queue entries are never returned to the system, there's no garbage collection. The total number of timer objects is determined by the largest number of concurrently active timers.

Parameters

in	*time*	requested delta time
in	*cb*	timer callback
in	*data*	userdata for timer callback

Returns

Fl_Timeout∗ Timer entry

See also

Fl::add_timeout(), Fl::repeat_timeout()

13.141.2.5 has_timeout()

```
int Fl_Timeout::has_timeout (
            Fl_Timeout_Handler cb,
            void * data )  [static]
```

Returns true if the timeout exists and has not been called yet.

Parameters

in	*cb*	Timer callback (must match)
in	*data*	Callback user data (must match)

Returns

whether the timer was found in the queue

Return values

0	not found
1	found

Implements:

```
int Fl::has_timeout(Fl_Timeout_Handler cb, void *data)
```

See also

Fl::has_timeout(Fl_Timeout_Handler cb, void ∗data)

13.141.2.6 insert()

```
void Fl_Timeout::insert ( ) [protected]
```

Insert this timer entry into the active timer queue.
The timer is inserted at the required position so the timer queue is always ordered by due time.

13.141.2.7 make_current()

```
void Fl_Timeout::make_current ( ) [protected]
```

Remove the timeout from the active timer queue and push it onto the stack of currently running callbacks.
This becomes the current() timeout which can be used in Fl::repeat_timeout().

See also

Fl_Timeout::current()

13.141.2.8 release()

```
void Fl_Timeout::release ( ) [protected]
```

Remove the top-most timeout from the stack of currently running timeout callbacks and insert it into the list of free timers.
Typical code in the library would look like:

```
// The timeout \p Fl_Timeout *t has exired, run its callback
t->make_current();
(t->callback)(t->data);
t->release();
```

13.141.2.9 remove_next_timeout()

```
int Fl_Timeout::remove_next_timeout (
            Fl_Timeout_Handler cb,
            void * data = NULL,
            void ** data_return = NULL ) [static]
```

Remove the next matching timeout callback and return its `data` pointer.
Implements:

```
int Fl::remove_next_timeout(Fl_Timeout_Handler cb, void *data, void **data_return)
```

Parameters

in	*cb*	Timer callback to be removed (must match)
in	*data*	Wildcard if NULL, must match otherwise
in,out	*data_return*	pointer to void * to receive the data value

Returns

non-zero if a timer was found and removed

Return values

0	no matching timer was found
1	the last matching timeout was found and removed
N>1	a matching timeout was removed and there are (N - 1) matching timeouts pending

For details

See also

Fl::remove_next_timeout(Fl_Timeout_Handler cb, void *data, void **data_return)

13.141.2.10 remove_timeout()

```
void Fl_Timeout::remove_timeout (
            Fl_Timeout_Handler cb,
            void * data )  [static]
```

Remove a timeout callback.
This method removes all matching timeouts, not just the first one. This may change in the future.

Parameters

in	*cb*	Timer callback to be removed (must match)
in	*data*	Wildcard if NULL, must match otherwise

Implements:

```
void Fl::remove_timeout(Fl_Timeout_Handler cb, void *data)
```

See also

Fl::remove_timeout(Fl_Timeout_Handler cb, void *data)

13.141.2.11 repeat_timeout()

```
void Fl_Timeout::repeat_timeout (
            double time,
            Fl_Timeout_Handler cb,
            void * data )  [static]
```

Repeats a timeout callback from the expiration of the previous timeout, allowing for more accurate timing.

Parameters

in	*time*	delta time in seconds until the timer expires
in	*cb*	callback function
in	*data*	optional user data (default: `NULL`)

Implements:

```
void Fl::repeat_timeout(double time, Fl_Timeout_Handler cb, void *data)
```

See also

Fl::repeat_timeout(double time, Fl_Timeout_Handler cb, void *data)

13.141.2.12 time_to_wait()

```
double Fl_Timeout::time_to_wait (
            double ttw )  [static]
```

Returns the delay in seconds until the next timer expires, limited by `ttw`.
This function calculates the time to wait for the FLTK event queue processing, depending on the given value `ttw`.
If at least one timer is active and its timeout value is smaller than `ttw` then this value is returned. Fl::wait() will wait no longer than until the next timer expires.
If no timer is active this returns the input value `ttw` unchanged.
If at least one timer is expired this returns 0.0 so the event processing does not wait.

Parameters

in	*ttw*	time to wait from Fl::wait() etc. (upper limit)

Returns

delay until next timeout or 0.0 (see description)

13.141.3 Member Data Documentation

13.141.3.1 current_timeout

`Fl_Timeout * Fl_Timeout::current_timeout = 0 [static], [protected]`

The list of current timeouts is used to store the timeout whose callback is called while the callback is executed. This is used like a stack, the current timeout is pushed to the front of the list and once the callback is finished, that timeout is removed and entered into the free list.
Background: Fl::repeat_timeout() needs to know which timeout triggered it and the exact schedule time and/or the delay of that timeout, i.e. how long the scheduled time was missed before the callback was called. A static, global variable is not sufficient since the user code can call other functions, e.g. dialogs, that run a nested event loop which can run another timeout callback. Hence this list of "current" timeouts is used like a stack (last in, first out).

See also

Fl_Timeout::push() Member function (method)

13.141.3.2 first_timeout

`Fl_Timeout * Fl_Timeout::first_timeout = 0 [static], [protected]`

List of active timeouts.
These timeouts can be triggered when due, which calls their callbacks. The lifetime of a timeout:

- active, in this queue
- callback running, in queue `current_timeout`
- done, in list of free timeouts, ready to be reused.

13.141.3.3 free_timeout

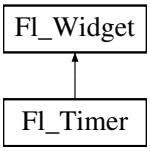
`Fl_Timeout * Fl_Timeout::free_timeout = 0 [static], [protected]`

List of free timeouts after use.
Timeouts can be reused many times.
The documentation for this class was generated from the following files:

- Fl_Timeout.h
- Fl_Timeout.cxx

13.142 Fl_Timer Class Reference

This is provided only to emulate the Forms Timer widget.
`#include <Fl_Timer.H>`
Inheritance diagram for Fl_Timer:



Public Member Functions

- char direction () const

  Gets or sets the direction of the timer.
- void direction (char d)

  Gets or sets the direction of the timer.
- Fl_Timer (uchar t, int x, int y, int w, int h, const char *l)

  Creates a new Fl_Timer widget using the given type, position, size, and label string.
- int handle (int) override

  Handles the specified event.
- char suspended () const

  Gets or sets whether the timer is suspended.
- void **suspended** (char d)

  Gets or sets whether the timer is suspended.
- double **value** () const

  See void Fl_Timer::value(double)
- void **value** (double)

  Sets the current timer value.
- ~**Fl_Timer** ()

  Destroys the timer and removes the timeout.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.

- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

Shortcut to set the label text and type in one call.

- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.

- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

Returns whether a widget and all its parents are visible.

- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void draw () override

  Draws the widget.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

Internal use only.

- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.142.1 Detailed Description

This is provided only to emulate the Forms Timer widget.
It works by making a timeout callback every 1/5 second. This is wasteful and inaccurate if you just want something to happen a fixed time in the future. You should directly call Fl::add_timeout() instead.

13.142.2 Constructor & Destructor Documentation

13.142.2.1 Fl_Timer()

```
Fl_Timer::Fl_Timer (
            uchar t,
            int X,
            int Y,
            int W,
            int H,
            const char * l )
```

Creates a new Fl_Timer widget using the given type, position, size, and label string.
The type parameter can be any of the following symbolic constants:

- FL_NORMAL_TIMER - The timer just does the callback and displays the string "Timer" in the widget.
- FL_VALUE_TIMER - The timer does the callback and displays the current timer value in the widget.
- FL_HIDDEN_TIMER - The timer just does the callback and does not display anything.

13.142.3 Member Function Documentation

13.142.3.1 direction() [1/2]

```
char Fl_Timer::direction ( ) const  [inline]
```

Gets or sets the direction of the timer.
If the direction is zero then the timer will count up, otherwise it will count down from the initial value().

13.142.3.2 direction() [2/2]

```
void Fl_Timer::direction (
            char d )  [inline]
```

Gets or sets the direction of the timer.
If the direction is zero then the timer will count up, otherwise it will count down from the initial value().

13.142.3.3 draw()

```
void Fl_Timer::draw ( )  [override], [protected], [virtual]
```

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();                 // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

13.142.3.4 handle()

```
int Fl_Timer::handle (
            int event )  [override], [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to *override* the behavior of the base class. This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to *filter* event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you *consumed* the event by returning 1 even if you didn't *do* anything with it.

Parameters

in	*event*	the kind of event received

Return values

0	if the event was not used or understood
1	if the event was used and can be deleted

See also

Fl_Event

Reimplemented from Fl_Widget.

13.142.3.5 suspended()

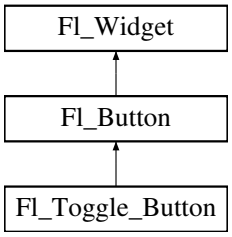
`char Fl_Timer::suspended ( ) const [inline]`
Gets or sets whether the timer is suspended.

The documentation for this class was generated from the following files:

- Fl_Timer.H
- forms_timer.cxx

13.143 Fl_Toggle_Button Class Reference

The toggle button is a push button that needs to be clicked once to toggle on, and one more time to toggle off.
`#include <Fl_Toggle_Button.H>`
Inheritance diagram for Fl_Toggle_Button:



Public Member Functions

- Fl_Toggle_Button (int X, int Y, int W, int H, const char *l=0)

  Creates a new Fl_Toggle_Button widget using the given position, size, and label string.

Public Member Functions inherited from Fl_Button

- int clear ()

  Same as `value(0)`.
- uchar compact () const

  Return true if buttons are rendered as compact buttons.
- void compact (uchar v)

  Decide if buttons should be rendered in compact mode.
- Fl_Boxtype down_box () const

  Returns the current down box type, which is drawn when value() is non-zero.
- void down_box (Fl_Boxtype b)

  Sets the down box type.
- Fl_Color **down_color** () const

  (for backwards compatibility)
- void **down_color** (unsigned c)

  (for backwards compatibility)
- Fl_Button (int X, int Y, int W, int H, const char *L=0)

  The constructor creates the button using the given position, size, and label.
- int handle (int) override

  Handles the specified event.
- int set ()

  Same as `value(1)`.
- void **setonly** ()

  Turns on this button and turns off all other radio buttons in the group (calling `value(1)` or `set()` does not do this).

- int shortcut () const

  Returns the current shortcut key for the button.
- void **shortcut** (const char *s)

  (for backwards compatibility)
- void shortcut (int s)

  Sets the shortcut key to `s`.
- char **value** () const

  Returns the current value of the button (0 or 1).
- int value (int v)

  Sets the current value of the button.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

Gets the current callback function for the widget.

- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.

- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

Gets the font to use.

- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.

- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

Gets the widget position in its window.

- int y () const

  Gets the widget position in its window.

- virtual ~Fl_Widget ()

  Destroys the widget.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.

- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.

- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions inherited from Fl_Button

- void draw () override

  Draws the widget.

- void **simulate_key_action** ()

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.

- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.

- void **draw_box** () const

  Draws the widget box according to its box style.

- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.

- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.

- void draw_focus () const

  Draws a focus rectangle around the widget.

- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Static Protected Member Functions inherited from Fl_Button

- static void **key_release_timeout** (void *)

Static Protected Attributes inherited from Fl_Button

- static Fl_Widget_Tracker * **key_release_tracker** = 0

13.143.1 Detailed Description

The toggle button is a push button that needs to be clicked once to toggle on, and one more time to toggle off.
The Fl_Toggle_Button subclass displays the "on" state by drawing a pushed-in button.
Buttons generate callbacks when they are clicked by the user. You control exactly when and how by changing the values for type() and when().

13.143.2 Constructor & Destructor Documentation

13.143.2.1 Fl_Toggle_Button()

```
Fl_Toggle_Button::Fl_Toggle_Button (
            int X,
            int Y,
            int W,
            int H,
            const char * L = 0 )
```

Creates a new Fl_Toggle_Button widget using the given position, size, and label string.
The constructor creates the button using the given position, size, and label.
The inherited destructor deletes the toggle button.
The Button type() is set to FL_TOGGLE_BUTTON.

Parameters

in	*X,Y,W,H*	position and size of the widget
in	*L*	widget label, default is no label

The documentation for this class was generated from the following files:

- Fl_Toggle_Button.H
- Fl_Button.cxx

13.144 Fl_Tooltip Class Reference

The Fl_Tooltip class provides tooltip support for all FLTK widgets.
`#include <Fl_Tooltip.H>`

Static Public Member Functions

- static Fl_Color color ()

  Gets the background color for tooltips.
- static void color (Fl_Color c)

  Sets the background color for tooltips.
- static Fl_Widget * **current** ()

  Gets the current widget target.
- static void current (Fl_Widget *)

  Sets the current widget target.
- static Fl_Window * **current_window** (void)

  Returns the window that is used for tooltips.
- static float delay ()

  Gets the tooltip delay.
- static void delay (float f)

  Sets the tooltip delay.
- static void disable ()

  Same as enable(0), disables tooltips on all widgets.
- static void enable (int b=1)

  Enables tooltips on all widgets (or disables if b is false).
- static int enabled ()

  Returns non-zero if tooltips are enabled.
- static void enter_area (Fl_Widget *w, int X, int Y, int W, int H, const char *tip)

  You may be able to use this to provide tooltips for internal pieces of your widget.
- static Fl_Font font ()

  Gets the typeface for the tooltip text.
- static void font (Fl_Font i)

  Sets the typeface for the tooltip text.
- static float hidedelay ()

  Gets the time until an open tooltip hides again.
- static void hidedelay (float f)

  Sets the time until an open tooltip hides again.
- static float hoverdelay ()

  Gets the tooltip hover delay, the delay between tooltips.
- static void hoverdelay (float f)

  Sets the tooltip hover delay, the delay between tooltips.
- static int margin_height ()

  Gets the amount of extra space above and below the tooltip's text.
- static void margin_height (int v)

  Sets the amount of extra space above and below the tooltip's text.
- static int margin_width ()

  Gets the amount of extra space left/right of the tooltip's text.

- static void margin_width (int v)

  Sets the amount of extra space left/right of the tooltip's text.
- static int override_text (const char *new_text)

  Temporarily Override Tooltip Text during an FL_BEFORE_TOOLTIP event.
- static Fl_Fontsize size ()

  Gets the size of the tooltip text.
- static void size (Fl_Fontsize s)

  Sets the size of the tooltip text.
- static Fl_Color textcolor ()

  Gets the color of the text in the tooltip.
- static void textcolor (Fl_Color c)

  Sets the color of the text in the tooltip.
- static int wrap_width ()

  Gets the maximum width for tooltip's text before it word wraps.
- static void wrap_width (int v)

  Sets the maximum width for tooltip's text before it word wraps.

Static Public Attributes

- static void(* **enter**)(Fl_Widget *w) = nothing
- static void(* **exit**)(Fl_Widget *w) = nothing

Friends

- class **Fl_TooltipBox**
- void **Fl_Widget::copy_tooltip** (const char *)
- void **Fl_Widget::tooltip** (const char *)

13.144.1 Detailed Description

The Fl_Tooltip class provides tooltip support for all FLTK widgets.
It contains only static methods.

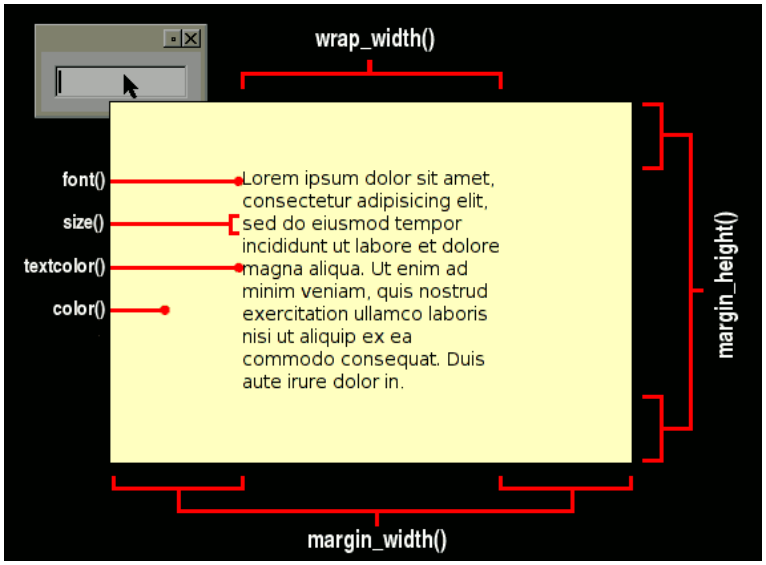


Figure 13.61 Fl_Tooltip Options

13.144.2 Member Function Documentation

13.144.2.1 color() [1/2]

```
static Fl_Color Fl_Tooltip::color ( ) [inline], [static]
```

Gets the background color for tooltips.
The default background color is a pale yellow.

13.144.2.2 color() [2/2]

```
static void Fl_Tooltip::color (
            Fl_Color c ) [inline], [static]
```

Sets the background color for tooltips.
The default background color is a pale yellow.

13.144.2.3 current()

```
void Fl_Tooltip::current (
            Fl_Widget * w ) [static]
```

Sets the current widget target.
Acts as though enter(widget) was done but does not pop up a tooltip. This is useful to prevent a tooltip from reappearing when a modal overlapping window is deleted. FLTK does this automatically when you click the mouse button.

13.144.2.4 delay() [1/2]

```
static float Fl_Tooltip::delay ( ) [inline], [static]
```

Gets the tooltip delay.
The default delay is 1.0 seconds.

13.144.2.5 delay() [2/2]

```
static void Fl_Tooltip::delay (
            float f ) [inline], [static]
```

Sets the tooltip delay.
The default delay is 1.0 seconds.

13.144.2.6 disable()

```
static void Fl_Tooltip::disable ( ) [inline], [static]
```

Same as enable(0), disables tooltips on all widgets.

13.144.2.7 enable()

```
static void Fl_Tooltip::enable (
            int b = 1 ) [inline], [static]
```

Enables tooltips on all widgets (or disables if *b* is false).

13.144.2.8 enabled()

```
static int Fl_Tooltip::enabled ( ) [inline], [static]
```

Returns non-zero if tooltips are enabled.

13.144.2.9 enter_area()

```
void Fl_Tooltip::enter_area (
            Fl_Widget * wid,
            int x,
            int y,
            int w,
            int h,
            const char * t )  [static]
```

You may be able to use this to provide tooltips for internal pieces of your widget.
Call this after setting Fl::belowmouse() to your widget (because that calls the above enter() method). Then figure out what thing the mouse is pointing at, and call this with the widget (this pointer is used to remove the tooltip if the widget is deleted or hidden, and to locate the tooltip), the rectangle surrounding the area, relative to the top-left corner of the widget (used to calculate where to put the tooltip), and the text of the tooltip (which must be a pointer to static data as it is not copied).

13.144.2.10 font() [1/2]

```
static Fl_Font Fl_Tooltip::font ( )  [inline], [static]
```

Gets the typeface for the tooltip text.

13.144.2.11 font() [2/2]

```
static void Fl_Tooltip::font (
            Fl_Font i )  [inline], [static]
```

Sets the typeface for the tooltip text.

13.144.2.12 hidedelay() [1/2]

```
static float Fl_Tooltip::hidedelay ( )  [inline], [static]
```

Gets the time until an open tooltip hides again.
The default delay is 12.0 seconds.

13.144.2.13 hidedelay() [2/2]

```
static void Fl_Tooltip::hidedelay (
            float f )  [inline], [static]
```

Sets the time until an open tooltip hides again.
The default delay is 12.0 seconds.

13.144.2.14 hoverdelay() [1/2]

```
static float Fl_Tooltip::hoverdelay ( )  [inline], [static]
```

Gets the tooltip hover delay, the delay between tooltips.
The default delay is 0.2 seconds.

13.144.2.15 hoverdelay() [2/2]

```
static void Fl_Tooltip::hoverdelay (
            float f )  [inline], [static]
```

Sets the tooltip hover delay, the delay between tooltips.
The default delay is 0.2 seconds.

13.144.2.16 margin_height() [1/2]

```
static int Fl_Tooltip::margin_height ( ) [inline], [static]
```

Gets the amount of extra space above and below the tooltip's text.
Default is 3.

13.144.2.17 margin_height() [2/2]

```
static void Fl_Tooltip::margin_height (
            int v ) [inline], [static]
```

Sets the amount of extra space above and below the tooltip's text.
Default is 3.

13.144.2.18 margin_width() [1/2]

```
static int Fl_Tooltip::margin_width ( ) [inline], [static]
```

Gets the amount of extra space left/right of the tooltip's text.
Default is 3.

13.144.2.19 margin_width() [2/2]

```
static void Fl_Tooltip::margin_width (
            int v ) [inline], [static]
```

Sets the amount of extra space left/right of the tooltip's text.
Default is 3.

13.144.2.20 override_text()

```
int Fl_Tooltip::override_text (
            const char * new_text ) [static]
```

Temporarily Override Tooltip Text during an FL_BEFORE_TOOLTIP event.
Use this method to temporarily change the tooltip text before it is displayed.
When FLTK sends an FL_BEFORE_TOOLTIP event to the widget under the mouse pointer, you can handle this event to modify the tooltip text dynamically. The provided text will be copied into a local buffer. To apply the override, the event handler must return 1.
To disable the tooltip for the current event, set the override text to nullptr or an empty string ("") and return 1.

Parameters

in	*new_text*	a C string that will be copied into a buffer

Returns

always 1, so this call can finish the FL_BEFORE_TOOLTIP event handling.

See also

void Fl_Widget::tooltip(const char *text).

`test/color_chooser.cxx` for a usage example.

13.144.2.21 size() [1/2]

```
static Fl_Fontsize Fl_Tooltip::size ( ) [inline], [static]
```

Gets the size of the tooltip text.

13.144.2.22 size() [2/2]

```
static void Fl_Tooltip::size (
            Fl_Fontsize s )  [inline], [static]
```

Sets the size of the tooltip text.

13.144.2.23 textcolor() [1/2]

```
static Fl_Color Fl_Tooltip::textcolor ( )  [inline], [static]
```

Gets the color of the text in the tooltip.
The default is black.

13.144.2.24 textcolor() [2/2]

```
static void Fl_Tooltip::textcolor (
            Fl_Color c )  [inline], [static]
```

Sets the color of the text in the tooltip.
The default is black.

13.144.2.25 wrap_width() [1/2]

```
static int Fl_Tooltip::wrap_width ( )  [inline], [static]
```

Gets the maximum width for tooltip's text before it word wraps.
Default is 400.

13.144.2.26 wrap_width() [2/2]

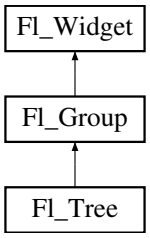
```
static void Fl_Tooltip::wrap_width (
            int v )  [inline], [static]
```

Sets the maximum width for tooltip's text before it word wraps.
Default is 400.
The documentation for this class was generated from the following files:

- Fl_Tooltip.H
- Fl.cxx
- Fl_Tooltip.cxx

13.145 Fl_Tree Class Reference

Tree widget.
`#include <Fl_Tree.H>`
Inheritance diagram for Fl_Tree:



Public Member Functions

- Fl_Tree_Item * add (const char *path, Fl_Tree_Item *newitem=0)

  Adds a new item, given a menu style `'path'`.

- Fl_Tree_Item * add (Fl_Tree_Item *parent_item, const char *name)

  Add a new child item labeled `'name'` to the specified `'parent_item'`.
- int auto_resize_children ()

  Returns whether auto resizing of children is active.
- void auto_resize_children (int mode)

  Set mode to resize children when the Fl_Tree container widget is resized.
- void calc_dimensions ()

  Recalculate widget dimensions and scrollbar visibility, normally managed automatically.
- void calc_tree ()

  Recalculates the tree's sizes and scrollbar visibility, normally managed automatically.
- Fl_Tree_Item * callback_item ()

  Gets the item that caused the callback.
- void callback_item (Fl_Tree_Item *item)

  Sets the item that was changed for this callback.
- Fl_Tree_Reason callback_reason () const

  Gets the reason for this callback.
- void callback_reason (Fl_Tree_Reason reason)

  Sets the reason for this callback.
- void clear ()

  Clear the entire tree's children, including the root.
- void clear_children (Fl_Tree_Item *item)

  Clear all the children for `'item'`.
- int close (const char *path, int docallback=1)

  Closes the item specified by `'path'`.
- int close (Fl_Tree_Item *item, int docallback=1)

  Closes the specified `'item'`.
- Fl_Image * closeicon () const

  Returns the icon to be used as the 'close' icon.
- void closeicon (Fl_Image *val)

  Sets the icon to be used as the 'close' icon.
- Fl_Color **connectorcolor** () const

  Get the connector color used for tree connection lines.
- void **connectorcolor** (Fl_Color val)

  Set the connector color used for tree connection lines.
- Fl_Tree_Connector **connectorstyle** () const

  Returns the line drawing style for inter-connecting items.
- void connectorstyle (Fl_Tree_Connector val)

  Sets the line drawing style for inter-connecting items.
- int **connectorwidth** () const

  Gets the width of the horizontal connection lines (in pixels) that appear to the left of each tree item's label.
- void **connectorwidth** (int val)

  Sets the width of the horizontal connection lines (in pixels) that appear to the left of each tree item's label.
- int deselect (const char *path, int docallback=1)

  Deselect an item specified by `'path'`.
- int deselect (Fl_Tree_Item *item, int docallback=1)

  Deselect the specified `item`.
- int deselect_all (Fl_Tree_Item *item=0, int docallback=1)

  Deselect `'item'` and all its children.
- void display (Fl_Tree_Item *item)

  Displays `'item'`, scrolling the tree as necessary.
- int displayed (Fl_Tree_Item *item)

See if 'item' is currently displayed on-screen (visible within the widget).

- int extend_selection (Fl_Tree_Item *from, Fl_Tree_Item *to, int val=1, bool visible=false)

  Extend a selection between 'from' and 'to' depending on 'visible'.
- int extend_selection_dir (Fl_Tree_Item *from, Fl_Tree_Item *to, int dir, int val, bool visible)

  Extend the selection between and including 'from' and 'to' depending on direction 'dir', 'val', and 'visible'.
- Fl_Tree_Item * **find_clicked** (int yonly=0)

  Non-const version of Fl_Tree::find_clicked(int yonly) const.
- const Fl_Tree_Item * find_clicked (int yonly=0) const

  Find the item that was last clicked on.
- Fl_Tree_Item * **find_item** (const char *path)

  *Non-const version of Fl_Tree::find_item(const char *path) const.*
- const Fl_Tree_Item * find_item (const char *path) const

  Find the item, given a menu style path, e.g.
- Fl_Tree_Item * first ()

  Returns the first item in the tree, or 0 if none.
- Fl_Tree_Item * first_selected_item ()

  Returns the first selected item in the tree.
- Fl_Tree_Item * first_visible ()

  Returns the first open(), visible item in the tree, or 0 if none.
- Fl_Tree_Item * first_visible_item ()

  Returns the first open(), visible item in the tree, or 0 if none.
- **Fl_Tree** (int X, int Y, int W, int H, const char *L=0)

  Constructor.
- Fl_Tree_Item * **get_item_focus** () const

  Get the item that currently has keyboard focus.
- int get_selected_items (Fl_Tree_Item_Array &items)

  Returns the currently selected items as an array of 'ret_items'.
- int handle (int e) override

  Standard FLTK event handler for this widget.
- int hposition () const

  Returns the horizontal scroll position as a pixel offset.
- void hposition (int pos)

  Sets the horizontal scroll offset to position 'pos'.
- Fl_Tree_Item * insert (Fl_Tree_Item *item, const char *name, int pos)

  Insert a new item 'name' into 'item's children at position 'pos'.
- Fl_Tree_Item * insert_above (Fl_Tree_Item *above, const char *name)

  Inserts a new item 'name' above the specified Fl_Tree_Item 'above'.
- int is_close (const char *path) const

  See if item specified by 'path' is closed.
- int is_close (Fl_Tree_Item *item) const

  See if the specified 'item' is closed.
- int is_hscroll_visible () const

  See if the horizontal scrollbar is currently visible.
- int is_open (const char *path) const

  See if item specified by 'path' is open.
- int is_open (Fl_Tree_Item *item) const

  See if 'item' is open.
- int is_scrollbar (Fl_Widget *w)

  See if widget 'w' is one of the Fl_Tree widget's scrollbars.
- int is_selected (const char *path)

See if item specified by `'path'` is selected.

- int is_selected (Fl_Tree_Item *item) const

  See if the specified `'item'` is selected.

- int is_vscroll_visible () const

  See if the vertical scrollbar is currently visible.

- Fl_Tree_Item * item_clicked ()

  Return the item that was last clicked.

- Fl_Tree_Item_Draw_Mode item_draw_mode () const

  Get the 'item draw mode' used for the tree.

- void item_draw_mode (Fl_Tree_Item_Draw_Mode mode)

  Set the 'item draw mode' used for the tree to `'mode'`.

- void item_draw_mode (int mode)

  Set the 'item draw mode' used for the tree to integer `'mode'`.

- void item_labelbgcolor (Fl_Color val)

  Set the default label background color used for creating new items.

- Fl_Color item_labelbgcolor (void) const

  Get the default label background color used for creating new items.

- void item_labelfgcolor (Fl_Color val)

  Set the default label foreground color used for creating new items.

- Fl_Color **item_labelfgcolor** (void) const

  Get the default label foreground color used for creating new items.

- Fl_Font **item_labelfont** () const

  Get the default font face used for creating new items.

- void item_labelfont (Fl_Font val)

  Set the default font face used for creating new items.

- Fl_Fontsize **item_labelsize** () const

  Get the default label fontsize used for creating new items.

- void item_labelsize (Fl_Fontsize val)

  Set the default label font size used for creating new items.

- int item_pathname (char *pathname, int pathnamelen, const Fl_Tree_Item *item) const

  Return `'pathname'` of size `'pathnamelen'` for the specified `'item'`.

- Fl_Tree_Item_Reselect_Mode item_reselect_mode () const

  Returns the current item re/selection mode.

- void item_reselect_mode (Fl_Tree_Item_Reselect_Mode mode)

  Sets the item re/selection mode.

- int **labelmarginleft** () const

  Get the amount of white space (in pixels) that should appear to the left of the label text.

- void **labelmarginleft** (int val)

  Set the amount of white space (in pixels) that should appear to the left of the label text.

- Fl_Tree_Item * last ()

  Returns the last item in the tree.

- Fl_Tree_Item * last_selected_item ()

  Returns the last selected item in the tree.

- Fl_Tree_Item * last_visible ()

  Returns the last open(), visible item in the tree.

- Fl_Tree_Item * last_visible_item ()

  Returns the last open(), visible item in the tree.

- int **linespacing** () const

  Get the amount of white space (in pixels) that should appear between items in the tree.

- void **linespacing** (int val)

  Sets the amount of white space (in pixels) that should appear between items in the tree.

- void load (class Fl_Preferences &)

  Load FLTK preferences.
- int **marginbottom** () const

  Get the amount of white space (in pixels) that should appear below the last visible item when the vertical scroller is scrolled to the bottom.
- void **marginbottom** (int val)

  Sets the amount of white space (in pixels) that should appear below the last visible item when the vertical scroller is scrolled to the bottom.
- int **marginleft** () const

  Get the amount of white space (in pixels) that should appear between the widget's left border and the tree's contents.
- void **marginleft** (int val)

  Set the amount of white space (in pixels) that should appear between the widget's left border and the left side of the tree's contents.
- int **margintop** () const

  Get the amount of white space (in pixels) that should appear between the widget's top border and the top of the tree's contents.
- void **margintop** (int val)

  Sets the amount of white space (in pixels) that should appear between the widget's top border and the top of the tree's contents.
- Fl_Tree_Item * next (Fl_Tree_Item *item=0)

  Return the next item after `'item'`, or 0 if no more items.
- Fl_Tree_Item * next_item (Fl_Tree_Item *item, int dir=FL_Down, bool visible=false)

  Returns next item after `'item'` in direction `'dir'` depending on `'visible'`.
- Fl_Tree_Item * next_selected_item (Fl_Tree_Item *item=0, int dir=FL_Down)

  Returns the next selected item above or below `'item'`, depending on `'dir'`.
- Fl_Tree_Item * next_visible_item (Fl_Tree_Item *start, int dir)

  Returns next open(), visible item above (`dir==FL_Up`) or below (`dir==FL_Down`) the specified `'item'`, or 0 if no more items.
- int open (const char *path, int docallback=1)

  Opens the item specified by `'path'`.
- int open (Fl_Tree_Item *item, int docallback=1)

  Open the specified `'item'`.
- void open_toggle (Fl_Tree_Item *item, int docallback=1)

  Toggle the open state of `'item'`.
- int **openchild_marginbottom** () const

  Get the amount of white space (in pixels) that should appear below an open child tree's contents.
- void **openchild_marginbottom** (int val)

  Set the amount of white space (in pixels) that should appear below an open child tree's contents.
- Fl_Image * openicon () const

  Returns the icon to be used as the 'open' icon.
- void openicon (Fl_Image *val)

  Sets the icon to be used as the 'open' icon.
- const Fl_Tree_Prefs & **prefs** () const
- Fl_Tree_Item * prev (Fl_Tree_Item *item=0)

  Return the previous item before `'item'`, or 0 if no more items.
- void recalc_tree ()

  Schedule tree to recalc the entire tree size.
- int remove (Fl_Tree_Item *item)

  Remove the specified `'item'` from the tree.
- void resize (int, int, int, int) override

  Resizes the Fl_Group widget and all of its children.
- Fl_Tree_Item * **root** ()

Returns the root item.

- void root (Fl_Tree_Item *newitem)

  Sets the root item to `'newitem'`.
- void root_label (const char *new_label)

  Set the label for the root item to `'new_label'`.
- int scrollbar_size () const

  Gets the default size of scrollbars' troughs for this widget in pixels.
- void scrollbar_size (int size)

  Sets the pixel size of the scrollbars' troughs to `'size'` for this widget, in pixels.
- int select (const char *path, int docallback=1)

  Select the item specified by `'path'`.
- int select (Fl_Tree_Item *item, int docallback=1)

  Select the specified `'item'`.
- int select_all (Fl_Tree_Item *item=0, int docallback=1)

  Select `'item'` and all its children.
- int select_only (Fl_Tree_Item *selitem, int docallback=1)

  Select only the specified item, deselecting all others that might be selected.
- void select_toggle (Fl_Tree_Item *item, int docallback=1)

  Toggle the select state of the specified `'item'`.
- Fl_Boxtype selectbox () const

  Sets the style of box used to draw selected items.
- void selectbox (Fl_Boxtype val)

  Gets the style of box used to draw selected items.
- Fl_Tree_Select selectmode () const

  Gets the tree's current selection mode.
- void selectmode (Fl_Tree_Select val)

  Sets the tree's selection mode.
- void set_item_focus (Fl_Tree_Item *item)

  Set the item that currently should have keyboard focus.
- void show_item (Fl_Tree_Item *item)

  Adjust the vertical scrollbar to show `'item'` at the top of the display IF it is currently off-screen (for instance show_item_top()).
- void show_item (Fl_Tree_Item *item, int yoff)

  Adjust the vertical scrollbar so that `'item'` is visible `'yoff'` pixels from the top of the Fl_Tree widget's display.
- void show_item_bottom (Fl_Tree_Item *item)

  Adjust the vertical scrollbar so that `'item'` is at the bottom of the display.
- void show_item_middle (Fl_Tree_Item *item)

  Adjust the vertical scrollbar so that `'item'` is in the middle of the display.
- void show_item_top (Fl_Tree_Item *item)

  Adjust the vertical scrollbar so that `'item'` is at the top of the display.
- void show_self ()

  Print the tree as 'ascii art' to stdout.
- int showcollapse () const

  Returns 1 if the collapse icon is enabled, 0 if not.
- void showcollapse (int val)

  Set if we should show the collapse icon or not.
- int **showroot** () const

  Returns 1 if the root item is to be shown, or 0 if not.
- void showroot (int val)

  Set if the root item should be shown or not.
- Fl_Tree_Sort sortorder () const

Set the default sort order used when items are added to the tree.

- void **sortorder** (Fl_Tree_Sort val)

  Gets the sort order used to add items to the tree.
- Fl_Image * usericon () const

  Returns the Fl_Image being used as the default user icon for all newly created items.
- void usericon (Fl_Image *val)

  Sets the Fl_Image to be used as the default user icon for all newly created items.
- int **usericonmarginleft** () const

  Get the amount of white space (in pixels) that should appear to the left of the usericon.
- void **usericonmarginleft** (int val)

  Set the amount of white space (in pixels) that should appear to the left of the usericon.
- int vposition () const

  Returns the vertical scroll position as a pixel offset.
- void vposition (int pos)

  Sets the vertical scroll offset to position `'pos'`.
- int **widgetmarginleft** () const

  Get the amount of white space (in pixels) that should appear to the left of the child fltk widget (if any).
- void **widgetmarginleft** (int val)

  Set the amount of white space (in pixels) that should appear to the left of the child fltk widget (if any).
- ~**Fl_Tree** ()

  Destructor.

Public Member Functions inherited from Fl_Group

- Fl_Widget *& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget *o)

  See void Fl_Group::add(Fl_Widget &w)
- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget *const * array () const

  Returns a pointer to the internal array of children.
- Fl_Group const * as_group () const override
- Fl_Group * as_group () override

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const

  Returns the n'th child.
- int **children** () const

  Returns how many child widgets the group has.
- void clear ()

  Deletes all child widgets from memory recursively.
- unsigned int clip_children () const

  Returns the current clipping mode.
- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)

  Removes the widget at `index` from the group and deletes it.

- void end ()

  Exactly the same as current(this->parent()).
- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*
- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.
- void **forms_end** ()

  This is for forms compatibility only.
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

  This does insert(w, find(before)).
- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.
- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.
- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

Sets the current label.

- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.

- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

Sends the specified event to the widget.

- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.

- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void **do_callback_for_item** (Fl_Tree_Item ∗item, Fl_Tree_Reason reason)

  Do the callback for the specified `'item'` using `'reason'`, setting the callback_item() and callback_reason().
- void draw () override

  Standard FLTK draw() method, handles drawing the tree widget.
- void item_clicked (Fl_Tree_Item ∗val)

  Set the item that was last clicked.

Protected Member Functions inherited from Fl_Group

- Fl_Rect ∗ bounds ()

  Returns the internal array of widget sizes and positions.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget ∗, int)

  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)

  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)

  Allow derived groups to act when a child widget is removed from the group.
- int ∗ sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Protected Attributes

- Fl_Scrollbar * **_hscroll**

  Horizontal scrollbar.
- int **_tih**

  Tree widget inner xywh dimension: inside borders + scrollbars.
- int **_tiw**
- int **_tix**
- int **_tiy**
- int **_toh**

  Tree widget outer xywh dimension: outside scrollbars, inside widget border.
- int **_tow**
- int **_tox**
- int **_toy**
- int **_tree_h**

the calculated height of the entire tree hierarchy. See calc_tree()

- int **_tree_w**

  the calculated width of the entire tree hierarchy. See calc_tree()

- Fl_Scrollbar * **_vscroll**

  Vertical scrollbar.

Friends

- class **Fl_Tree_Item**

Additional Inherited Members

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()

  Returns the currently active group.

- static void current (Fl_Group *g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.

- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.

- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.145.1 Detailed Description

Tree widget.

Figure 13.62 Fl_Tree example program

```
Fl_Tree                                          // Top level widget
   |--- Fl_Tree_Item                              // Items in the tree
   |--- Fl_Tree_Prefs                             // Preferences for the tree
            |--- Fl_Tree_Connector (enum)         // Connection modes
            |--- Fl_Tree_Select (enum)            // Selection modes
            |--- Fl_Tree_Sort (enum)              // Sort behavior
```

Similar to Fl_Browser, Fl_Tree is a browser of Fl_Tree_Item's arranged in a parented hierarchy, or 'tree'. Subtrees can be expanded or closed. Items can be added, deleted, inserted, sorted and re-ordered.
The tree items may also contain other FLTK widgets, like buttons, input fields, or even "custom" widgets.
The callback() is invoked depending on the value of when():

- FL_WHEN_RELEASE – callback invoked when left mouse button is released on an item
- FL_WHEN_CHANGED – callback invoked when left mouse changes selection state

The simple way to define a tree:

```
#include <FL/Fl_Tree.H>
[..]
Fl_Tree tree(X,Y,W,H);
tree.begin();
  tree.add("Flintstones/Fred");
  tree.add("Flintstones/Wilma");
  tree.add("Flintstones/Pebbles");
  tree.add("Simpsons/Homer");
  tree.add("Simpsons/Marge");
  tree.add("Simpsons/Bart");
  tree.add("Simpsons/Lisa");
tree.end();
```

FEATURES

Items can be added with add(),
removed with remove(),
completely cleared with clear(),
inserted with insert() and insert_above(),
selected/deselected with select() and deselect(),
open/closed with open() and close(),
positioned on the screen with show_item_top(), show_item_middle() and show_item_bottom(),
item children can be swapped around with Fl_Tree_Item::swap_children(),
items can be moved around with Fl_Tree_Item::move(),
an item's children can be walked with Fl_Tree_Item::first() and Fl_Tree_Item::next(), an item's children can be indexed directly with Fl_Tree_Item::child() and Fl_Tree_Item::children(),
items can be moved from one subtree to another with Fl_Tree_Item::deparent() and Fl_Tree_Item::reparent(),
sorting can be controlled when items are add()ed via sortorder().
You can walk the entire tree with first() and next().
You can walk visible items with first_visible_item() and next_visible_item().
You can walk selected items with first_selected_item() and next_selected_item().
Items can be found by their pathname using find_item(const char*), and an item's pathname can be found

with item_pathname().
The selected items' colors are controlled by selection_color() (inherited from Fl_Widget).
A hook is provided to allow you to redefine how item's labels are drawn via Fl_Tree::item_draw_callback().
Items can be interactively dragged using FL_TREE_SELECT_SINGLE_DRAGGABLE.

SELECTION OF ITEMS

The tree can have different selection behaviors controlled by selectmode(). The background color used for selected items is the Fl_Tree::selection_color(). The foreground color for selected items is controlled internally with fl_contrast().

CHILD WIDGETS

FLTK widgets (including custom widgets) can be assigned to tree items via Fl_Tree_Item::widget().

When an Fl_Tree_Item::widget() is defined, the default behavior is for the widget() to be shown in place of the item's label (if it has one). Only the widget()'s width will be used; the widget()'s x() and y() position will be managed by the tree, and the h() will track the item's height. This default behavior can be altered (ABI 1.3.1): Setting Fl_Tree::item_draw_mode()'s FL_TREE_ITEM_DRAW_LABEL_AND_WIDGET flag causes the label + widget to be displayed together in that order, and adding the FL_TREE_ITEM_HEIGHT_FROM_WIDGET flag causes widget's height to define the widget()'s height.

ICONS

The tree's open/close icons can be redefined with Fl_Tree::openicon(), Fl_Tree::closeicon(). User icons can either be changed globally with Fl_Tree::usericon(), or on a per-item basis with Fl_Tree_Item::usericon().

Various default preferences can be globally manipulated via Fl_Tree_Prefs, including colors, margins, icons, connection lines, etc.

FONTS AND COLORS

When adding new items to the tree, the new items get the defaults for fonts and colors from:

- Fl_Tree::item_labelfont() – The default item label font (default: FL_HELVETICA)
- Fl_Tree::item_labelsize() – The default item label size (default: FL_NORMAL_SIZE)
- Fl_Tree::item_labelfgcolor() – The default item label foreground color (default: FL_FOREGROUND_↩ COLOR)
- Fl_Tree::item_labelbgcolor() – The default item label background color (default: 0xffffffff, which tree uses as 'transparent')

Each item (Fl_Tree_Item) inherits a copy of these font/color attributes when created, and each item has its own methods to let the app change these values on a per-item basis using methods of the same name:

- Fl_Tree_Item::labelfont() – The item's label font (default: FL_HELVETICA)
- Fl_Tree_Item::labelsize() – The item's label size (default: FL_NORMAL_SIZE)
- Fl_Tree_Item::labelfgcolor() – The item's label foreground color (default: FL_FOREGROUND_COLOR)
- Fl_Tree_Item::labelbgcolor() – The item's label background color (default: 0xffffffff, which uses the tree's own bg color)

CALLBACKS

The tree's callback() will be invoked when items change state or are open/closed. when() controls when mouse/keyboard events invoke the callback. callback_item() and callback_reason() can be used to determine the cause of the callback. e.g.

```
void MyTreeCallback(Fl_Widget *w, void *data) {
  Fl_Tree      *tree = (Fl_Tree*)w;
  Fl_Tree_Item *item = (Fl_Tree_Item*)tree->callback_item();   // get selected item
  switch ( tree->callback_reason() ) {
    case FL_TREE_REASON_SELECTED: [..]
    case FL_TREE_REASON_DESELECTED: [..]
    case FL_TREE_REASON_RESELECTED: [..]
    case FL_TREE_REASON_OPENED: [..]
    case FL_TREE_REASON_CLOSED: [..]
  }
  :
}
```

SIMPLE EXAMPLES

To find all the selected items:

```
for ( Fl_Tree_Item *i=first_selected_item(); i; i=next_selected_item(i) )
  printf("Item %s is selected\n", i->label());
```

To get an item's full menu pathname, use Fl_Tree::item_pathname(), e.g.

```
[..]
char pathname[256] = "???";
tree->item_pathname(pathname, sizeof(pathname), item);         // eg. "Parent/Child/Item"
[..]
```

To walk all the items of the tree from top to bottom:

```
// Walk all the items in the tree, and print their labels
for ( Fl_Tree_Item *item = tree->first(); item; item = tree->next(item) ) {
    printf("Item: %s\n", item->label());
}
```

To recursively walk all the children of a particular item, define a function that uses recursion:

```
// Find all of the item's children and print an indented report of their labels
void my_print_all_children(Fl_Tree_Item *item, int indent=0) {
    for ( int t=0; t<item->children(); t++ ) {
        printf("%*s Item: %s\n", indent, "", item->child(t)->label());
        my_print_all_children(item->child(t), indent+4);   // recurse
    }
}
```

To change the default label font and color when creating new items:

```
tree = new Fl_Tree(..);
tree->item_labelfont(FL_COURIER);      // Use Courier font for all new items
tree->item_labelfgcolor(FL_RED);       // Use red color for labels of all new items
[..]
// Now create the items in the tree using the above defaults.
tree->add("Aaa");
tree->add("Bbb");
```

To change the font and color of all existing items in the tree:

```
// Change the font and color of all items currently in the tree
for ( Fl_Tree_Item *item = tree->first(); item; item = tree->next(item) ) {
    item->labelfont(FL_COURIER);
    item->labelcolor(FL_RED);
}
```

DISPLAY DESCRIPTION

The following image shows the tree's various visual elements and the methods that control them:

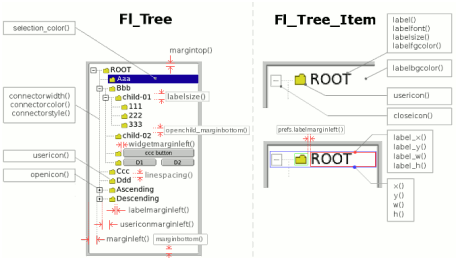


Figure 13.63 Fl_Tree elements

The following shows the protected dimension variables 'tree inner' (tix..) and 'tree outer' (tox..):

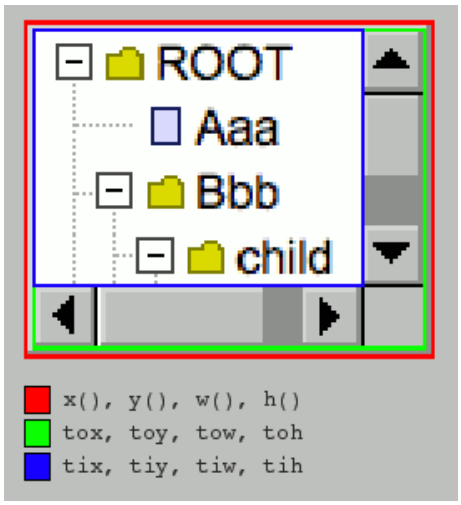


Figure 13.64 Fl_Tree inner/outer dimensions

KEYBOARD BINDINGS

The following table lists keyboard bindings for navigating the tree:

Keyboard	FL_TREE_SELECT↩ _MULTI	FL_TREE_SELECT↩ _SINGLE	FL_TREE_SELECT↩ _NONE

Keyboard	FL_TREE_SELECT↩ _MULTI	FL_TREE_SELECT↩ _SINGLE	FL_TREE_SELECT↩ _NONE
Ctrl-A (Linux/Windows)	Select all items	N/A	N/A
Command-A (Mac)	Select all items	N/A	N/A
Space	Selects item	Selects item	N/A
Ctrl-Space	Toggle item	Toggle item	N/A
Shift-Space	Extends selection	Selects item	N/A
Enter	Toggles open/close	Toggles open/close	Toggles open/close
Ctrl-Enter	Toggles open/close	Toggles open/close	Toggles open/close
Shift-Enter	Toggles open/close	Toggles open/close	Toggles open/close
Right / Left	Open/Close item	Open/Close item	Open/Close item
Up / Down	Move focus box up/down	Move focus box up/down	N/A
Shift-Up / Shift-Down	Extend selection up/down	Move focus up/down	N/A
Home / End	Move to top/bottom of tree	Move to top/bottom of tree	Move to top/bottom of tree
PageUp / PageDown	Page up/down	Page up/down	Page up/down

13.145.2 Member Function Documentation

13.145.2.1 add() [1/2]

```
Fl_Tree_Item * Fl_Tree::add (
            const char * path,
            Fl_Tree_Item * item = 0 )
```

Adds a new item, given a menu style `'path'`.
Any parent nodes that don't already exist are created automatically. Adds the item based on the value of sortorder().
If `'item'` is NULL, a new item is created.
To specify items or submenus that contain slashes ('/' or '\') use an escape character to protect them, e.g.

```
:
tree->add("/Holidays/Photos/12\\/25\\/2010");          // Adds item "12/25/2010"
tree->add("/Pathnames/c:\\\\Program Files\\\\MyApp"); // Adds item "c:\Program Files\MyApp"
:
```

Parameters

in	*path*	The path to the item, e.g. "Flintstone/Fred".
in	*item*	The new item to be added. If NULL, a new item is created with a name that is the last element in `'path'`.

Returns

The new item added, or 0 on error.

Version

1.3.3

13.145.2.2 add() [2/2]

```
Fl_Tree_Item * Fl_Tree::add (
            Fl_Tree_Item * parent_item,
            const char * name )
```

Add a new child item labeled 'name' to the specified 'parent_item'.

Parameters

in	*parent_item*	The parent item the new child item will be added to. Must not be NULL.
in	*name*	The label for the new item

Returns

The new item added.

Version

1.3.0 release

13.145.2.3 auto_resize_children() [1/2]

```
int Fl_Tree::auto_resize_children ( ) [inline]
```

Returns whether auto resizing of children is active.

Returns

0 (auto resizing **off**) or 1 (**on**)

Since

1.5.0

See also

void auto_resize_children(int mode) for details

13.145.2.4 auto_resize_children() [2/2]

```
void Fl_Tree::auto_resize_children (
            int mode ) [inline]
```

Set mode to resize children when the Fl_Tree container widget is resized.
The default value is `false`, meaning that child widgets within Fl_Tree widgets are not automatically resized.
This default behavior was changed in version 1.5.0 because resizing children automatically led to unpredictable behavior depending on their positions and sizes. This was caused by using the default resizing mechanism, Fl_Group::resize(), and was fixed in version 1.5.0 by disabling automatic resizing of children.
This method can be used to restore the 1.4.x behavior to resize children automatically for backwards compatibility by calling Fl_Tree::auto_resize_children(1). See example below.
Note: use with care and only if required for strict backwards compatibility. The resulting resize behavior can be *surprising*. If your development target is FLTK 1.4 or lower you can't use this method (you may include it inside conditional compilation for FLTK 1.5 and higher).
Example code for strict backwards compatibility to 1.4.x or lower:

```
#include <FL/Fl_Tree.H>
  Fl_Tree *tree = new Fl_Tree(0, 0, 400, 400);
#if FL_API_VERSION >= 10500
  tree->auto_resize_children(1);
#endif
```

Parameters

in	*mode*	enable auto resizing if true (non-zero)

Since

1.5.0

See also

int auto_resize_children()

13.145.2.5 calc_dimensions()

```
void Fl_Tree::calc_dimensions ( )
```

Recalculate widget dimensions and scrollbar visibility, normally managed automatically.
Low overhead way to update the tree widget's outer/inner dimensions and re-determine scrollbar visibility based on these changes without recalculating the entire size of the tree data.
Assumes that either the tree's size in _tree_w/_tree_h are correct so that scrollbar visibility can be calculated easily, or are both zero indicating scrollbar visibility can't be calculated yet.
This method is called when the widget is resize()ed or if the scrollbar's sizes are changed (affects tree widget's inner dimensions tix/y/w/h), and also used by calc_tree().

Version

1.3.3 ABI feature

13.145.2.6 calc_tree()

```
void Fl_Tree::calc_tree ( )
```

Recalculates the tree's sizes and scrollbar visibility, normally managed automatically.
On return:

- _tree_w will be the overall pixel width of the entire viewable tree
- _tree_h will be the overall pixel height ""
- scrollbar visibility and pan sizes are updated
- internal _tix/_tiy/_tiw/_tih dimensions are updated

_tree_w/_tree_h include the tree's margins (e.g. marginleft()), whether items are open or closed, label contents and font sizes, etc.
The tree hierarchy's size is managed separately from the widget's size as an optimization; this way resize() on the widget doesn't involve recalculating the tree's hierarchy needlessly, as widget size has no bearing on the tree hierarchy.
The tree hierarchy's size only changes when items are added/removed, open/closed, label contents or font sizes changed, margins changed, etc.
This calculation involves walking the *entire* tree from top to bottom, potentially a slow calculation if the tree has many items (potentially hundreds of thousands), and should therefore be called sparingly.
For this reason, recalc_tree() is used as a way to /schedule/ calculation when changes affect the tree hierarchy's size.
Apps may want to call this method directly if the app makes changes to the tree's geometry, then immediately needs to work with the tree's new dimensions before an actual redraw (and recalc) occurs. (This use by an app should only rarely be needed)

13.145.2.7 callback_item() [1/2]

```
Fl_Tree_Item * Fl_Tree::callback_item ( )
```

Gets the item that caused the callback.
The callback() can use this value to see which item changed.

13.145.2.8 callback_item() [2/2]

```
void Fl_Tree::callback_item (
            Fl_Tree_Item * item )
```

Sets the item that was changed for this callback.
Used internally to pass the item that invoked the callback.

13.145.2.9 callback_reason() [1/2]

```
Fl_Tree_Reason Fl_Tree::callback_reason ( ) const
```

Gets the reason for this callback.
The callback() can use this value to see why it was called. Example:

```
:
void MyTreeCallback(Fl_Widget *w, void *userdata) {
    Fl_Tree *tree = (Fl_Tree*)w;
    Fl_Tree_Item *item = tree->callback_item();    // the item changed (can be NULL if more than one
item was changed!)
    switch ( tree->callback_reason() ) {           // reason callback was invoked
        case     FL_TREE_REASON_OPENED: ..item was opened..
        case     FL_TREE_REASON_CLOSED: ..item was closed..
        case   FL_TREE_REASON_SELECTED: ..item was selected..
        case FL_TREE_REASON_RESELECTED: ..item was reselected (double-clicked, etc)..
        case FL_TREE_REASON_DESELECTED: ..item was deselected..
    }
}
:
```

See also

item_reselect_mode() – enables FL_TREE_REASON_RESELECTED events

13.145.2.10 callback_reason() [2/2]

```
void Fl_Tree::callback_reason (
            Fl_Tree_Reason reason )
```

Sets the reason for this callback.
Used internally to pass the reason the callback was invoked.

13.145.2.11 clear()

```
void Fl_Tree::clear (
            void  )
```

Clear the entire tree's children, including the root.
The tree will be left completely empty.

13.145.2.12 clear_children()

```
void Fl_Tree::clear_children (
            Fl_Tree_Item * item )
```

Clear all the children for `'item'`.
Item may not be NULL.

13.145.2.13 close() [1/2]

```
int Fl_Tree::close (
            const char * path,
            int docallback = 1 )
```

Closes the item specified by `'path'`.
Invokes the callback depending on the value of optional parameter `'docallback'`.
Handles calling redraw() if anything changed.
Items or submenus that themselves contain slashes ('/' or '\') should be escaped, e.g. close("Holidays/12\\/25\\/2010").
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

in	*path*	– the tree item's pathname (e.g. "Flintstones/Fred")

Parameters

in	*docallback*	– A flag that determines if the callback() is invoked or not: • 0 - callback() is not invoked • 1 - callback() is invoked if item changed (default), callback_reason() will be FL_TREE_REASON_CLOSED

Returns

- 1 – OK: item closed
- 0 – OK: item was already closed, no change
- -1 – ERROR: item was not found

See also

open(), close(), is_open(), is_close(), callback_item(), callback_reason()

13.145.2.14 close() [2/2]

```
int Fl_Tree::close (
            Fl_Tree_Item * item,
            int docallback = 1 )
```

Closes the specified `'item'`.
Invokes the callback depending on the value of optional parameter `'docallback'`.
Handles calling redraw() if anything changed.
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

in	*item*	– the item to be closed. Must not be NULL.
in	*docallback*	– A flag that determines if the callback() is invoked or not: • 0 - callback() is not invoked • 1 - callback() is invoked if item changed (default), callback_reason() will be FL_TREE_REASON_CLOSED

Returns

- 1 – item was closed
- 0 – item was already closed, no change

See also

open(), close(), is_open(), is_close(), callback_item(), callback_reason()

13.145.2.15 closeicon() [1/2]

```
Fl_Image * Fl_Tree::closeicon ( ) const
```

Returns the icon to be used as the 'close' icon.
If none was set, the internal default is returned, a simple '[-]' icon.

13.145.2.16 closeicon() [2/2]

```
void Fl_Tree::closeicon (
            Fl_Image * val )
```

Sets the icon to be used as the 'close' icon.
This overrides the built in default '[-]' icon.

Parameters

in	*val*	– The new image, or zero to use the default [-] icon.

13.145.2.17 connectorstyle()

```
void Fl_Tree::connectorstyle (
            Fl_Tree_Connector val )
```

Sets the line drawing style for inter-connecting items.
See Fl_Tree_Connector for possible values.

13.145.2.18 deselect() [1/2]

```
int Fl_Tree::deselect (
            const char * path,
            int docallback = 1 )
```

Deselect an item specified by `'path'`.
Invokes the callback depending on the value of optional parameter `'docallback'`.
Handles calling redraw() if anything changed.
Items or submenus that themselves contain slashes ('/' or '\') should be escaped, e.g. deselect("↩ Holidays/12\\/25\\/2010").
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

in	*path*	– the tree item's pathname (e.g. "Flintstones/Fred")
in	*docallback*	– A flag that determines if the callback() is invoked or not: • 0 - the callback() is not invoked • 1 - the callback() is invoked if item changed state (default), callback_reason() will be FL_TREE_REASON_DESELECTED

Returns

- 1 - OK: item's state was changed
- 0 - OK: item was already deselected, no change was made
- -1 - ERROR: item was not found

13.145.2.19 deselect() [2/2]

```
int Fl_Tree::deselect (
            Fl_Tree_Item * item,
            int docallback = 1 )
```

Deselect the specified `item`.
Invokes the callback depending on the value of optional parameter `'docallback'`.
Handles calling redraw() if anything changed.
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

in	*item*	– the item to be deselected. Must not be NULL.
in	*docallback*	– A flag that determines if the callback() is invoked or not: • 0 - the callback() is not invoked • 1 - the callback() is invoked if item changed state (default), callback_reason() will be FL_TREE_REASON_DESELECTED

Returns

- 0 - item was already deselected, no change was made
- 1 - item's state was changed

13.145.2.20 deselect_all()

```
int Fl_Tree::deselect_all (
            Fl_Tree_Item * item = 0,
            int docallback = 1 )
```

Deselect `'item'` and all its children.
If item is NULL, first() is used.
Invokes the callback depending on the value of optional parameter `'docallback'`.
Handles calling redraw() if anything changed.
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

in	*item*	The item that will be deselected (along with all its children). If NULL, first() is used.
in	*docallback*	– A flag that determines if the callback() is invoked or not: • 0 - the callback() is not invoked • 1 - the callback() is invoked for each item that changed state (default), callback_reason() will be FL_TREE_REASON_DESELECTED

Returns

Count of how many items were actually changed to the deselected state.

13.145.2.21 display()

```
void Fl_Tree::display (
            Fl_Tree_Item * item )
```

Displays `'item'`, scrolling the tree as necessary.

Parameters

in	*item*	The item to be displayed. If NULL, first() is used.

13.145.2.22 displayed()

```
int Fl_Tree::displayed (
            Fl_Tree_Item * item )
```

See if 'item' is currently displayed on-screen (visible within the widget).
This can be used to detect if the item is scrolled off-screen. Checks to see if the item's vertical position is within the top and bottom edges of the display window. This does NOT take into account the hide() / show() or open() / close() status of the item.

Parameters

in	*item*	The item to be checked. If NULL, first() is used.

Returns

1 if displayed, 0 if scrolled off screen or no items are in tree.

13.145.2.23 draw()

```
void Fl_Tree::draw (
            void ) [override], [protected], [virtual]
```

Standard FLTK draw() method, handles drawing the tree widget.
Reimplemented from Fl_Group.

13.145.2.24 extend_selection()

```
int Fl_Tree::extend_selection (
            Fl_Tree_Item * from,
            Fl_Tree_Item * to,
            int val = 1,
            bool visible = false )
```

Extend a selection between 'from' and 'to' depending on 'visible'.
Similar to the more efficient extend_selection_dir(Fl_Tree_Item*,Fl_Tree_Item*,int dir,int val,bool vis) method, but direction (up or down) doesn't need to be known.
We're less efficient because we search the tree for to/from, then operate on items in between. The more efficient method avoids the "search", but necessitates a direction to be specified to find 'to'.
Used by SHIFT-click to extend a selection between two items inclusive.
Handles calling redraw() if anything changed.

Parameters

in	*from*	Starting item
in	*to*	Ending item
in	*val*	Select or deselect items (0=deselect, 1=select, 2=toggle)
in	*visible*	true=affect only open(), visible items, false=affect open or closed items (default)

Returns

The number of items whose selection states were changed, if any.

Version

1.3.3 ABI feature

13.145.2.25 extend_selection_dir()

```
int Fl_Tree::extend_selection_dir (
            Fl_Tree_Item * from,
            Fl_Tree_Item * to,
```

```
int dir,
int val,
bool visible )
```

Extend the selection between and including 'from' and 'to' depending on direction 'dir', 'val', and 'visible'.
Efficient: does not walk entire tree; starts with 'from' and stops at 'to' while moving in direction 'dir'. Dir must be specified though.
If dir cannot be known in advance, such as during SHIFT-click operations, the method extend_selection(Fl_Tree_Item*,Fl_Tree_Item*,i should be used.
Handles calling redraw() if anything changed.

Parameters

in	*from*	Starting item
in	*to*	Ending item
in	*dir*	Direction to extend selection (FL_Up or FL_Down)
in	*val*	0=deselect, 1=select, 2=toggle
in	*visible*	true=affect only open(), visible items, false=affect open or closed items (default)

Returns

The number of items whose selection states were changed, if any.

Version

1.3.3

13.145.2.26 find_clicked()

```
const Fl_Tree_Item * Fl_Tree::find_clicked (
            int yonly = 0 ) const
```

Find the item that was last clicked on.
You should use callback_item() instead, which is fast, and is meant to be used within a callback to determine the item clicked.
This method walks the entire tree looking for the first item that is under the mouse. (The value of the 'yonly' flag affects whether both x and y events are checked, or just y)
Use this method /only/ if you've subclassed Fl_Tree, and are receiving events before Fl_Tree has been able to process and update callback_item().

Parameters

in	*yonly*	– 0: check both event's X and Y values. – 1: only check event's Y value, don't care about X.

Returns

The item clicked, or NULL if no item was under the current event.

Version

1.3.0

1.3.3 ABI feature: added yonly parameter

13.145.2.27 find_item()

```
const Fl_Tree_Item * Fl_Tree::find_item (
            const char * path ) const
```

Find the item, given a menu style path, e.g.
"/Parent/Child/item". There is both a const and non-const version of this method. Const version allows pure const methods to use this method to do lookups without causing compiler errors.
To specify items or submenus that contain slashes ('/' or '\') use an escape character to protect them, e.g.

```
:
tree->add("/Holidays/Photos/12\\/25\\/2010");          // Adds item "12/25/2010"
tree->add("/Pathnames/c:\\\\Program Files\\\\MyApp"); // Adds item "c:\Program Files\MyApp"
:
```

Parameters

in	*path*	– the tree item's pathname to be found (e.g. "Flintstones/Fred")

Returns

The item, or NULL if not found.

See also

item_pathname()

13.145.2.28 first()

`Fl_Tree_Item * Fl_Tree::first ( )`
Returns the first item in the tree, or 0 if none.
Use this to walk the tree in the forward direction, e.g.

```
:
for ( Fl_Tree_Item *item = tree->first(); item; item = tree->next(item) )
    printf("Item: %s\n", item->label());
:
```

Returns

First item in tree, or 0 if none (tree empty).

See also

first(), next(), last(), prev()

13.145.2.29 first_selected_item()

`Fl_Tree_Item * Fl_Tree::first_selected_item ( )`
Returns the first selected item in the tree.
Use this to walk the tree from top to bottom looking for all the selected items, e.g.

```
:
// Walk tree forward, from top to bottom
for ( Fl_Tree_Item *i=tree->first_selected_item(); i; i=tree->next_selected_item(i) )
    printf("Selected item: %s\n", i->label());
:
```

Returns

The first selected item, or 0 if none.

See also

first_selected_item(), last_selected_item(), next_selected_item()

13.145.2.30 first_visible()

```
Fl_Tree_Item * Fl_Tree::first_visible ( )
```

Returns the first open(), visible item in the tree, or 0 if none.

Deprecated in 1.3.3 ABI – use first_visible_item() instead.

13.145.2.31 first_visible_item()

```
Fl_Tree_Item * Fl_Tree::first_visible_item ( )
```

Returns the first open(), visible item in the tree, or 0 if none.

Returns

First visible item in tree, or 0 if none.

See also

first_visible_item(), last_visible_item(), next_visible_item()

Version

1.3.3

13.145.2.32 get_selected_items()

```
int Fl_Tree::get_selected_items (
            Fl_Tree_Item_Array & ret_items )
```

Returns the currently selected items as an array of `'ret_items'`.
Example:

```
:
// Get selected items as an array
Fl_Tree_Item_Array items;
tree->get_selected_items(items);
// Manipulate the returned array
for ( int t=0; t<items.total(); t++ ) {
    Fl_Tree_Item &item = items[t];
    ..do stuff with each selected item..
}
:
```

Parameters

out	*ret_items*	The returned array of selected items.

Returns

The number of items in the returned array.

See also

first_selected_item(), next_selected_item()

Version

1.3.3 ABI feature

13.145.2.33 handle()

```
int Fl_Tree::handle (
            int e )  [override], [virtual]
```

Standard FLTK event handler for this widget.

Todo add Fl_Widget_Tracker (see Fl_Browser_.cxx::handle())

Reimplemented from Fl_Group.

13.145.2.34 hposition() [1/2]

```
int Fl_Tree::hposition ( ) const
```

Returns the horizontal scroll position as a pixel offset.
The position returned is how many pixels of the tree are scrolled off the left edge of the screen.

See also

hposition(int), vposition(), vposition(int)

Note

Must be using FLTK ABI 1.3.3 or higher for this to be effective.

13.145.2.35 hposition() [2/2]

```
void Fl_Tree::hposition (
            int pos )
```

Sets the horizontal scroll offset to position `'pos'`.
The position is how many pixels of the tree are scrolled off the left edge of the screen.

Parameters

in	*pos*	The vertical position (in pixels) to scroll the tree to.

See also

hposition(), vposition(), vposition(int)

Note

Must be using FLTK ABI 1.3.3 or higher for this to be effective.

13.145.2.36 insert()

```
Fl_Tree_Item * Fl_Tree::insert (
            Fl_Tree_Item * item,
            const char * name,
            int pos )
```

Insert a new item `'name'` into `'item's` children at position `'pos'`.
If `pos` is out of range the new item is

- prepended if `pos` < 0 or
- appended if `pos` > item->children().

Note: `pos` == children() is not considered out of range: the item is appended to the child list.
Example:

```
:
tree->add("Aaa/000");       // "000" is index 0 in Aaa's children
tree->add("Aaa/111");       // "111" is index 1 in Aaa's children
tree->add("Aaa/222");       // "222" is index 2 in Aaa's children
:
// How to use insert() to insert a new item between Aaa/111 + Aaa/222
Fl_Tree_Item *item = tree->find_item("Aaa");  // get parent item Aaa
if (item) tree->insert(item, "New item", 2);  // insert as a child of Aaa at index #2
:
```

Parameters

in	*item*	The existing item to insert new child into. Must not be NULL.
in	*name*	The label for the new item
in	*pos*	The position of the new item in the child list

Returns

The new item added.

See also

insert_above()

13.145.2.37 insert_above()

```
Fl_Tree_Item * Fl_Tree::insert_above (
            Fl_Tree_Item * above,
            const char * name )
```

Inserts a new item `'name'` above the specified Fl_Tree_Item `'above'`.
Example:

```
:
tree->add("Aaa/000");       // "000" is index 0 in Aaa's children
tree->add("Aaa/111");       // "111" is index 1 in Aaa's children
tree->add("Aaa/222");       // "222" is index 2 in Aaa's children
..
// How to use insert_above() to insert a new item above Aaa/222
Fl_Tree_Item *item = tree->find_item("Aaa/222");  // get item Aaa/222
if (item) tree->insert_above(item, "New item");   // insert new item above it
:
```

Parameters

in	*above*	– the item above which to insert the new item. Must not be NULL.
in	*name*	– the name of the new item

Returns

The new item added, or 0 if 'above' could not be found.

See also

insert()

13.145.2.38 is_close() [1/2]

```
int Fl_Tree::is_close (
            const char * path ) const
```

See if item specified by `'path'` is closed.
Items or submenus that themselves contain slashes ('/' or '\') should be escaped, e.g. is_close("↩ Holidays/12\\/25\\/2010").

Parameters

in	*path*	– the tree item's pathname (e.g. "Flintstones/Fred")

Returns

- 1 - OK: item is closed
- 0 - OK: item is open
- -1 - ERROR: item was not found

13.145.2.39 is_close() [2/2]

```
int Fl_Tree::is_close (
            Fl_Tree_Item * item ) const
```

See if the specified 'item' is closed.

Parameters

in	*item*	– the item to be tested. Must not be NULL.

Returns

- 1 : item is closed
- 0 : item is open

13.145.2.40 is_hscroll_visible()

```
int Fl_Tree::is_hscroll_visible ( ) const
```

See if the horizontal scrollbar is currently visible.

Returns

1 if scrollbar visible, 0 if not.

Note

Must be using FLTK ABI 1.3.3 or higher for this to be effective.

13.145.2.41 is_open() [1/2]

```
int Fl_Tree::is_open (
            const char * path ) const
```

See if item specified by 'path' is open.
Items or submenus that themselves contain slashes ('/' or '\') should be escaped, e.g. is_open("↩ Holidays/12\\/25\\/2010").
Items that are 'open' are themselves not necessarily visible; one of the item's parents might be closed.

Parameters

in	*path*	– the tree item's pathname (e.g. "Flintstones/Fred")

Returns

- 1 - OK: item is open
- 0 - OK: item is closed
- -1 - ERROR: item was not found

See also

Fl_Tree_Item::visible_r()

13.145.2.42 is_open() [2/2]

```
int Fl_Tree::is_open (
            Fl_Tree_Item * item ) const
```

See if 'item' is open.
Items that are 'open' are themselves not necessarily visible; one of the item's parents might be closed.

Parameters

in	*item*	– the item to be tested. Must not be NULL.

Returns

- 1 : item is open
- 0 : item is closed

13.145.2.43 is_scrollbar()

```
int Fl_Tree::is_scrollbar (
            Fl_Widget * w )
```

See if widget 'w' is one of the Fl_Tree widget's scrollbars.
Use this to skip over the scrollbars when walking the child() array. Example:

```
:
for ( int i=0; i<tree->children(); i++ ) {     // walk children
    Fl_Widget *w = tree->child(i);
    if ( tree->is_scrollbar(w) ) continue;     // skip scrollbars
    ..do work here..
}
:
```

Parameters

in	*w*	Widget to test

Returns

1 if w is a scrollbar, 0 if not.

Todo should be const

13.145.2.44 is_selected() [1/2]

```
int Fl_Tree::is_selected (
            const char * path )
```

See if item specified by 'path' is selected.
Items or submenus that themselves contain slashes ('/' or '\') should be escaped, e.g. is_selected("↩ Holidays/12\\/25\\/2010").

Parameters

in	*path*	– the tree item's pathname (e.g. "Flintstones/Fred")

Returns

- 1 : item selected
- 0 : item deselected

- -1 : item was not found

13.145.2.45 is_selected() [2/2]

```
int Fl_Tree::is_selected (
            Fl_Tree_Item * item ) const
```

See if the specified 'item' is selected.

Parameters

in	*item*	– the item to be tested. Must not be NULL.

Returns

- 1 : item selected
- 0 : item deselected

13.145.2.46 is_vscroll_visible()

```
int Fl_Tree::is_vscroll_visible ( ) const
```

See if the vertical scrollbar is currently visible.

Returns

1 if scrollbar visible, 0 if not.

13.145.2.47 item_clicked() [1/2]

```
Fl_Tree_Item * Fl_Tree::item_clicked ( )
```

Return the item that was last clicked.
Valid only from within the callback().

Returns

The item clicked, or 0 if none. 0 may also be used to indicate several items were clicked/changed.

Deprecated in 1.3.3 ABI – use callback_item() instead.

13.145.2.48 item_clicked() [2/2]

```
void Fl_Tree::item_clicked (
            Fl_Tree_Item * item )  [protected]
```

Set the item that was last clicked.
Should only be used by subclasses needing to change this value. Normally Fl_Tree manages this value.

Deprecated in 1.3.3 ABI – use callback_item() instead.

13.145.2.49 item_draw_mode() [1/3]

```
Fl_Tree_Item_Draw_Mode Fl_Tree::item_draw_mode ( ) const
```

Get the 'item draw mode' used for the tree.

Version

1.3.1 ABI feature

13.145.2.50 item_draw_mode() [2/3]

```
void Fl_Tree::item_draw_mode (
            Fl_Tree_Item_Draw_Mode mode )
```

Set the 'item draw mode' used for the tree to `'mode'`.
This affects how items in the tree are drawn, such as when a widget() is defined. See Fl_Tree_Item_Draw_Mode for possible values.

Version

1.3.1 ABI feature

13.145.2.51 item_draw_mode() [3/3]

```
void Fl_Tree::item_draw_mode (
            int mode )
```

Set the 'item draw mode' used for the tree to integer `'mode'`.
This affects how items in the tree are drawn, such as when a widget() is defined. See Fl_Tree_Item_Draw_Mode for possible values.

Version

1.3.1 ABI feature

13.145.2.52 item_labelbgcolor() [1/2]

```
void Fl_Tree::item_labelbgcolor (
            Fl_Color val )
```

Set the default label background color used for creating new items.
A special case is made for color 0xffffffff (default) which is treated as 'transparent'. To change the background color on a per-item basis, use Fl_Tree_Item::labelbgcolor(Fl_Color)

13.145.2.53 item_labelbgcolor() [2/2]

```
Fl_Color Fl_Tree::item_labelbgcolor (
            void ) const
```

Get the default label background color used for creating new items.
If the color is 0xffffffff, it is 'transparent'.

13.145.2.54 item_labelfgcolor()

```
void Fl_Tree::item_labelfgcolor (
            Fl_Color val )
```

Set the default label foreground color used for creating new items.
To change the foreground color on a per-item basis, use Fl_Tree_Item::labelfgcolor(Fl_Color)

13.145.2.55 item_labelfont()

```
void Fl_Tree::item_labelfont (
            Fl_Font val )
```

Set the default font face used for creating new items.
To change the font face on a per-item basis, use Fl_Tree_Item::labelfont(Fl_Font)

13.145.2.56 item_labelsize()

```
void Fl_Tree::item_labelsize (
            Fl_Fontsize val )
```

Set the default label font size used for creating new items.
To change the font size on a per-item basis, use Fl_Tree_Item::labelsize(Fl_Fontsize)

13.145.2.57 item_pathname()

```
int Fl_Tree::item_pathname (
            char * pathname,
            int pathnamelen,
            const Fl_Tree_Item * item ) const
```

Return 'pathname' of size 'pathnamelen' for the specified 'item'.
If 'item' is NULL, root() is used.
The tree's root will be included in the pathname if showroot() is on.
Menu items or submenus that contain slashes ('/' or '\') in their names will be escaped with a backslash. This is symmetrical with the add() function which uses the same escape pattern to set names.

Parameters

out	*pathname*	The string to use to return the pathname
in	*pathnamelen*	The maximum length of the string (including NULL). Must not be zero.
in	*item*	The item whose pathname is to be returned.

Returns

- 0 : OK (pathname returns the item's pathname)
- -1 : item not found (pathname="")
- -2 : pathname not large enough (pathname="")

See also

find_item()

13.145.2.58 item_reselect_mode() [1/2]

```
Fl_Tree_Item_Reselect_Mode Fl_Tree::item_reselect_mode ( ) const
```

Returns the current item re/selection mode.

Version

1.3.1 ABI feature

13.145.2.59 item_reselect_mode() [2/2]

```
void Fl_Tree::item_reselect_mode (
            Fl_Tree_Item_Reselect_Mode mode )
```

Sets the item re/selection mode.
See Fl_Tree_Item_Reselect_Mode for possible values.

Version

1.3.1 ABI feature

13.145.2.60 last()

```
Fl_Tree_Item * Fl_Tree::last ( )
```

Returns the last item in the tree.
This can be used to walk the tree in reverse, e.g.

```
for ( Fl_Tree_Item *item = tree->last(); item; item = tree->prev() )
    printf("Item: %s\n", item->label());
```

Returns

Last item in the tree, or 0 if none (tree empty).

See also

first(), next(), last(), prev()

13.145.2.61 last_selected_item()

```
Fl_Tree_Item * Fl_Tree::last_selected_item ( )
```

Returns the last selected item in the tree.
Use this to walk the tree in reverse from bottom to top looking for all the selected items, e.g.

```
:
// Walk tree in reverse, from bottom to top
for ( Fl_Tree_Item *i=tree->last_selected_item(); i; i=tree->next_selected_item(i, FL_Up) )
    printf("Selected item: %s\n", i->label());
:
```

Returns

The last selected item, or 0 if none.

See also

first_selected_item(), last_selected_item(), next_selected_item()

Version

1.3.3

13.145.2.62 last_visible()

```
Fl_Tree_Item * Fl_Tree::last_visible ( )
```

Returns the last open(), visible item in the tree.

Deprecated in 1.3.3 – use last_visible_item() instead.

13.145.2.63 last_visible_item()

```
Fl_Tree_Item * Fl_Tree::last_visible_item ( )
```

Returns the last open(), visible item in the tree.

Returns

Last visible item in the tree, or 0 if none.

See also

first_visible_item(), last_visible_item(), next_visible_item()

Version

1.3.3

13.145.2.64 load()

```
void Fl_Tree::load (
            class Fl_Preferences & prefs )
```

Load FLTK preferences.
Read a preferences database into the tree widget.
A preferences database is a hierarchical collection of data which can be directly loaded into the tree view for inspection.

Parameters

in	*prefs*	the Fl_Preferences database

13.145.2.65 next()

```
Fl_Tree_Item * Fl_Tree::next (
            Fl_Tree_Item * item = 0 )
```

Return the next item after `'item'`, or 0 if no more items.
Use this code to walk the entire tree:

```
:
for ( Fl_Tree_Item *i = tree->first(); i; i = tree->next(i) )
    printf("Item: %s\n", i->label());
:
```

Parameters

in	*item*	The item to use to find the next item. If NULL, returns 0.

Returns

Next item in tree, or 0 if at last item.

See also

first(), next(), last(), prev()

13.145.2.66 next_item()

```
Fl_Tree_Item * Fl_Tree::next_item (
            Fl_Tree_Item * item,
            int dir = FL_Down,
            bool visible = false )
```

Returns next item after `'item'` in direction `'dir'` depending on `'visible'`.
Next item will be above (if dir==FL_Up) or below (if dir==FL_Down). If `'visible'` is true, only items whose parents are open() will be returned. If `'visible'` is false, even items whose parents are close()ed will be returned.
If `item` is 0, the return value will be the result of this truth table:

```
                        visible=true            visible=false
                        --------------------    -------------
          dir=FL_Up:    last_visible_item()     last()
        dir=FL_Down:    first_visible_item()    first()
```

Example use:

```
:
// Walk down the tree showing open(), visible items
for ( Fl_Tree_Item *i=tree->first_visible_item(); i; i=tree->next_item(i, FL_Down, true) )
    printf("Item: %s\n", i->label());

// Walk up the tree showing open(), visible items
for ( Fl_Tree_Item *i=tree->last_visible_item(); i; i=tree->next_item(i, FL_Up, true) )
    printf("Item: %s\n", i->label());

// Walk down the tree showing all items (open or closed)
for ( Fl_Tree_Item *i=tree->first(); i; i=tree->next_item(i, FL_Down, false) )
    printf("Item: %s\n", i->label());

// Walk up the tree showing all items (open or closed)
for ( Fl_Tree_Item *i=tree->last(); i; i=tree->next_item(i, FL_Up, false) )
    printf("Item: %s\n", i->label());
:
```

Parameters

in	*item*	The item to use to find the next item. If NULL, returns 0.
in	*dir*	Can be FL_Up or FL_Down (default=FL_Down or 'next')
in	*visible*	true=return only open(), visible items, false=return open or closed items (default)

Returns

Next item in tree in the direction and visibility specified, or 0 if no more items of specified visibility in that direction.

See also

first(), last(), next(),
first_visible_item(), last_visible_item(), next_visible_item(),
first_selected_item(), last_selected_item(), next_selected_item()

Version

1.3.3

13.145.2.67 next_selected_item()

```
Fl_Tree_Item * Fl_Tree::next_selected_item (
            Fl_Tree_Item * item = 0,
            int dir = FL_Down )
```

Returns the next selected item above or below `'item'`, depending on `'dir'`.
If `'item'` is 0, search starts at either first() or last(), depending on `'dir'`: first() if `'dir'` is FL_Down (default), last() if `'dir'` is FL_Up.
Use this to walk the tree looking for all the selected items, e.g.

```
:
// Walk down the tree (forwards)
for ( Fl_Tree_Item *i=tree->first_selected_item(); i; i=tree->next_selected_item(i, FL_Down) )
    printf("Item: %s\n", i->label());

// Walk up the tree (backwards)
for ( Fl_Tree_Item *i=tree->last_selected_item(); i; i=tree->next_selected_item(i, FL_Up) )
    printf("Item: %s\n", i->label());
:
```

Parameters

in	*item*	The item above or below which we'll find the next selected item. If NULL, first() is used if FL_Down, last() if FL_Up. (default=NULL)
in	*dir*	The direction to go. FL_Up for moving up the tree, FL_Down for down the tree (default)

Returns

The next selected item, or 0 if there are no more selected items.

See also

first_selected_item(), last_selected_item(), next_selected_item()

Version

1.3.3

13.145.2.68 next_visible_item()

```
Fl_Tree_Item * Fl_Tree::next_visible_item (
            Fl_Tree_Item * item,
            int dir )
```

Returns next open(), visible item above (`dir==FL_Up`) or below (`dir==FL_Down`) the specified `'item'`, or 0 if no more items.
If `'item'` is 0, returns last() if `'dir'` is FL_Up, or first() if `dir` is FL_Down.

```
:
// Walk down the tree (forwards)
for ( Fl_Tree_Item *i=tree->first_visible_item(); i; i=tree->next_visible_item(i, FL_Down) )
    printf("Item: %s\n", i->label());

// Walk up the tree (backwards)
for ( Fl_Tree_Item *i=tree->last_visible_item(); i; i=tree->next_visible_item(i, FL_Up) )
    printf("Item: %s\n", i->label());
:
```

Parameters

in	*item*	The item above/below which we'll find the next visible item
in	*dir*	The direction to search. Can be FL_Up or FL_Down.

Returns

The item found, or 0 if there's no visible items above/below the specified `item`.

Version

1.3.3

13.145.2.69 open() [1/2]

```
int Fl_Tree::open (
            const char * path,
            int docallback = 1 )
```

Opens the item specified by `'path'`.
This causes the item's children (if any) to be shown.
Invokes the callback depending on the value of optional parameter `'docallback'`.
Handles calling redraw() if anything changed.
Items or submenus that themselves contain slashes ('/' or '\') should be escaped, e.g. open("Holidays/12\\/25\\/2010").
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

in	*path*	– the tree item's pathname (e.g. "Flintstones/Fred")
in	*docallback*	– A flag that determines if the callback() is invoked or not: • 0 - callback() is not invoked • 1 - callback() is invoked if item changed (default), callback_reason() will be FL_TREE_REASON_OPENED

Returns

- 1 – OK: item opened
- 0 – OK: item was already open, no change

- -1 – ERROR: item was not found

See also

open(), close(), is_open(), is_close(), callback_item(), callback_reason()

13.145.2.70 open() [2/2]

```
int Fl_Tree::open (
            Fl_Tree_Item * item,
            int docallback = 1 )
```

Open the specified `'item'`.
This causes the item's children (if any) to be shown.
Invokes the callback depending on the value of optional parameter `'docallback'`.
Handles calling redraw() if anything changed.
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

in	*item*	– the item to be opened. Must not be NULL.
in	*docallback*	– A flag that determines if the callback() is invoked or not: • 0 - callback() is not invoked • 1 - callback() is invoked if item changed (default), callback_reason() will be FL_TREE_REASON_OPENED

Returns

- 1 – item was opened
- 0 – item was already open, no change

See also

open(), close(), is_open(), is_close(), callback_item(), callback_reason()

13.145.2.71 open_toggle()

```
void Fl_Tree::open_toggle (
            Fl_Tree_Item * item,
            int docallback = 1 )
```

Toggle the open state of `'item'`.
Invokes the callback depending on the value of optional parameter `'docallback'`.
Handles calling redraw() if anything changed.
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

in	*item*	– the item whose open state is to be toggled. Must not be NULL.
in	*docallback*	– A flag that determines if the callback() is invoked or not: • 0 - callback() is not invoked • 1 - callback() is invoked (default), callback_reason() will be either FL_TREE_REASON_OPENED or FL_TREE_REASON_CLOSED

See also

open(), close(), is_open(), is_close(), callback_item(), callback_reason()

13.145.2.72 openicon() [1/2]

```
Fl_Image * Fl_Tree::openicon ( ) const
```

Returns the icon to be used as the 'open' icon.
If none was set, the internal default is returned, a simple '[+]' icon.

13.145.2.73 openicon() [2/2]

```
void Fl_Tree::openicon (
            Fl_Image * val )
```

Sets the icon to be used as the 'open' icon.
This overrides the built in default '[+]' icon.

Parameters

in	*val*	– The new image, or zero to use the default [+] icon.

13.145.2.74 prev()

```
Fl_Tree_Item * Fl_Tree::prev (
            Fl_Tree_Item * item = 0 )
```

Return the previous item before `'item'`, or 0 if no more items.
This can be used to walk the tree in reverse, e.g.

```
:
for ( Fl_Tree_Item *item = tree->first(); item; item = tree->prev(item) )
    printf("Item: %s\n", item->label());
:
```

Parameters

in	*item*	The item to use to find the previous item. If NULL, returns 0.

Returns

Previous item in tree, or 0 if at first item.

See also

first(), next(), last(), prev()

13.145.2.75 recalc_tree()

```
void Fl_Tree::recalc_tree ( )
```

Schedule tree to recalc the entire tree size.

Note

Must be using FLTK ABI 1.3.3 or higher for this to be effective.

13.145.2.76 remove()

```
int Fl_Tree::remove (
            Fl_Tree_Item * item )
```

Remove the specified `'item'` from the tree.
`item` may not be NULL. If it has children, all those are removed too. If item being removed has focus, no item will have focus.

Returns

0 if done, -1 if 'item' not found.

13.145.2.77 resize()

```
void Fl_Tree::resize (
            int X,
            int Y,
            int W,
            int H )  [override], [virtual]
```

Resizes the Fl_Group widget and all of its children.
The Fl_Group widget first resizes itself, and then it moves and resizes all its children according to the rules documented for Fl_Group::resizable(Fl_Widget∗)

See also

Fl_Group::resizable(Fl_Widget∗)

Fl_Group::resizable()

Fl_Widget::resize(int,int,int,int)

Reimplemented from Fl_Group.

13.145.2.78 root()

```
void Fl_Tree::root (
            Fl_Tree_Item * newitem )
```

Sets the root item to `'newitem'`.
If a root item already exists, clear() is called first to clear it before replacing it with newitem.
Use this to install a custom item (derived from Fl_Tree_Item) as the root of the tree. This allows the derived class to implement custom drawing by overriding Fl_Tree_Item::draw_item_content().

Version

1.3.3

13.145.2.79 root_label()

```
void Fl_Tree::root_label (
            const char * new_label )
```

Set the label for the root item to `'new_label'`.
Makes an internally managed copy of 'new_label'.

13.145.2.80 scrollbar_size() [1/2]

```
int Fl_Tree::scrollbar_size ( ) const
```

Gets the default size of scrollbars' troughs for this widget in pixels.
If this value is zero (default), this widget will use the global Fl::scrollbar_size() value as the scrollbar's width.

Returns

Scrollbar size in pixels, or 0 if the global Fl::scrollbar_size() is being used.

See also

Fl::scrollbar_size(int)

13.145.2.81 scrollbar_size() [2/2]

```
void Fl_Tree::scrollbar_size (
            int size )
```

Sets the pixel size of the scrollbars' troughs to `'size'` for this widget, in pixels.
Normally you should not need this method, and should use the global Fl::scrollbar_size(int) instead to manage the size of ALL your widgets' scrollbars. This ensures your application has a consistent UI, and is the default behavior. Normally this is what you want.
Only use this method if you really need to override just THIS instance of the widget's scrollbar size. (This need should be rare.)
Setting `size` to the special value of 0 causes the widget to track the global Fl::scrollbar_size(), which is the default.

Parameters

in	*size*	Sets the scrollbar size in pixels. If 0 (default), scrollbar size tracks the global Fl::scrollbar_size()

See also

Fl::scrollbar_size()

13.145.2.82 select() [1/2]

```
int Fl_Tree::select (
            const char * path,
            int docallback = 1 )
```

Select the item specified by `'path'`.
Invokes the callback depending on the value of optional parameter `'docallback'`.
Handles calling redraw() if anything changed.
Items or submenus that themselves contain slashes ('/' or '\') should be escaped, e.g. select("↩ Holidays/12\\/25\\/2010").
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

in	*path*	– the tree item's pathname (e.g. "Flintstones/Fred")
in	*docallback*	– A flag that determines if the callback() is invoked or not: • 0 - the callback() is not invoked • 1 - the callback() is invoked if item changed state (default), callback_reason() will be FL_TREE_REASON_SELECTED

Returns

- 1 : OK: item's state was changed
- 0 : OK: item was already selected, no change was made
- -1 : ERROR: item was not found

13.145.2.83 select() [2/2]

```
int Fl_Tree::select (
            Fl_Tree_Item * item,
            int docallback = 1 )
```

Select the specified `'item'`.

Use 'deselect()' to deselect it.
Invokes the callback depending on the value of optional parameter `docallback`.
Handles calling redraw() if anything changed.
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

in	*item*	– the item to be selected. Must not be NULL.
in	*docallback*	– A flag that determines if the callback() is invoked or not: • 0 - the callback() is not invoked • 1 - the callback() is invoked if item changed state, callback_reason() will be FL_TREE_REASON_SELECTED

Returns

- 1 - item's state was changed
- 0 - item was already selected, no change was made

13.145.2.84 select_all()

```
int Fl_Tree::select_all (
            Fl_Tree_Item * item = 0,
            int docallback = 1 )
```

Select `'item'` and all its children.
If item is NULL, first() is used.
Invokes the callback depending on the value of optional parameter `'docallback'`.
Handles calling redraw() if anything changed.
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

in	*item*	The item that will be selected (along with all its children). If NULL, first() is used.
in	*docallback*	– A flag that determines if the callback() is invoked or not: • 0 - the callback() is not invoked • 1 - the callback() is invoked for each item that changed state (default), callback_reason() will be FL_TREE_REASON_SELECTED

Returns

Count of how many items were actually changed to the selected state.

13.145.2.85 select_only()

```
int Fl_Tree::select_only (
            Fl_Tree_Item * selitem,
            int docallback = 1 )
```

Select only the specified item, deselecting all others that might be selected.
If `'selitem'` is 0, first() is used.
Invokes the callback depending on the value of optional parameter `'docallback'`.
Handles calling redraw() if anything changed.

The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

in	*selitem*	The item to be selected. If NULL, first() is used.
in	*docallback*	– A flag that determines if the callback() is invoked or not: • 0 - the callback() is not invoked • 1 - the callback() is invoked for each item that changed state (default), callback_reason() will be either FL_TREE_REASON_SELECTED or FL_TREE_REASON_DESELECTED

Returns

The number of items whose selection states were changed, if any.

13.145.2.86 select_toggle()

```
void Fl_Tree::select_toggle (
            Fl_Tree_Item * item,
            int docallback = 1 )
```

Toggle the select state of the specified `'item'`.
Invokes the callback depending on the value of optional parameter `'docallback'`.
Handles calling redraw() if anything changed.
The callback can use callback_item() and callback_reason() respectively to determine the item changed and the reason the callback was called.

Parameters

in	*item*	– the item to be selected. Must not be NULL.
in	*docallback*	– A flag that determines if the callback() is invoked or not: • 0 - the callback() is not invoked • 1 - the callback() is invoked (default), callback_reason() will be either FL_TREE_REASON_SELECTED or FL_TREE_REASON_DESELECTED

13.145.2.87 selectbox() [1/2]

```
Fl_Boxtype Fl_Tree::selectbox ( ) const
```

Sets the style of box used to draw selected items.
This is an fltk Fl_Boxtype. The default is influenced by FLTK's current Fl::scheme()

13.145.2.88 selectbox() [2/2]

```
void Fl_Tree::selectbox (
            Fl_Boxtype val )
```

Gets the style of box used to draw selected items.
This is an fltk Fl_Boxtype. The default is influenced by FLTK's current Fl::scheme()

13.145.2.89 selectmode() [1/2]

```
Fl_Tree_Select Fl_Tree::selectmode ( ) const
```

Gets the tree's current selection mode.
See Fl_Tree_Select for possible values.

13.145.2.90 selectmode() [2/2]

```
void Fl_Tree::selectmode (
            Fl_Tree_Select val )
```

Sets the tree's selection mode.
See Fl_Tree_Select for possible values.

13.145.2.91 set_item_focus()

```
void Fl_Tree::set_item_focus (
            Fl_Tree_Item * item )
```

Set the item that currently should have keyboard focus.
Handles calling redraw() to update the focus box (if it is visible).

Parameters

in	*item*	The item that should take focus. If NULL, none will have focus.

13.145.2.92 show_item() [1/2]

```
void Fl_Tree::show_item (
            Fl_Tree_Item * item )
```

Adjust the vertical scrollbar to show `'item'` at the top of the display IF it is currently off-screen (for instance show_item_top()).
If it is already on-screen, no change is made.

Parameters

in	*item*	The item to be shown. If NULL, first() is used.

See also

show_item_top(), show_item_middle(), show_item_bottom()

13.145.2.93 show_item() [2/2]

```
void Fl_Tree::show_item (
            Fl_Tree_Item * item,
            int yoff )
```

Adjust the vertical scrollbar so that `'item'` is visible `'yoff'` pixels from the top of the Fl_Tree widget's display.
For instance, yoff=0 will position the item at the top.
If yoff is larger than the vertical scrollbar's limit, the value will be clipped. So if yoff=100, but scrollbar's max is 50, then 50 will be used.

Parameters

in	*item*	The item to be shown. If NULL, first() is used.
in	*yoff*	The pixel offset from the top for the displayed position.

See also

show_item_top(), show_item_middle(), show_item_bottom()

13.145.2.94 show_item_bottom()

```
void Fl_Tree::show_item_bottom (
            Fl_Tree_Item * item )
```

Adjust the vertical scrollbar so that `'item'` is at the bottom of the display.

Parameters

in	*item*	The item to be shown. If NULL, first() is used.

13.145.2.95 show_item_middle()

```
void Fl_Tree::show_item_middle (
            Fl_Tree_Item * item )
```

Adjust the vertical scrollbar so that `'item'` is in the middle of the display.

Parameters

in	*item*	The item to be shown. If NULL, first() is used.

13.145.2.96 show_item_top()

```
void Fl_Tree::show_item_top (
            Fl_Tree_Item * item )
```

Adjust the vertical scrollbar so that `'item'` is at the top of the display.

Parameters

in	*item*	The item to be shown. If NULL, first() is used.

13.145.2.97 show_self()

```
void Fl_Tree::show_self ( )
```

Print the tree as 'ascii art' to stdout.
Used mainly for debugging.

Todo should be const

Version

1.3.0

13.145.2.98 showcollapse() [1/2]

```
int Fl_Tree::showcollapse ( ) const
```

Returns 1 if the collapse icon is enabled, 0 if not.

See also

showcollapse(int)

13.145.2.99 showcollapse() [2/2]

```
void Fl_Tree::showcollapse (
            int val )
```

Set if we should show the collapse icon or not.
If collapse icons are disabled, the user will not be able to interactively collapse items in the tree, unless the application provides some other means via open() and close().

Parameters

in	*val*	1: shows collapse icons (default), 0: hides collapse icons.

13.145.2.100 showroot()

```
void Fl_Tree::showroot (
            int val )
```

Set if the root item should be shown or not.

Parameters

in	*val*	1 – show the root item (default) 0 – hide the root item.

13.145.2.101 sortorder()

```
Fl_Tree_Sort Fl_Tree::sortorder ( ) const
```

Set the default sort order used when items are added to the tree.
See Fl_Tree_Sort for possible values.

13.145.2.102 usericon() [1/2]

```
Fl_Image * Fl_Tree::usericon ( ) const
```

Returns the Fl_Image being used as the default user icon for all newly created items.
Returns zero if no icon has been set, which is the default.

13.145.2.103 usericon() [2/2]

```
void Fl_Tree::usericon (
            Fl_Image * val )
```

Sets the Fl_Image to be used as the default user icon for all newly created items.
If you want to specify user icons on a per-item basis, use Fl_Tree_Item::usericon() instead.

Parameters

in	*val*	– The new image to be used, or zero to disable user icons.

13.145.2.104 vposition() [1/2]

```
int Fl_Tree::vposition ( ) const
```

Returns the vertical scroll position as a pixel offset.
The position returned is how many pixels of the tree are scrolled off the top edge of the screen.

See also

vposition(int), hposition(), hposition(int)

13.145.2.105 vposition() [2/2]

```
void Fl_Tree::vposition (
            int pos )
```

Sets the vertical scroll offset to position `'pos'`.
The position is how many pixels of the tree are scrolled off the top edge of the screen.

Parameters

in	*pos*	The vertical position (in pixels) to scroll the tree to.

See also

vposition(), hposition(), hposition(int)

The documentation for this class was generated from the following files:

- Fl_Tree.H
- Fl_Tree.cxx

13.146 Fl_Tree_Item Class Reference

Tree widget item.

`#include <Fl_Tree_Item.H>`

Public Member Functions

- void activate (int val=1)

  Change the item's activation state to the optionally specified 'val'.
- Fl_Tree_Item * add (const Fl_Tree_Prefs &prefs, char **arr)

  Descend into the path specified by `'arr'`, and add a new child there.
- Fl_Tree_Item * add (const Fl_Tree_Prefs &prefs, char **arr, Fl_Tree_Item *newitem)

  Descend into path specified by `'arr'` and add `'newitem'` there.
- Fl_Tree_Item * add (const Fl_Tree_Prefs &prefs, const char *new_label)

  Add a new child to this item with the name `'new_label'` and defaults from `'prefs'`.
- Fl_Tree_Item * add (const Fl_Tree_Prefs &prefs, const char *new_label, Fl_Tree_Item *newitem)

  Add `'item'` as immediate child with `'new_label'` and defaults from `'prefs'`.
- Fl_Tree_Item * **child** (int index)

  Return the child item for the given 'index'.
- const Fl_Tree_Item * child (int t) const

  Return the const child item for the given 'index'.
- int **children** () const

  Return the number of children this item has.
- void **clear_children** ()

  Clear all the children for this item.
- void **close** ()

  Close this item and all its children.
- void deactivate ()

  Deactivate the item; the callback() won't be invoked when clicked.
- Fl_Tree_Item * deparent (int index)

  Deparent child at index position `'pos'`.
- int depth () const

  Returns how many levels deep this item is in the hierarchy.
- void **deselect** ()

  Disable the item's selection state.
- int deselect_all ()

  Deselect item and all its children.
- void draw (int X, int &Y, int W, Fl_Tree_Item *itemfocus, int &tree_item_xmax, int lastchild=1, int render=1)

  Draw this item and its children.
- virtual int draw_item_content (int render)

Draw the item content.

- int **event_on_collapse_icon** (const Fl_Tree_Prefs &prefs) const

  Was the event on the 'collapse' button of this item?
- int **event_on_item** (const Fl_Tree_Prefs &prefs) const

  Was event anywhere on the item?
- int **event_on_label** (const Fl_Tree_Prefs &prefs) const

  Was event on the label() of this item?
- int **event_on_user_icon** (const Fl_Tree_Prefs &prefs) const

  Was the event on the 'user icon' of this item, if any?
- int find_child (const char *name)

  Return the index of the immediate child of this item that has the label `'name'`.
- int find_child (Fl_Tree_Item *item)

  Find the index number for the specified `'item'` in the current item's list of children.
- Fl_Tree_Item * **find_child_item** (char **arr)

  *Non-const version of Fl_Tree_Item::find_child_item(char **arr) const.*
- const Fl_Tree_Item * find_child_item (char **arr) const

  Find child item by descending array `'arr'` of names.
- Fl_Tree_Item * **find_child_item** (const char *name)

  *Non-const version of Fl_Tree_Item::find_child_item(const char *name) const.*
- const Fl_Tree_Item * find_child_item (const char *name) const

  Return the /immediate/ child of current item that has the label `'name'`.
- Fl_Tree_Item * **find_clicked** (const Fl_Tree_Prefs &prefs, int yonly=0)

  Non-const version of Fl_Tree_Item::find_clicked(const Fl_Tree_Prefs&,int) const.
- const Fl_Tree_Item * find_clicked (const Fl_Tree_Prefs &prefs, int yonly=0) const

  Find the item that the last event was over.
- Fl_Tree_Item * **find_item** (char **arr)

  *Non-const version of Fl_Tree_Item::find_item(char **names) const.*
- const Fl_Tree_Item * find_item (char **arr) const

  Find item by descending array of `'names'`.
- **Fl_Tree_Item** (const Fl_Tree_Item *o)

  Copy constructor.
- Fl_Tree_Item (const Fl_Tree_Prefs &prefs)

  Constructor.
- Fl_Tree_Item (Fl_Tree *tree)

  Constructor.
- int **h** () const

  The item's height.
- int **has_children** () const

  See if this item has children.
- Fl_Tree_Item * insert (const Fl_Tree_Prefs &prefs, const char *new_label, int pos=0)

  Insert a new item named `'new_label'` into current item's children at a specified position `'pos'`.
- Fl_Tree_Item * insert_above (const Fl_Tree_Prefs &prefs, const char *new_label)

  Insert a new item named `'new_label'` above this item.
- char **is_activated** () const

  See if the item is activated.
- char **is_active** () const

  See if the item is activated. Alias for is_activated().
- int **is_close** () const

  See if the item is 'closed'.
- int **is_open** () const

  See if the item is 'open'.

- int **is_root** () const

  Is this item the root of the tree?
- char **is_selected** () const

  See if the item is selected.
- int **is_visible** () const

  See if the item is visible.
- int is_visible_r () const

  See if item and all its parents are open() and visible().
- const char * **label** () const

  Return the label.
- void label (const char *val)

  Set the label to `'name'`.
- int label_h () const

  The item's label height.
- int label_w () const

  The item's maximum label width to right edge of Fl_Tree's inner width within scrollbars.
- int label_x () const

  The item's label x position relative to the window.
- int label_y () const

  The item's label y position relative to the window.
- Fl_Color labelbgcolor () const

  Return item's label background text color.
- void labelbgcolor (Fl_Color val)

  Set item's label background color.
- Fl_Color **labelcolor** () const

  Return item's label text color. Alias for labelfgcolor() const).
- void **labelcolor** (Fl_Color val)

  Set item's label text color. Alias for labelfgcolor(Fl_Color)).
- Fl_Color **labelfgcolor** () const

  Return item's label foreground text color.
- void **labelfgcolor** (Fl_Color val)

  Set item's label foreground text color.
- Fl_Font **labelfont** () const

  Get item's label font face.
- void **labelfont** (Fl_Font val)

  Set item's label font face.
- Fl_Fontsize **labelsize** () const

  Get item's label font size.
- void **labelsize** (Fl_Fontsize val)

  Set item's label font size.
- int move (Fl_Tree_Item *item, int op=0, int pos=0)

  Move the current item above/below/into the specified `'item'`, where `'op'` determines the type of move:
- int move (int to, int from)

  Move an item within its parent using index numbers.
- int move_above (Fl_Tree_Item *item)

  Move the current item above the specified `'item'`.
- int move_below (Fl_Tree_Item *item)

  Move the current item below the specified `'item'`.
- int move_into (Fl_Tree_Item *item, int pos=0)

  Parent the current item as a child of the specified `'item'`.
- Fl_Tree_Item * next ()

Return the next item in the tree.

- Fl_Tree_Item * next_displayed (Fl_Tree_Prefs &prefs)

  Same as next_visible().
- Fl_Tree_Item * next_sibling ()

  Return this item's next sibling.
- Fl_Tree_Item * next_visible (Fl_Tree_Prefs &prefs)

  Return the next open(), visible() item.
- void **open** ()

  Open this item and all its children.
- void **open_toggle** ()

  Toggle the item's open/closed state.
- Fl_Tree_Item * **parent** ()

  Return the parent for this item. Returns NULL if we are the root.
- const Fl_Tree_Item * **parent** () const

  Return the const parent for this item. Returns NULL if we are the root.
- void parent (Fl_Tree_Item *val)

  Set the parent for this item.
- const Fl_Tree_Prefs & prefs () const

  Return the parent tree's prefs.
- Fl_Tree_Item * prev ()

  Return the previous item in the tree.
- Fl_Tree_Item * prev_displayed (Fl_Tree_Prefs &prefs)

  Same as prev_visible().
- Fl_Tree_Item * prev_sibling ()

  Return this item's previous sibling.
- Fl_Tree_Item * prev_visible (Fl_Tree_Prefs &prefs)

  Return the previous open(), visible() item.
- int remove_child (const char *new_label)

  Remove immediate child (and its children) by its label `'name'`.
- int remove_child (Fl_Tree_Item *item)

  Remove `'item'` from the current item's children.
- int reparent (Fl_Tree_Item *newchild, int index)

  Reparent specified item as a child of ourself at position `'pos'`.
- Fl_Tree_Item * replace (Fl_Tree_Item *new_item)

  Replace the current item with a new item.
- Fl_Tree_Item * replace_child (Fl_Tree_Item *olditem, Fl_Tree_Item *newitem)

  Replace existing child `'olditem'` with `'newitem'`.
- void select (int val=1)

  Change the item's selection state to the optionally specified 'val'.
- int select_all ()

  Select item and all its children.
- void **select_toggle** ()

  Toggle the item's selection state.
- void show_self (const char *indent="") const

  Print the tree as 'ascii art' to stdout.
- int swap_children (Fl_Tree_Item *a, Fl_Tree_Item *b)

  Swap two of our immediate children, given item pointers.
- void swap_children (int ax, int bx)

  Swap two of our children, given two child index values `'ax'` and `'bx'`.
- Fl_Tree * tree ()

  Return the tree for this item.

- const Fl_Tree * tree () const

  Return the tree for this item.
- void update_prev_next (int index)

  Update our _prev_sibling and _next_sibling pointers to point to neighbors given `index` as being our current position in the parent's item array.
- void * **user_data** () const

  Retrieve the user-data value that has been assigned to the item.
- void **user_data** (void *data)

  Set a user-data value for the item.
- Fl_Image * userdeicon () const

  Return the deactivated version of the user icon, if any.
- void userdeicon (Fl_Image *val)

  Set the usericon to draw when the item is deactivated.
- Fl_Image * **usericon** () const

  Get the item's user icon as an Fl_Image. Returns '0' if disabled.
- void usericon (Fl_Image *val)

  Set the item's user icon to an Fl_Image.
- int **visible** () const

  See if the item is visible. Alias for is_visible().
- int visible_r () const

  See if item and all its parents are open() and visible().
- int **w** () const

  The entire item's width to right edge of Fl_Tree's inner width within scrollbars.
- Fl_Widget * **widget** () const

  Return FLTK widget assigned to this item.
- void **widget** (Fl_Widget *val)

  Assign an FLTK widget to this item.
- int **x** () const

  The item's x position relative to the window.
- int **y** () const

  The item's y position relative to the window.

Protected Member Functions

- void **_Init** (const Fl_Tree_Prefs &prefs, Fl_Tree *tree)
- int calc_item_height (const Fl_Tree_Prefs &prefs) const

  Return the item's 'visible' height.
- virtual void draw_horizontal_connector (int x1, int x2, int y, const Fl_Tree_Prefs &prefs)

  Horizontal connector line based on preference settings.
- virtual void draw_vertical_connector (int x, int y1, int y2, const Fl_Tree_Prefs &prefs)

  Vertical connector line based on preference settings.
- Fl_Color drawbgcolor () const

  Returns the recommended background color used for drawing this item.
- Fl_Color drawfgcolor () const

  Returns the recommended foreground color used for drawing this item.
- void hide_widgets ()

  Internal: Hide the FLTK widget() for this item and all children.
- int **is_flag** (unsigned short val) const

  See if flag set. Returns 0 or 1.
- void recalc_tree ()

  Call this when our geometry is changed.

- void **set_flag** (unsigned short flag, int val)

  Set a flag to an on or off value. val is 0 or 1.
- void show_widgets ()

  Internal: Show the FLTK widget() for this item and all children.

13.146.1 Detailed Description

Tree widget item.
This class is a single tree item, and manages all of the item's attributes. Fl_Tree_Item is used by Fl_Tree, which is comprised of many instances of Fl_Tree_Item.
Fl_Tree_Item is hierarchical; it dynamically manages an Fl_Tree_Item_Array of children that are themselves instances of Fl_Tree_Item. Each item can have zero or more children. When an item has children, close() and open() can be used to hide or show them.
Items have their own attributes; font size, face, color. Items maintain their own hierarchy of children.
When you make changes to items, you'll need to tell the tree to redraw() for the changes to show up.
New 1.3.3 ABI feature: You can define custom items by either adding a custom widget to the item with Fl_Tree_Item::widget(), or override the draw_item_content() method if you want to just redefine how the label is drawn.
The following shows the Fl_Tree_Item's dimensions, useful when overriding the draw_item_content() method:

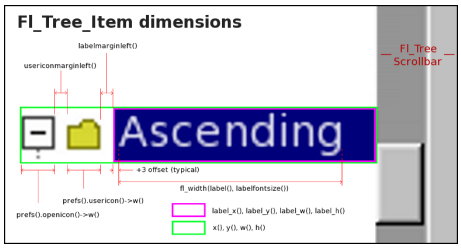


Figure 13.65 Fl_Tree_Item's internal dimensions.

13.146.2 Constructor & Destructor Documentation

13.146.2.1 Fl_Tree_Item() [1/2]

```
Fl_Tree_Item::Fl_Tree_Item (
            const Fl_Tree_Prefs & prefs )
```

Constructor.
Makes a new instance of Fl_Tree_Item using defaults from `'prefs'`.

Deprecated in 1.3.3 ABI – you must use Fl_Tree_Item(Fl_Tree*) for proper horizontal scrollbar behavior.

13.146.2.2 Fl_Tree_Item() [2/2]

```
Fl_Tree_Item::Fl_Tree_Item (
            Fl_Tree * tree )
```

Constructor.
Makes a new instance of Fl_Tree_Item for `'tree'` with an empty label (""). It's up to you to set the label() of the item before adding it to the tree so that it draws the item's name, and helps methods like Fl_Tree::find_item(path) find the item by name.
This must be used instead of the older, deprecated Fl_Tree_Item(Fl_Tree_Prefs) constructor for proper horizontal scrollbar calculation.

Version

1.3.3 ABI feature

13.146.3 Member Function Documentation

13.146.3.1 activate()

```
void Fl_Tree_Item::activate (
            int val = 1 )  [inline]
```

Change the item's activation state to the optionally specified 'val'.
When deactivated, the item will be 'grayed out'; the callback() won't be invoked if the user clicks on the label. If a widget() is associated with the item, its activation state will be changed as well.
If 'val' is not specified, the item will be activated.

13.146.3.2 add() [1/4]

```
Fl_Tree_Item * Fl_Tree_Item::add (
            const Fl_Tree_Prefs & prefs,
            char ** arr )
```

Descend into the path specified by `'arr'`, and add a new child there.
Should be used only by Fl_Tree's internals. Adds the item based on the value of prefs.sortorder().

Returns

the item added.

Version

1.3.0 release

13.146.3.3 add() [2/4]

```
Fl_Tree_Item * Fl_Tree_Item::add (
            const Fl_Tree_Prefs & prefs,
            char ** arr,
            Fl_Tree_Item * newitem )
```

Descend into path specified by `'arr'` and add `'newitem'` there.
Should be used only by Fl_Tree's internals. If item is NULL, a new item is created. Adds the item based on the value of prefs.sortorder().

Returns

the item added.

Version

1.3.3 ABI feature

13.146.3.4 add() [3/4]

```
Fl_Tree_Item * Fl_Tree_Item::add (
            const Fl_Tree_Prefs & prefs,
            const char * new_label )
```

Add a new child to this item with the name `'new_label'` and defaults from `'prefs'`.
An internally managed copy is made of the label string. Adds the item based on the value of prefs.sortorder().

Returns

the item added

Version

1.3.0 release

13.146.3.5 add() [4/4]

```
Fl_Tree_Item * Fl_Tree_Item::add (
            const Fl_Tree_Prefs & prefs,
            const char * new_label,
            Fl_Tree_Item * item )
```

Add 'item' as immediate child with 'new_label' and defaults from 'prefs'.
If 'item' is NULL, a new item is created. An internally managed copy is made of the label string. Adds the item based on the value of prefs.sortorder().

Returns

the item added

Version

1.3.3

13.146.3.6 calc_item_height()

```
int Fl_Tree_Item::calc_item_height (
            const Fl_Tree_Prefs & prefs ) const  [protected]
```

Return the item's 'visible' height.
Takes into account the item's:

- visibility (if !is_visible(), returns 0)
- labelfont() height: if label() != NULL
- widget() height: if widget() != NULL
- openicon() height (if has children)
- usericon() height (if not NULL) Does NOT include Fl_Tree::linespacing();

Returns

maximum pixel height

13.146.3.7 child()

```
const Fl_Tree_Item * Fl_Tree_Item::child (
            int t ) const
```

Return the const child item for the given 'index'.
Return const child item for the specified 'index'.

13.146.3.8 deactivate()

```
void Fl_Tree_Item::deactivate ( )  [inline]
```

Deactivate the item; the callback() won't be invoked when clicked.
Same as activate(0)

13.146.3.9 deparent()

```
Fl_Tree_Item * Fl_Tree_Item::deparent (
            int pos )
```

Deparent child at index position 'pos'.
This creates an "orphaned" item that is still allocated, but has no parent or siblings. Normally the caller would want to immediately reparent the orphan elsewhere.
A successfully orphaned item will have its parent() and prev_sibling()/next_sibling() set to NULL.

Returns

- pointer to orphaned item on success
- NULL on error (could not deparent the item)

See also

reparent()

13.146.3.10 depth()

```
int Fl_Tree_Item::depth ( ) const
```

Returns how many levels deep this item is in the hierarchy.
For instance; root has a depth of zero, and its immediate children would have a depth of 1, and so on. Use e.g. for determining the horizontal indent of this item during drawing.

13.146.3.11 deselect_all()

```
int Fl_Tree_Item::deselect_all ( )  [inline]
```

Deselect item and all its children.
Returns count of how many items were in the 'selected' state, ie. how many items were "changed".

13.146.3.12 draw()

```
void Fl_Tree_Item::draw (
            int X,
            int & Y,
            int W,
            Fl_Tree_Item * itemfocus,
            int & tree_item_xmax,
            int lastchild = 1,
            int render = 1 )
```

Draw this item and its children.

Parameters

in	*X*	Horizontal position for item being drawn
in,out	*Y*	Vertical position for item being drawn, returns new position for next item
in	*W*	Recommended width for item
in	*itemfocus*	The tree's current focus item (if any)
in,out	*tree_item_xmax*	The tree's running xmax (right-most edge so far). Mainly used by parent tree when render==0 to calculate tree's max width.
in	*lastchild*	Is this item the last child in a subtree?
in	*render*	Whether or not to render the item: 0: no rendering, just calculate size w/out drawing. 1: render item as well as size calc

Version

1.3.3 ABI feature: modified parameters

13.146.3.13 draw_horizontal_connector()

```
void Fl_Tree_Item::draw_horizontal_connector (
            int x1,
            int x2,
            int y,
            const Fl_Tree_Prefs & prefs )  [protected], [virtual]
```

Horizontal connector line based on preference settings.
This method can be overridden to implement custom connection line drawing.

Parameters

in	*x1*	The left hand X position of the horizontal connector
in	*x2*	The right hand X position of the horizontal connector
in	*y*	The vertical position of the horizontal connector
in	*prefs*	The Fl_Tree prefs

13.146.3.14 draw_item_content()

```
int Fl_Tree_Item::draw_item_content (
            int render )  [virtual]
```

Draw the item content.
This method can be overridden to implement custom drawing by filling the label_[xywh]() area with content.
A minimal example of how to override draw_item_content() and draw just a normal item's background and label ourselves:

```
class MyTreeItem : public Fl_Tree_Item {
public:
    MyTreeItem(Fl_Tree *tree_val) : Fl_Tree_Item(tree_val) { }
    // DRAW OUR CUSTOM CONTENT FOR THE ITEM
    int draw_item_content(int render) {
      // Our item's dimensions + text content
      int X=label_x(), Y=label_y(), W=label_w(), H=label_h();
      const char *text = label() ? label() : "";
      // Rendering? Do any drawing that's needed
      if ( render ) {
        // Draw bg -- a filled rectangle
        fl_color(drawbgcolor()); fl_rectf(X,Y,W,H);
        // Draw label
        fl_font(labelfont(), labelsize());      // use item's label font/size
        fl_color(drawfgcolor());                // use recommended fg color
        fl_draw(text, X,Y,W,H, FL_ALIGN_LEFT);  // draw the item's label
      }
      // Rendered or not, we must calculate content's max X position
      int lw=0, lh=0;
      fl_measure(text, lw, lh);                 // get width of label text
      return X + lw;                            // return X + label width
   }
};
```

You can draw anything you want inside draw_item_content() using any of the fl_draw.H functions, as long as it's within the label's xywh area.
To add instances of your custom item to the tree, you can use, e.g.:

```
// Example #1: using add()
MyTreeItem *item = new MyTreeItem(tree);  // class derived from Fl_Tree_Item
item->label("Bart");
tree->add("/Simpsons/.", item);           // Adds custom item as "/Simpsons/Bart"
```

..or you can insert or replace existing items:

```
// Example #2: using replace()
MyTreeItem *marge = new MyTreeItem(tree); // class derived from Fl_Tree_Item
marge->label("Marge");
Fl_Tree_Item *item = tree->add("/Simpsons/Marge"); // create default item
item->replace(marge);                              // replace default item with custom item
```

Parameters

in	*render*	Whether we should render content (1), or just tally the geometry (0). Fl_Tree may want only to find the widest item in the tree for scrollbar calculations.

Returns

the right-most X coordinate, or 'xmax' of content we drew, i.e. the "scrollable" content. The tree uses the largest xmax to determine the maximum width of the tree's content (needed for e.g. computing the horizontal scrollbar's size).

Version

1.3.3 ABI feature

13.146.3.15 draw_vertical_connector()

```
void Fl_Tree_Item::draw_vertical_connector (
            int x,
            int y1,
            int y2,
            const Fl_Tree_Prefs & prefs )  [protected], [virtual]
```

Vertical connector line based on preference settings.
This method can be overridden to implement custom connection line drawing.

Parameters

in	*x*	The x position of the vertical connector
in	*y1*	The top of the vertical connector
in	*y2*	The bottom of the vertical connector
in	*prefs*	The Fl_Tree prefs

13.146.3.16 drawbgcolor()

```
Fl_Color Fl_Tree_Item::drawbgcolor ( ) const  [protected]
```

Returns the recommended background color used for drawing this item.

See also

draw_item_content()

Version

1.3.3 ABI

13.146.3.17 drawfgcolor()

```
Fl_Color Fl_Tree_Item::drawfgcolor ( ) const  [protected]
```

Returns the recommended foreground color used for drawing this item.

See also

draw_item_content()

Version

1.3.3 ABI ABI

13.146.3.18 find_child() [1/2]

```
int Fl_Tree_Item::find_child (
            const char * name )
```

Return the index of the immediate child of this item that has the label `'name'`.

Returns

index of found item, or -1 if not found.

Version

1.3.0 release

13.146.3.19 find_child() [2/2]

```
int Fl_Tree_Item::find_child (
            Fl_Tree_Item * item )
```

Find the index number for the specified 'item' in the current item's list of children.

Returns

the index, or -1 if not found.

13.146.3.20 find_child_item() [1/2]

```
const Fl_Tree_Item * Fl_Tree_Item::find_child_item (
            char ** arr ) const
```

Find child item by descending array 'arr' of names.
Does not include self in search. Only Fl_Tree should need this method.

Returns

item, or 0 if not found

Version

1.3.0 release

13.146.3.21 find_child_item() [2/2]

```
const Fl_Tree_Item * Fl_Tree_Item::find_child_item (
            const char * name ) const
```

Return the /immediate/ child of current item that has the label 'name'.

Returns

const found item, or 0 if not found.

Version

1.3.3

13.146.3.22 find_clicked()

```
const Fl_Tree_Item * Fl_Tree_Item::find_clicked (
            const Fl_Tree_Prefs & prefs,
            int yonly = 0 ) const
```

Find the item that the last event was over.
If 'yonly' is 1, only check event's y value, don't care about x.

Parameters

in	*prefs*	The parent tree's Fl_Tree_Prefs
in	*yonly*	– 0: check both event's X and Y values. – 1: only check event's Y value, don't care about X.

Returns

pointer to clicked item, or NULL if none found

Version

1.3.3 ABI feature

13.146.3.23 find_item()

```
const Fl_Tree_Item * Fl_Tree_Item::find_item (
            char ** names ) const
```

Find item by descending array of 'names'.
Includes self in search. Only Fl_Tree should need this method. Use Fl_Tree::find_item() instead.

Returns

const item, or 0 if not found

13.146.3.24 hide_widgets()

```
void Fl_Tree_Item::hide_widgets ( )  [protected]
```

Internal: Hide the FLTK widget() for this item and all children.
Used by close() to hide widgets.

13.146.3.25 insert()

```
Fl_Tree_Item * Fl_Tree_Item::insert (
            const Fl_Tree_Prefs & prefs,
            const char * new_label,
            int pos = 0 )
```

Insert a new item named 'new_label' into current item's children at a specified position 'pos'.
If pos is out of range the new item is

- prepended if pos < 0 or
- appended if pos > item->children().

Returns

the new item inserted

See also

Fl_Tree::insert()

13.146.3.26 insert_above()

```
Fl_Tree_Item * Fl_Tree_Item::insert_above (
            const Fl_Tree_Prefs & prefs,
            const char * new_label )
```

Insert a new item named 'new_label' above this item.

Returns

the new item inserted, or 0 if an error occurred.

13.146.3.27 is_visible_r()

```
int Fl_Tree_Item::is_visible_r ( ) const
```

See if item and all its parents are open() and visible().

Returns

1 – item and its parents are open() and visible() 0 – item or one of its parents are either not visible() or close()ed.

13.146.3.28 label()

```
void Fl_Tree_Item::label (
            const char * name )
```

Set the label to `'name'`.
Makes and manages an internal copy of `'name'`.

13.146.3.29 label_h()

```
int Fl_Tree_Item::label_h ( ) const  [inline]
```

The item's label height.

Version

1.3.3

13.146.3.30 label_w()

```
int Fl_Tree_Item::label_w ( ) const  [inline]
```

The item's maximum label width to right edge of Fl_Tree's inner width within scrollbars.

Version

1.3.3

13.146.3.31 label_x()

```
int Fl_Tree_Item::label_x ( ) const  [inline]
```

The item's label x position relative to the window.

Version

1.3.3

13.146.3.32 label_y()

```
int Fl_Tree_Item::label_y ( ) const  [inline]
```

The item's label y position relative to the window.

Version

1.3.3

13.146.3.33 labelbgcolor() [1/2]

```
Fl_Color Fl_Tree_Item::labelbgcolor ( ) const  [inline]
```

Return item's label background text color.
If the color is 0xffffffff, the default behavior is the parent tree's bg color will be used. (An overloaded draw_item_content() can override this behavior.)

13.146.3.34 labelbgcolor() [2/2]

```
void Fl_Tree_Item::labelbgcolor (
            Fl_Color val )  [inline]
```

Set item's label background color.
A special case is made for color 0xffffffff which uses the parent tree's bg color.

13.146.3.35 move() [1/2]

```
int Fl_Tree_Item::move (
            Fl_Tree_Item * item,
            int op = 0,
            int pos = 0 )
```

Move the current item above/below/into the specified `'item'`, where `'op'` determines the type of move:

- 0: move above `'item'` (`'pos'` ignored)
- 1: move below `'item'` (`'pos'` ignored)
- 2: move into `'item'` as a child (at optional position `'pos'`)

..and `'pos'` determines an optional index position after the move.

Returns

0 on success. a negative number on error:

- -1: one of the items has no parent
- -2: item's index could not be determined
- -3: bad 'op'
- -4: index range error
- -5: could not deparent
- -6: could not reparent at `'pos'`
- (Other return values reserved for future use.)

See also

move_above(), move_below(), move_into(), move(int,int)

13.146.3.36 move() [2/2]

```
int Fl_Tree_Item::move (
            int to,
            int from )
```

Move an item within its parent using index numbers.
Item is moved `'to'` its new position `'from'` its old position.

Returns

- 0: Success
- -1: range error (e.g. if `'to'` or `'from'` out of range).
- (Other return values reserved for future use)

See also

move_above(), move_below(), move_into(), move(Fl_Tree_Item∗,int,int)

13.146.3.37 move_above()

```
int Fl_Tree_Item::move_above (
            Fl_Tree_Item * item )
```

Move the current item above the specified `'item'`.
This is the equivalent of calling move(item,0,0).

Returns

0 on success.
On error returns a negative value; see move(Fl_Tree_Item∗,int,int) for possible error codes.

See also

move_below(), move_into(), move(int,int), move(Fl_Tree_Item∗,int,int)

13.146.3.38 move_below()

```
int Fl_Tree_Item::move_below (
            Fl_Tree_Item * item )
```

Move the current item below the specified `'item'`.
This is the equivalent of calling move(item,1,0).

Returns

0 on success.
On error returns a negative value; see move(Fl_Tree_Item∗,int,int) for possible error codes.

See also

move_above(), move_into(), move(int,int), move(Fl_Tree_Item∗,int,int)

13.146.3.39 move_into()

```
int Fl_Tree_Item::move_into (
            Fl_Tree_Item * item,
            int pos = 0 )
```

Parent the current item as a child of the specified `'item'`.
This is the equivalent of calling move(item,2,pos).

Returns

0 on success.
On error returns a negative value; see move(Fl_Tree_Item∗,int,int) for possible error codes.

See also

move_above(), move_below(), move(int,int), move(Fl_Tree_Item∗,int,int)

13.146.3.40 next()

```
Fl_Tree_Item * Fl_Tree_Item::next ( )
```

Return the next item in the tree.
This method can be used to walk the tree forward. For an example of how to use this method, see Fl_Tree::first().

Returns

the next item in the tree, or 0 if there's no more items.

13.146.3.41 next_displayed()

```
Fl_Tree_Item * Fl_Tree_Item::next_displayed (
            Fl_Tree_Prefs & prefs )
```

Same as next_visible().

Deprecated in 1.3.3 for confusing name, use next_visible() instead

13.146.3.42 next_sibling()

```
Fl_Tree_Item * Fl_Tree_Item::next_sibling ( )
```

Return this item's next sibling.
Moves to the next item below us at the same level (sibling). Use this to move down the tree without changing depth(). effectively skipping over this item's children/descendents.

Returns

item's next sibling, or 0 if none.

13.146.3.43 next_visible()

```
Fl_Tree_Item * Fl_Tree_Item::next_visible (
            Fl_Tree_Prefs & prefs )
```

Return the next open(), visible() item.
(If this item has children and is closed, children are skipped)
This method can be used to walk the tree forward, skipping items that are not currently open/visible to the user.

Returns

the next open() visible() item below us, or 0 if there's no more items.

Version

1.3.3

13.146.3.44 parent()

```
void Fl_Tree_Item::parent (
            Fl_Tree_Item * val )  [inline]
```

Set the parent for this item.
Should only be used by Fl_Tree's internals.

13.146.3.45 prefs()

```
const Fl_Tree_Prefs & Fl_Tree_Item::prefs ( ) const
```

Return the parent tree's prefs.

Returns

a reference to the parent tree's Fl_Tree_Prefs

Version

1.3.3 ABI feature

13.146.3.46 prev()

```
Fl_Tree_Item * Fl_Tree_Item::prev ( )
```

Return the previous item in the tree.
This method can be used to walk the tree backwards. For an example of how to use this method, see Fl_Tree::last().

Returns

the previous item in the tree, or 0 if there's no item above this one (hit the root).

13.146.3.47 prev_displayed()

```
Fl_Tree_Item * Fl_Tree_Item::prev_displayed (
            Fl_Tree_Prefs & prefs )
```

Same as prev_visible().

Deprecated in 1.3.3 for confusing name, use prev_visible()

13.146.3.48 prev_sibling()

```
Fl_Tree_Item * Fl_Tree_Item::prev_sibling ( )
```

Return this item's previous sibling.
Moves to the previous item above us at the same level (sibling). Use this to move up the tree without changing depth().

Returns

This item's previous sibling, or 0 if none.

13.146.3.49 prev_visible()

```
Fl_Tree_Item * Fl_Tree_Item::prev_visible (
            Fl_Tree_Prefs & prefs )
```

Return the previous open(), visible() item.
(If this item above us has children and is closed, its children are skipped)
This method can be used to walk the tree backward, skipping items that are not currently open/visible to the user.

Returns

the previous open() visible() item above us, or 0 if there's no more items.

13.146.3.50 recalc_tree()

```
void Fl_Tree_Item::recalc_tree ( )  [protected]
```

Call this when our geometry is changed.
(Font size, label contents, etc) Schedules tree to recalculate itself, as changes to us may affect tree widget's scrollbar visibility and tab sizes.

Version

1.3.3 ABI

13.146.3.51 remove_child() [1/2]

```
int Fl_Tree_Item::remove_child (
            const char * name )
```

Remove immediate child (and its children) by its label `'name'`.
If more than one item matches `'name'`, only the first matching item is removed.

Parameters

in	*name*	The label name of the immediate child to remove

Returns

0 if removed, -1 if not found.

Version

1.3.3

13.146.3.52 remove_child() [2/2]

```
int Fl_Tree_Item::remove_child (
            Fl_Tree_Item * item )
```

Remove `'item'` from the current item's children.

Returns

0 if removed, -1 if item not an immediate child.

13.146.3.53 reparent()

```
int Fl_Tree_Item::reparent (
            Fl_Tree_Item * newchild,
            int pos )
```

Reparent specified item as a child of ourself at position `'pos'`.
Typically `'newchild'` was recently orphaned with deparent().

Returns

- 0: on success
- -1: on error (e.g. if `'pos'` out of range) with no changes made.

See also

deparent()

13.146.3.54 replace()

```
Fl_Tree_Item * Fl_Tree_Item::replace (
            Fl_Tree_Item * newitem )
```

Replace the current item with a new item.
The current item is destroyed if successful. No checks are made to see if an item with the same name exists.
This method can be used to, for example, install 'custom' items into the tree derived from Fl_Tree_Item; see draw_item_content().

Parameters

in	*newitem*	The new item to replace the current item

Returns

newitem on success, NULL if could not be replaced.

See also

Fl_Tree_Item::draw_item_content(), Fl_Tree::root(Fl_Tree_Item*)

Version

1.3.3 ABI feature

13.146.3.55 replace_child()

```
Fl_Tree_Item * Fl_Tree_Item::replace_child (
            Fl_Tree_Item * olditem,
            Fl_Tree_Item * newitem )
```

Replace existing child `'olditem'` with `'newitem'`.
The `'olditem'` is destroyed if successful. Can be used to put custom items (derived from Fl_Tree_Item) into the tree. No checks are made to see if an item with the same name exists.

Parameters

in	*olditem*	The item to be found and replaced
in	*newitem*	The new item to take the place of `'olditem'`

Returns

newitem on success and `'olditem'` is destroyed. NULL on error if `'olditem'` was not found as an immediate child.

See also

replace(), Fl_Tree_Item::draw()

Version

1.3.3 ABI feature

13.146.3.56 select()

```
void Fl_Tree_Item::select (
            int val = 1 )  [inline]
```

Change the item's selection state to the optionally specified 'val'.
If 'val' is not specified, the item will be selected.

13.146.3.57 select_all()

```
int Fl_Tree_Item::select_all ( )  [inline]
```

Select item and all its children.
Returns count of how many items were in the 'deselected' state, ie. how many items were "changed".

13.146.3.58 show_self()

```
void Fl_Tree_Item::show_self (
            const char * indent = "" ) const
```

Print the tree as 'ascii art' to stdout.
Used mainly for debugging.

13.146.3.59 show_widgets()

```
void Fl_Tree_Item::show_widgets ( )  [protected]
```

Internal: Show the FLTK widget() for this item and all children.
Used by open() to re-show widgets that were hidden by a previous close()

13.146.3.60 swap_children() [1/2]

```
int Fl_Tree_Item::swap_children (
            Fl_Tree_Item * a,
            Fl_Tree_Item * b )
```

Swap two of our immediate children, given item pointers.
Use e.g. for sorting.
This method is SLOW because it involves linear lookups.
For speed, use swap_children(int,int) instead.

Parameters

in	*a,b*	The item ptrs of the two items to swap. Both must be immediate children of the current item.

Returns

- 0 : OK
- -1 : failed: item `'a'` or `'b'` is not our child.

13.146.3.61 swap_children() [2/2]

```
void Fl_Tree_Item::swap_children (
            int ax,
            int bx )
```

Swap two of our children, given two child index values `'ax'` and `'bx'`.
Use e.g. for sorting.
This method is FAST, and does not involve lookups.
No range checking is done on either index value.

Parameters

in	*ax,bx*	the index of the items to swap

13.146.3.62 tree() [1/2]

```
Fl_Tree * Fl_Tree_Item::tree ( ) [inline]
```

Return the tree for this item.

Version

1.3.4

13.146.3.63 tree() [2/2]

```
const Fl_Tree * Fl_Tree_Item::tree ( ) const [inline]
```

Return the tree for this item.

Version

1.3.3

13.146.3.64 update_prev_next()

```
void Fl_Tree_Item::update_prev_next (
            int index )
```

Update our _prev_sibling and _next_sibling pointers to point to neighbors given `index` as being our current position in the parent's item array.
Call this whenever items in the array are added/removed/moved/swapped/etc.

Parameters

in	*index*	Our index# in the parent. Special case if index=-1: become an orphan; null out all parent/sibling associations.

13.146.3.65 userdeicon() [1/2]

```
Fl_Image * Fl_Tree_Item::userdeicon ( ) const [inline]
```

Return the deactivated version of the user icon, if any.
Returns 0 if none.

13.146.3.66 userdeicon() [2/2]

```
void Fl_Tree_Item::userdeicon (
            Fl_Image * val ) [inline]
```

Set the usericon to draw when the item is deactivated.
Use '0' to disable. No internal copy is made; caller must manage icon's memory.
To create a typical 'grayed out' version of your usericon image, you can do the following:

```
// Create tree + usericon for items
Fl_Tree *tree = new Fl_Tree(..);
Fl_Image *usr_icon = new Fl_Pixmap(..); // your usericon
Fl_Image *de_icon  = usr_icon->copy();  // make a copy, and..
de_icon->inactive();                    // make it 'grayed out'
...
for ( .. ) {                  // item loop..
  item = tree->add("...");    // create new item
  item->usericon(usr_icon);   // assign usericon to items
  item->userdeicon(de_icon);  // assign userdeicon to items
  ..
}
```

In the above example, the app should 'delete' the two icons when they're no longer needed (e.g. after the tree is destroyed)

Version

1.3.4

13.146.3.67 usericon()

```
void Fl_Tree_Item::usericon (
            Fl_Image * val )  [inline]
```

Set the item's user icon to an Fl_Image.
Use '0' to disable. No internal copy is made, caller must manage icon's memory.
Note, if you expect your items to be deactivated(), use userdeicon(Fl_Image∗) to set up a 'grayed out' version of your icon to be used for display.

See also

userdeicon(Fl_Image∗)

13.146.3.68 visible_r()

```
int Fl_Tree_Item::visible_r ( ) const  [inline]
```

See if item and all its parents are open() and visible().
Alias for is_visible_r().

Returns

1 – item and its parents are open() and visible() 0 – item (or one of its parents) are not visible or close()ed.

The documentation for this class was generated from the following files:

- Fl_Tree_Item.H
- Fl_Tree_Item.cxx

13.147 Fl_Tree_Item_Array Class Reference

Manages an array of Fl_Tree_Item pointers.
`#include <Fl_Tree_Item_Array.H>`

Public Member Functions

- void add (Fl_Tree_Item ∗val)

  Add an item∗ to the end of the array.
- void clear ()

  Clear the entire array.
- int deparent (int pos)

  Deparent item at 'pos' from our list of children.
- **Fl_Tree_Item_Array** (const Fl_Tree_Item_Array ∗o)

  Copy constructor. Makes new copy of array, with new instances of each item.
- Fl_Tree_Item_Array (int new_chunksize=10)

  Constructor; creates an empty array.
- void insert (int pos, Fl_Tree_Item ∗new_item)

  Insert an item at index position pos.
- int **manage_item_destroy** () const
- void manage_item_destroy (int val)

  Option to control if Fl_Tree_Item_Array's destructor will also destroy the Fl_Tree_Item's.
- int move (int to, int from)

Move item at 'from' to new position 'to' in the array.

- Fl_Tree_Item * **operator[]** (int i)

  Return the item and index `i`.
- const Fl_Tree_Item * **operator[]** (int i) const

  Const version of operator[](int i)
- int remove (Fl_Tree_Item *item)

  Remove the item from the array.
- void remove (int index)

  Remove the item at.
- int reparent (Fl_Tree_Item *item, Fl_Tree_Item *newparent, int pos)

  Reparent specified item as a child of ourself.
- void replace (int pos, Fl_Tree_Item *new_item)

  Replace the item at `index` with `newitem`.
- void **swap** (int ax, int bx)

  Swap the two items at index positions `ax` and `bx`.
- int **total** () const

  Return the total items in the array, or 0 if empty.
- ~**Fl_Tree_Item_Array** ()

  Destructor. Calls each item's destructor, destroys internal _items array.

13.147.1 Detailed Description

Manages an array of Fl_Tree_Item pointers.
Because FLTK 1.x.x. has mandated that templates and STL not be used, we use this class to dynamically manage the arrays.
None of the methods do range checking on index values; the caller must be sure that index values are within the range 0<index<total() (unless otherwise noted).

13.147.2 Constructor & Destructor Documentation

13.147.2.1 Fl_Tree_Item_Array()

```
Fl_Tree_Item_Array::Fl_Tree_Item_Array (
            int new_chunksize = 10 )
```

Constructor; creates an empty array.

```
The optional 'chunksize' can be specified to optimize
memory allocation for potentially large arrays. Default chunksize is 10.
```

13.147.3 Member Function Documentation

13.147.3.1 add()

```
void Fl_Tree_Item_Array::add (
            Fl_Tree_Item * val )
```

Add an item* to the end of the array.

```
Assumes the item was created with 'new', and will remain
allocated.. Fl_Tree_Item_Array will handle calling the
item's destructor when the array is cleared or the item remove()'ed.
```

13.147.3.2 clear()

```
void Fl_Tree_Item_Array::clear ( )
```

Clear the entire array.

```
Each item will be deleted (destructors will be called),
and the array will be cleared. total() will return 0.
```

13.147.3.3 deparent()

```
int Fl_Tree_Item_Array::deparent (
            int pos )
```

Deparent item at 'pos' from our list of children.
Similar to a remove() without the destruction of the item. This creates an orphaned item (still allocated, has no parent) which soon after is typically reparented elsewhere.

```
\returns 0 on success, -1 on error (e.g. if \p 'pos' out of range)
```

13.147.3.4 insert()

```
void Fl_Tree_Item_Array::insert (
            int pos,
            Fl_Tree_Item * new_item )
```

Insert an item at index position pos.

```
Handles enlarging array if needed, total increased by 1.
If \p pos \>= total(), the item is appended to the array.
If \p pos \< 0, the item is prepended (works like pos == 0).
```

13.147.3.5 manage_item_destroy()

```
void Fl_Tree_Item_Array::manage_item_destroy (
            int val )  [inline]
```

Option to control if Fl_Tree_Item_Array's destructor will also destroy the Fl_Tree_Item's.
If set: items and item array is destroyed. If clear: only the item array is destroyed, not items themselves.

13.147.3.6 move()

```
int Fl_Tree_Item_Array::move (
            int to,
            int from )
```

Move item at 'from' to new position 'to' in the array.
Due to how the moving an item shuffles the array around, a positional 'move' implies things that may not be obvious:

- When 'from' moved lower in tree, appears BELOW item that was at 'to'.
- When 'from' moved higher in tree, appears ABOVE item that was at 'to'.

Returns

0 on success, -1 on range error (e.g. if 'to' or 'from' out of range)

13.147.3.7 remove() [1/2]

```
int Fl_Tree_Item_Array::remove (
            Fl_Tree_Item * item )
```

Remove the item from the array.

```
\returns 0 if removed, or -1 if the item was not in the array.
```

13.147.3.8 remove() [2/2]

```
void Fl_Tree_Item_Array::remove (
            int index )
```

Remove the item at.

Parameters

in	*index*	from the array. The item will be delete'd (if non-NULL), so its destructor will be called.

13.147.3.9 reparent()

```
int Fl_Tree_Item_Array::reparent (
            Fl_Tree_Item * item,
            Fl_Tree_Item * newparent,
            int pos )
```

Reparent specified item as a child of ourself.
Typically 'newchild' was recently orphaned with deparent().

```
\returns 0 on success, -1 on error (e.g. if \p 'pos' out of range)
```

13.147.3.10 replace()

```
void Fl_Tree_Item_Array::replace (
            int index,
            Fl_Tree_Item * newitem )
```

Replace the item at `index` with `newitem`.
Old item at index position will be destroyed, and the new item will take it's place, and stitched into the linked list.
The documentation for this class was generated from the following files:

- Fl_Tree_Item_Array.H
- Fl_Tree_Item_Array.cxx

13.148 Fl_Tree_Prefs Class Reference

Tree widget's preferences.
`#include <Fl_Tree_Prefs.H>`

Public Member Functions

- Fl_Image * closedeicon () const

  Return the deactivated version of the close icon, if any.
- Fl_Image * **closeicon** () const

  Gets the default 'close' icon Returns the Fl_Image of the icon, or 0 if none.*
- void closeicon (Fl_Image *val)

  Sets the icon to be used as the 'close' icon.
- int **closeicon_h** () const
- int **closeicon_w** () const
- Fl_Color **connectorcolor** () const

  Get the connector color used for tree connection lines.
- void **connectorcolor** (Fl_Color val)

  Set the connector color used for tree connection lines.
- Fl_Tree_Connector **connectorstyle** () const

  Get the connector style.
- void **connectorstyle** (Fl_Tree_Connector val)

  Set the connector style.
- void **connectorstyle** (int val)

  Set the connector style [integer].

- int **connectorwidth** () const

  Get the tree connection line's width.
- void **connectorwidth** (int val)

  Set the tree connection line's width.
- void **do_item_draw_callback** (Fl_Tree_Item *o) const
- **Fl_Tree_Prefs** ()

  Fl_Tree_Prefs constructor.
- Fl_Tree_Item_Draw_Callback * **item_draw_callback** () const
- void **item_draw_callback** (Fl_Tree_Item_Draw_Callback *cb, void *data=0)
- Fl_Tree_Item_Draw_Mode **item_draw_mode** () const

  Get the 'item draw mode' used for the tree.
- void item_draw_mode (Fl_Tree_Item_Draw_Mode val)

  Set the 'item draw mode' used for the tree to `val`.
- void * **item_draw_user_data** () const
- Fl_Color item_labelbgcolor () const

  Get the default label background color.
- void item_labelbgcolor (Fl_Color val)

  Set the default label background color.
- Fl_Color **item_labelfgcolor** () const

  Get the default label foreground color.
- void **item_labelfgcolor** (Fl_Color val)

  Set the default label foreground color.
- Fl_Font **item_labelfont** () const

  Return the label's font.
- void **item_labelfont** (Fl_Font val)

  Set the label's font to `val`.
- Fl_Fontsize **item_labelsize** () const

  Return the label's size in pixels.
- void **item_labelsize** (Fl_Fontsize val)

  Set the label's size in pixels to `val`.
- Fl_Tree_Item_Reselect_Mode **item_reselect_mode** () const

  Returns the current item re/selection mode.
- void **item_reselect_mode** (Fl_Tree_Item_Reselect_Mode mode)

  Sets the item re/selection mode.
- Fl_Color **labelbgcolor** () const

  Obsolete: Get the default label background color. Please use item_labelbgcolor() instead.
- void **labelbgcolor** (Fl_Color val)

  Obsolete: Set the default label background color. Please use item_labelbgcolor(Fl_Color) instead.
- Fl_Color **labelfgcolor** () const

  Obsolete: Get the default label foreground color. Please use item_labelfgcolor() instead.
- void **labelfgcolor** (Fl_Color val)

  Obsolete: Set the default label foreground color. Please use item_labelfgcolor(Fl_Color) instead.
- Fl_Font **labelfont** () const

  Obsolete: Return the label's font. Please use item_labelfont() instead.
- void **labelfont** (Fl_Font val)

  Obsolete: Set the label's font to `val`. Please use item_labelfont(Fl_Font) instead.
- int **labelmarginleft** () const

  Get the label's left margin value in pixels.
- void **labelmarginleft** (int val)

  Set the label's left margin value in pixels.
- Fl_Fontsize **labelsize** () const

Obsolete: Return the label's size in pixels. Please use item_labelsize() instead.

- void **labelsize** (Fl_Fontsize val)

  Obsolete: Set the label's size in pixels to `val`. Please use item_labelsize(Fl_Fontsize) instead.
- int **linespacing** () const

  Get the line spacing value in pixels.
- void **linespacing** (int val)

  Set the line spacing value in pixels.
- int marginbottom () const

  Get the bottom margin's value in pixels.
- void **marginbottom** (int val)

  Set the bottom margin's value in pixels This is the extra distance the vertical scroller lets you travel.
- int **marginleft** () const

  Get the left margin's value in pixels.
- void **marginleft** (int val)

  Set the left margin's value in pixels.
- int **margintop** () const

  Get the top margin's value in pixels.
- void **margintop** (int val)

  Set the top margin's value in pixels.
- int **openchild_marginbottom** () const

  Get the margin below an open child in pixels.
- void **openchild_marginbottom** (int val)

  Set the margin below an open child in pixels.
- Fl_Image * opendeicon () const

  Return the deactivated version of the open icon, if any.
- Fl_Image * openicon () const

  Get the current default 'open' icon.
- void openicon (Fl_Image *val)

  Sets the default icon to be used as the 'open' icon when items are add()ed to the tree.
- int **openicon_h** () const
- int **openicon_w** () const
- Fl_Boxtype **selectbox** () const

  Get the default selection box's box drawing style as an Fl_Boxtype.
- void **selectbox** (Fl_Boxtype val)

  Set the default selection box's box drawing style to `val`.
- Fl_Tree_Select **selectmode** () const

  Get the selection mode used for the tree.
- void selectmode (Fl_Tree_Select val)

  Set the selection mode used for the tree to `val`.
- char **showcollapse** () const

  Returns 1 if the collapse icon is enabled, 0 if not.
- void showcollapse (int val)

  Set if we should show the collapse icon or not.
- int **showroot** () const

  Returns 1 if the root item is to be shown, or 0 if not.
- void showroot (int val)

  Set if the root item should be shown or not.
- Fl_Tree_Sort **sortorder** () const

  Get the default sort order value.
- void sortorder (Fl_Tree_Sort val)

  Set the default sort order value.

- Fl_Image * userdeicon () const

  Return the deactivated version of the user icon, if any.
- Fl_Image * **usericon** () const

  Gets the default 'user icon' (default is 0)
- void **usericon** (Fl_Image *val)

  Sets the default 'user icon' Returns the Fl_Image of the icon, or 0 if none (default).*
- int **usericonmarginleft** () const

  Get the user icon's left margin value in pixels.
- void **usericonmarginleft** (int val)

  Set the user icon's left margin value in pixels.
- int **widgetmarginleft** () const

  Get the widget()'s left margin value in pixels.
- void **widgetmarginleft** (int val)

  Set the widget's left margin value in pixels.
- ~**Fl_Tree_Prefs** ()

  Fl_Tree_Prefs destructor.

13.148.1 Detailed Description

Tree widget's preferences.
Fl_Tree's Preferences class.
This class manages the Fl_Tree's defaults. You should probably be using the methods in Fl_Tree instead of trying to accessing tree's preferences settings directly.

13.148.2 Member Function Documentation

13.148.2.1 closedeicon()

```
Fl_Image * Fl_Tree_Prefs::closedeicon ( ) const  [inline]
```

Return the deactivated version of the close icon, if any.
Returns 0 if none.

13.148.2.2 closeicon()

```
void Fl_Tree_Prefs::closeicon (
            Fl_Image * val )
```

Sets the icon to be used as the 'close' icon.
This overrides the built in default '[-]' icon.

Parameters

in	*val*	– The new image, or zero to use the default [-] icon.

13.148.2.3 item_draw_mode()

```
void Fl_Tree_Prefs::item_draw_mode (
            Fl_Tree_Item_Draw_Mode val )  [inline]
```

Set the 'item draw mode' used for the tree to `val`.
This affects how items in the tree are drawn, such as when a widget() is defined. See Fl_Tree_Item_Draw_Mode for possible values.

13.148.2.4 item_labelbgcolor() [1/2]

```
Fl_Color Fl_Tree_Prefs::item_labelbgcolor ( ) const  [inline]
```

Get the default label background color.
This returns the Fl_Tree::color() unless item_labelbgcolor() has been set explicitly.

13.148.2.5 item_labelbgcolor() [2/2]

```
void Fl_Tree_Prefs::item_labelbgcolor (
            Fl_Color val ) [inline]
```

Set the default label background color.
Once set, overrides the default behavior of using Fl_Tree::color().

13.148.2.6 marginbottom()

```
int Fl_Tree_Prefs::marginbottom ( ) const [inline]
```

Get the bottom margin's value in pixels.
This is the extra distance the vertical scroller lets you travel.

13.148.2.7 opendeicon()

```
Fl_Image * Fl_Tree_Prefs::opendeicon ( ) const [inline]
```

Return the deactivated version of the open icon, if any.
Returns 0 if none.

13.148.2.8 openicon() [1/2]

```
Fl_Image * Fl_Tree_Prefs::openicon ( ) const [inline]
```

Get the current default 'open' icon.
Returns the Fl_Image∗ of the icon, or 0 if none.

13.148.2.9 openicon() [2/2]

```
void Fl_Tree_Prefs::openicon (
            Fl_Image * val )
```

Sets the default icon to be used as the 'open' icon when items are add()ed to the tree.
This overrides the built in default '[+]' icon.

Parameters

in	*val*	– The new image, or zero to use the default [+] icon.

13.148.2.10 selectmode()

```
void Fl_Tree_Prefs::selectmode (
            Fl_Tree_Select val ) [inline]
```

Set the selection mode used for the tree to `val`.
This affects how items in the tree are selected when clicked on and dragged over by the mouse. See Fl_Tree_Select for possible values.

13.148.2.11 showcollapse()

```
void Fl_Tree_Prefs::showcollapse (
            int val ) [inline]
```

Set if we should show the collapse icon or not.
If collapse icons are disabled, the user will not be able to interactively collapse items in the tree, unless the application provides some other means via open() and close().

Parameters

in	*val*	1: shows collapse icons (default), 0: hides collapse icons.

13.148.2.12 showroot()

```
void Fl_Tree_Prefs::showroot (
            int val ) [inline]
```

Set if the root item should be shown or not.

Parameters

in	*val*	1 – show the root item (default) 0 – hide the root item.

13.148.2.13 sortorder()

```
void Fl_Tree_Prefs::sortorder (
            Fl_Tree_Sort val ) [inline]
```

Set the default sort order value.
Defines the order new items appear when add()ed to the tree. See Fl_Tree_Sort for possible values.

13.148.2.14 userdeicon()

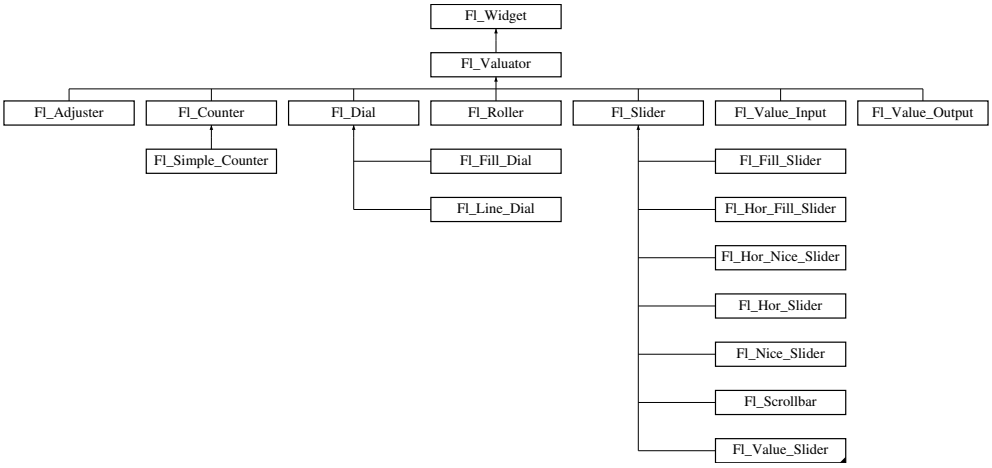
```
Fl_Image * Fl_Tree_Prefs::userdeicon ( ) const [inline]
```

Return the deactivated version of the user icon, if any.
Returns 0 if none.
The documentation for this class was generated from the following files:

- Fl_Tree_Prefs.H
- Fl_Tree_Prefs.cxx

13.149 Fl_Valuator Class Reference

The Fl_Valuator class controls a single floating-point value and provides a consistent interface to set the value, range, and step, and insures that callbacks are done the same for every object.
`#include <Fl_Valuator.H>`
Inheritance diagram for Fl_Valuator:



Public Member Functions

- void **bounds** (double a, double b)

  Sets the minimum (a) and maximum (b) values for the valuator widget.
- double **clamp** (double)

  Clamps the passed value to the valuator range.

- virtual int format (char ∗)

  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.
- virtual std::string format_str ()

  C++11 API for Fl_Valuator::format(char∗ buffer).
- double increment (double, int)

  Adds n times the step value to the passed value.
- double maximum () const

  Gets the maximum value for the valuator.
- void maximum (double a)

  Sets the maximum value for the valuator.
- double minimum () const

  Gets the minimum value for the valuator.
- void minimum (double a)

  Sets the minimum value for the valuator.
- void precision (int digits)

  Sets the step value to 1.0 / 10^{digits}.
- void range (double a, double b)

  Sets the minimum and maximum values for the valuator.
- double round (double)

  Round the passed value to the nearest step increment.
- double step () const

  Gets or sets the step value.
- void **step** (double a, int b)

  See double Fl_Valuator::step() const
- void **step** (double s)

  See double Fl_Valuator::step() const.
- void **step** (int a)

  See double Fl_Valuator::step() const
- double value () const

  Gets the floating point(double) value.
- int value (double)

  Sets the current value.
- ∼**Fl_Valuator** () override

  Destructor is accessible despite protected constructor.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

Gets the current user data (long) argument that is passed to the callback function.

- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

Gets the background color of the widget.

- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- virtual void draw ()=0

  Draws the widget.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual int handle (int event)

  Handles the specified event.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.

- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

Internal use only - "for hacks only".

- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.

- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- Fl_Valuator (int X, int Y, int W, int H, const char ∗L)

  Creates a new Fl_Valuator widget using the given position, size, and label string.
- void **handle_drag** (double newvalue)

  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.
- void **handle_push** ()

  Stores the current value in the previous value.
- void **handle_release** ()

  Called after an FL_WHEN_RELEASE event is received and before the callback.
- int **horizontal** () const

  Tells if the valuator is an FL_HORIZONTAL one.
- double **previous_value** () const

  Gets the previous floating point value before an event changed it.
- void **set_value** (double v)

  Sets the current floating point value.

- double **softclamp** (double)

  Clamps the value, but accepts v if the previous value is not already out of range.
- virtual void value_damage ()

  Asks for partial redraw.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.149.1 Detailed Description

The Fl_Valuator class controls a single floating-point value and provides a consistent interface to set the value, range, and step, and insures that callbacks are done the same for every object.
There are probably more of these classes in FLTK than any others:

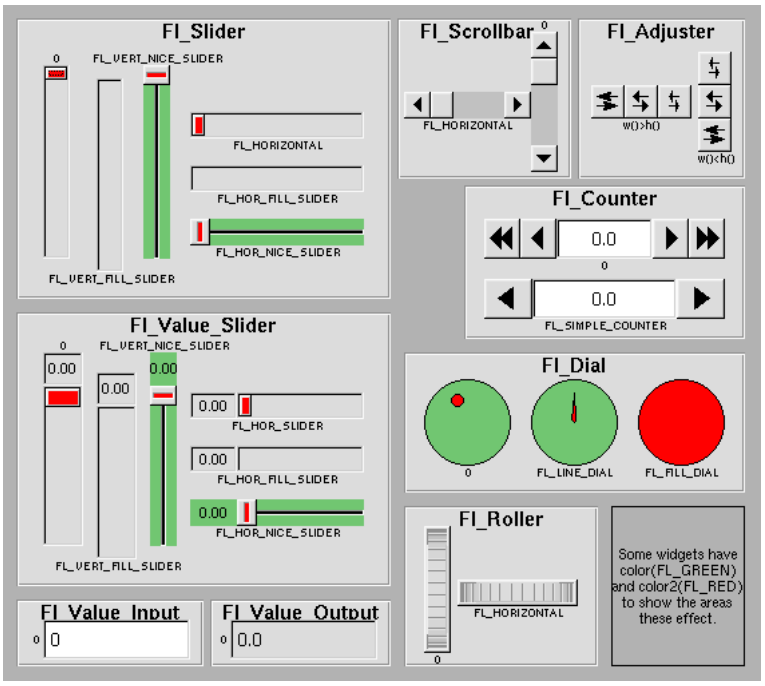


Figure 13.66 Valuators derived from Fl_Valuators

In the above diagram each box surrounds an actual subclass. These are further differentiated by setting the type() of the widget to the symbolic value labeling the widget. The ones labelled "0" are the default versions with a type(0). For consistency the symbol FL_VERTICAL is defined as zero.

13.149.2 Constructor & Destructor Documentation

13.149.2.1 Fl_Valuator()

```
Fl_Valuator::Fl_Valuator (
            int X,
            int Y,
```

```
int W,
int H,
const char * L )  [protected]
```

Creates a new Fl_Valuator widget using the given position, size, and label string.
The default boxtype is FL_NO_BOX.

13.149.3 Member Function Documentation

13.149.3.1 format()

```
int Fl_Valuator::format (
            char * buffer )  [virtual]
```

Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter. The actual format used depends on the current step value. If the step value has been set to zero then a %g format is used. If the step value is non-zero, then a %.∗f format is used, where the precision is calculated to show sufficient digits for the current step value. An integer step value, such as 1 or 1.0, gives a precision of 0, so the formatted value will appear as an integer.
This method is used by the Fl_Valuator_... group of widgets to format the current value into a text string. The return value is the length of the formatted text. The formatted value is written into `buffer`. `buffer` should have space for at least 128 bytes.
You may override this function to create your own text formatting.

13.149.3.2 format_str()

```
std::string Fl_Valuator::format_str ( )  [virtual]
```

C++11 API for Fl_Valuator::format(char∗ buffer).
Users can override either version to change the format of the text output in the valuator.

Returns

the formatted text of the current value

See also

Fl_Valuator::format(char∗ buffer)

13.149.3.3 increment()

```
double Fl_Valuator::increment (
            double v,
            int n )
```

Adds n times the step value to the passed value.
If step was set to zero it uses fabs(maximum() - minimum()) / 100.

13.149.3.4 maximum() [1/2]

```
double Fl_Valuator::maximum ( ) const  [inline]
```

Gets the maximum value for the valuator.

13.149.3.5 maximum() [2/2]

```
void Fl_Valuator::maximum (
            double a )  [inline]
```

Sets the maximum value for the valuator.

13.149.3.6 minimum() [1/2]

```
double Fl_Valuator::minimum ( ) const  [inline]
```

Gets the minimum value for the valuator.

13.149.3.7 minimum() [2/2]

```
void Fl_Valuator::minimum (
            double a )  [inline]
```

Sets the minimum value for the valuator.

13.149.3.8 precision()

```
void Fl_Valuator::precision (
            int digits )
```

Sets the step value to 1.0 / 10^{digits}.
Precision `digits` is limited to 0...9 to avoid internal overflow errors. Values outside this range are clamped.

Note

For negative values of `digits` the step value is set to `A` = 1.0 and `B` = 1, i.e. 1.0/1 = 1.

13.149.3.9 range()

```
void Fl_Valuator::range (
            double a,
            double b )  [inline]
```

Sets the minimum and maximum values for the valuator.
When the user manipulates the widget, the value is limited to this range. This clamping is done *after* rounding to the step value (this makes a difference if the range is not a multiple of the step).
The minimum may be greater than the maximum. This has the effect of "reversing" the object so the larger values are in the opposite direction. This also switches which end of the filled sliders is filled.
Some widgets consider this a "soft" range. This means they will stop at the range, but if the user releases and grabs the control again and tries to move it further, it is allowed.
The range may affect the display. You must redraw() the widget after changing the range.

13.149.3.10 round()

```
double Fl_Valuator::round (
            double v )
```

Round the passed value to the nearest step increment.
Does nothing if step is zero.

13.149.3.11 step()

```
double Fl_Valuator::step ( ) const  [inline]
```

Gets or sets the step value.
As the user moves the mouse the value is rounded to the nearest multiple of the step value. This is done *before* clamping it to the range. For most widgets the default step is zero.
For precision the step is stored as the ratio of a double `A` and an integer `B` = A/B. You can set these values directly. Currently setting a floating point value sets the nearest A/1 or 1/B value possible.

13.149.3.12 value() [1/2]

```
double Fl_Valuator::value ( ) const  [inline]
```

Gets the floating point(double) value.
See int value(double)

13.149.3.13 value() [2/2]

```
int Fl_Valuator::value (
            double v )
```

Sets the current value.

The new value is *not* clamped or otherwise changed before storing it. Use clamp() or round() to modify the value before calling value(). The widget is redrawn if the new value is different than the current one. The initial value is zero.

changed() will return true if the user has moved the slider, but it will be turned off by value(x) and just before doing a callback (the callback can turn it back on if desired).

13.149.3.14 value_damage()

```
void Fl_Valuator::value_damage ( ) [protected], [virtual]
```

Asks for partial redraw.

Reimplemented in Fl_Adjuster.

The documentation for this class was generated from the following files:

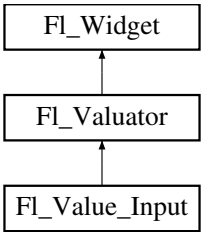
- Fl_Valuator.H
- Fl_Valuator.cxx

13.150 Fl_Value_Input Class Reference

The Fl_Value_Input widget displays a numeric value.

`#include <Fl_Value_Input.H>`

Inheritance diagram for Fl_Value_Input:



Public Member Functions

- Fl_Color cursor_color () const

  Gets the color of the text cursor.
- void cursor_color (Fl_Color n)

  Sets the color of the text cursor.
- Fl_Value_Input (int x, int y, int w, int h, const char *l=0)

  Creates a new Fl_Value_Input widget using the given position, size, and label string.
- int handle (int) override

  Handles the specified event.
- void resize (int, int, int, int) override

  Changes the size or position of the widget.
- int shortcut () const

  Returns the current shortcut key for the Input.
- void shortcut (int s)

  Sets the shortcut key to `s`.
- char soft () const

  If "soft" is turned on, the user is allowed to drag the value outside the range.
- void **soft** (char s)

  See void Fl_Value_Input::soft(char s)

- Fl_Color textcolor () const

  Gets the color of the text in the value box.
- void **textcolor** (Fl_Color n)

  Sets the color of the text in the value box.
- Fl_Font textfont () const

  Gets the typeface of the text in the value box.
- void textfont (Fl_Font s)

  Sets the typeface of the text in the value box.
- Fl_Fontsize textsize () const

  Gets the size of the text in the value box.
- void textsize (Fl_Fontsize s)

  Sets the size of the text in the value box.

Public Member Functions inherited from Fl_Valuator

- void **bounds** (double a, double b)

  Sets the minimum (a) and maximum (b) values for the valuator widget.
- double **clamp** (double)

  Clamps the passed value to the valuator range.
- virtual int format (char ∗)

  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.
- virtual std::string format_str ()

  C++11 API for Fl_Valuator::format(char∗ buffer).
- double increment (double, int)

  Adds n times the step value to the passed value.
- double maximum () const

  Gets the maximum value for the valuator.
- void maximum (double a)

  Sets the maximum value for the valuator.
- double minimum () const

  Gets the minimum value for the valuator.
- void minimum (double a)

  Sets the minimum value for the valuator.
- void precision (int digits)

  Sets the step value to 1.0 / 10^{digits}.
- void range (double a, double b)

  Sets the minimum and maximum values for the valuator.
- double round (double)

  Round the passed value to the nearest step increment.
- double step () const

  Gets or sets the step value.
- void **step** (double a, int b)

  See double Fl_Valuator::step() const
- void **step** (double s)

  See double Fl_Valuator::step() const.
- void **step** (int a)

  See double Fl_Valuator::step() const
- double value () const

  Gets the floating point(double) value.

- int value (double)

  Sets the current value.
- ~**Fl_Valuator** () override

  Destructor is accessible despite protected constructor.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

Sets the current callback function and data for the widget.

- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.

- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

Sets the font size in pixels.

- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.

- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Public Attributes

- Fl_Input **input**

Protected Member Functions

- void draw () override

  Draws the widget.

Protected Member Functions inherited from Fl_Valuator

- Fl_Valuator (int X, int Y, int W, int H, const char *L)

  Creates a new Fl_Valuator widget using the given position, size, and label string.
- void **handle_drag** (double newvalue)

  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.
- void **handle_push** ()

  Stores the current value in the previous value.
- void **handle_release** ()

  Called after an FL_WHEN_RELEASE event is received and before the callback.
- int **horizontal** () const

  Tells if the valuator is an FL_HORIZONTAL one.
- double **previous_value** () const

  Gets the previous floating point value before an event changed it.
- void **set_value** (double v)

  Sets the current floating point value.
- double **softclamp** (double)

  Clamps the value, but accepts v if the previous value is not already out of range.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.

- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.150.1 Detailed Description

The Fl_Value_Input widget displays a numeric value.
The user can click in the text field and edit it - there is in fact a hidden Fl_Input widget with type(FL_FLOAT_INPUT) or type(FL_INT_INPUT) in there - and when they hit return or tab the value updates to what they typed and the callback is done.
If step() is non-zero and integral, then the range of numbers is limited to integers instead of floating point numbers. As well as displaying the value as an integer, typed input is also limited to integer values, even if the hidden Fl_Input widget is of type(FL_FLOAT_INPUT).
If step() is non-zero, the user can also drag the mouse across the object and thus slide the value. The left button moves one step() per pixel, the middle by 10 step(), and the right button by 100 * step(). It is therefore impossible to select text by dragging across it, although clicking can still move the insertion cursor.
If step() is non-zero and integral, then the range of numbers are limited to integers instead of floating point values.

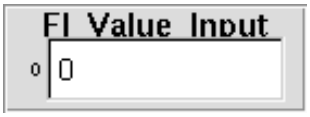


Figure 13.67 Fl_Value_Input

See also

Fl_Widget::shortcut_label(int)

13.150.2 Constructor & Destructor Documentation

13.150.2.1 Fl_Value_Input()

```
Fl_Value_Input::Fl_Value_Input (
            int X,
            int Y,
            int W,
            int H,
            const char * l = 0 )
```

Creates a new Fl_Value_Input widget using the given position, size, and label string.
The default boxtype is FL_DOWN_BOX.

13.150.3 Member Function Documentation

13.150.3.1 cursor_color() [1/2]

```
Fl_Color Fl_Value_Input::cursor_color ( ) const  [inline]
```

Gets the color of the text cursor.
The text cursor is black by default.

13.150.3.2 cursor_color() [2/2]

```
void Fl_Value_Input::cursor_color (
            Fl_Color n )  [inline]
```

Sets the color of the text cursor.
The text cursor is black by default.

13.150.3.3 draw()

```
void Fl_Value_Input::draw ( )  [override], [protected], [virtual]
```

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();                 // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

13.150.3.4 handle()

```
int Fl_Value_Input::handle (
            int event )  [override], [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to *override* the behavior of the base class. This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to *filter* event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you *consumed* the event by returning 1 even if you didn't *do* anything with it.

Parameters

in	*event*	the kind of event received

Return values

0	if the event was not used or understood
1	if the event was used and can be deleted

See also

Fl_Event

Reimplemented from Fl_Widget.

13.150.3.5 resize()

```
void Fl_Value_Input::resize (
            int x,
            int y,
            int w,
            int h )  [override], [virtual]
```

Changes the size or position of the widget.
This is a virtual function so that the widget may implement its own handling of resizing. The default version does *not* call the redraw() method, but instead relies on the parent widget to do so because the parent may know a faster way to update the display, such as scrolling from the old position.
Some window managers under X11 call resize() a lot more often than needed. Please verify that the position or size of a widget did actually change before doing any extensive calculations.
position(X, Y) is a shortcut for resize(X, Y, w(), h()), and size(W, H) is a shortcut for resize(x(), y(), W, H).

Parameters

in	*x,y*	new position relative to the parent window
in	*w,h*	new size

See also

position(int,int), size(int,int)

Reimplemented from Fl_Widget.

13.150.3.6 shortcut() [1/2]

```
int Fl_Value_Input::shortcut ( ) const  [inline]
```

Returns the current shortcut key for the Input.

See also

Fl_Value_Input::shortcut(int)

13.150.3.7 shortcut() [2/2]

```
void Fl_Value_Input::shortcut (
            int s )  [inline]
```

Sets the shortcut key to `s`.
Setting this overrides the use of '&' in the label(). The value is a bitwise OR of a key and a set of shift flags, for example FL_ALT | 'a' , FL_ALT | (FL_F + 10), or just 'a'. A value of 0 disables the shortcut.

The key can be any value returned by Fl::event_key(), but will usually be an ASCII letter. Use a lower-case letter unless you require the shift key to be held down.
The shift flags can be any set of values accepted by Fl::event_state(). If the bit is on that shift key must be pushed. Meta, Alt, Ctrl, and Shift must be off if they are not in the shift flags (zero for the other bits indicates a "don't care" setting).

13.150.3.8 soft()

`char Fl_Value_Input::soft ( ) const [inline]`
If "soft" is turned on, the user is allowed to drag the value outside the range.
If they drag the value to one of the ends, let go, then grab again and continue to drag, they can get to any value. The default is true.

13.150.3.9 textcolor()

`Fl_Color Fl_Value_Input::textcolor ( ) const [inline]`
Gets the color of the text in the value box.

13.150.3.10 textfont() [1/2]

`Fl_Font Fl_Value_Input::textfont ( ) const [inline]`
Gets the typeface of the text in the value box.

13.150.3.11 textfont() [2/2]

```
void Fl_Value_Input::textfont (
            Fl_Font s ) [inline]
```
Sets the typeface of the text in the value box.

13.150.3.12 textsize() [1/2]

`Fl_Fontsize Fl_Value_Input::textsize ( ) const [inline]`
Gets the size of the text in the value box.

13.150.3.13 textsize() [2/2]

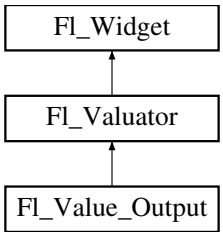
```
void Fl_Value_Input::textsize (
            Fl_Fontsize s ) [inline]
```
Sets the size of the text in the value box.

The documentation for this class was generated from the following files:

- Fl_Value_Input.H
- Fl_Value_Input.cxx

13.151 Fl_Value_Output Class Reference

The Fl_Value_Output widget displays a floating point value.
`#include <Fl_Value_Output.H>`
Inheritance diagram for Fl_Value_Output:



Public Member Functions

- Fl_Value_Output (int x, int y, int w, int h, const char *l=0)

  Creates a new Fl_Value_Output widget using the given position, size, and label string.
- int handle (int) override

  Handles the specified event.
- uchar soft () const

  If "soft" is turned on, the user is allowed to drag the value outside the range.
- void soft (uchar s)

  If "soft" is turned on, the user is allowed to drag the value outside the range.
- Fl_Color textcolor () const

  Sets the color of the text in the value box.
- void textcolor (Fl_Color s)

  Gets the color of the text in the value box.
- Fl_Font textfont () const

  Gets the typeface of the text in the value box.
- void textfont (Fl_Font s)

  Sets the typeface of the text in the value box.
- Fl_Fontsize textsize () const

  Gets the size of the text in the value box.
- void **textsize** (Fl_Fontsize s)

Public Member Functions inherited from Fl_Valuator

- void **bounds** (double a, double b)

  Sets the minimum (a) and maximum (b) values for the valuator widget.
- double **clamp** (double)

  Clamps the passed value to the valuator range.
- virtual int format (char *)

  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.
- virtual std::string format_str ()

  C++11 API for Fl_Valuator::format(char buffer).*
- double increment (double, int)

  Adds n times the step value to the passed value.
- double maximum () const

  Gets the maximum value for the valuator.
- void maximum (double a)

  Sets the maximum value for the valuator.
- double minimum () const

  Gets the minimum value for the valuator.
- void minimum (double a)

  Sets the minimum value for the valuator.
- void precision (int digits)

  Sets the step value to $1.0 / 10^{digits}$.

- void range (double a, double b)

  Sets the minimum and maximum values for the valuator.
- double round (double)

  Round the passed value to the nearest step increment.
- double step () const

  Gets or sets the step value.
- void **step** (double a, int b)

  See double Fl_Valuator::step() const
- void **step** (double s)

  See double Fl_Valuator::step() const.
- void **step** (int a)

  See double Fl_Valuator::step() const
- double value () const

  Gets the floating point(double) value.
- int value (double)

  Sets the current value.
- ~**Fl_Valuator** () override

  Destructor is accessible despite protected constructor.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

Sets the damage bits for an area inside the widget.

- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.

- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

Sets a widget to output only.

- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.

- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void draw () override

  Draws the widget.

Protected Member Functions inherited from Fl_Valuator

- Fl_Valuator (int X, int Y, int W, int H, const char ∗L)

  Creates a new Fl_Valuator widget using the given position, size, and label string.
- void **handle_drag** (double newvalue)

  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.
- void **handle_push** ()

  Stores the current value in the previous value.
- void **handle_release** ()

  Called after an FL_WHEN_RELEASE event is received and before the callback.
- int **horizontal** () const

  Tells if the valuator is an FL_HORIZONTAL one.
- double **previous_value** () const

  Gets the previous floating point value before an event changed it.
- void **set_value** (double v)

  Sets the current floating point value.
- double **softclamp** (double)

  Clamps the value, but accepts v if the previous value is not already out of range.
- virtual void value_damage ()

  Asks for partial redraw.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

Draws a box of type t, of color c at the position X,Y and size W,H.

- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.151.1 Detailed Description

The Fl_Value_Output widget displays a floating point value.
If step() is not zero, the user can adjust the value by dragging the mouse left and right. The left button moves one step() per pixel, the middle by 10 ∗ step(), and the right button by 100 ∗ step().
This is much lighter-weight than Fl_Value_Input because it contains no text editing code or character buffer.

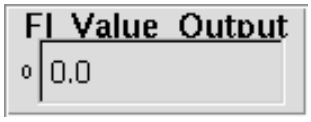


Figure 13.68 Fl_Value_Output

13.151.2 Constructor & Destructor Documentation

13.151.2.1 Fl_Value_Output()

```
Fl_Value_Output::Fl_Value_Output (
            int X,
            int Y,
            int W,
            int H,
            const char * l = 0 )
```

Creates a new Fl_Value_Output widget using the given position, size, and label string.
The default boxtype is FL_NO_BOX.
Inherited destructor destroys the Valuator.

13.151.3 Member Function Documentation

13.151.3.1 draw()

```
void Fl_Value_Output::draw ( )  [override], [protected], [virtual]
```

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();                 // calls Fl_Scrollbar::draw()
```

Implements Fl_Widget.

13.151.3.2 handle()

```
int Fl_Value_Output::handle (
            int event )  [override], [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to *override* the behavior of the base class. This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to *filter* event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you *consumed* the event by returning 1 even if you didn't *do* anything with it.

Parameters

in	*event*	the kind of event received

Return values

0	if the event was not used or understood
1	if the event was used and can be deleted

See also

Fl_Event

Reimplemented from Fl_Widget.

13.151.3.3 soft() [1/2]

```
uchar Fl_Value_Output::soft ( ) const  [inline]
```

If "soft" is turned on, the user is allowed to drag the value outside the range.
If they drag the value to one of the ends, let go, then grab again and continue to drag, they can get to any value.
Default is one.

13.151.3.4 soft() [2/2]

```
void Fl_Value_Output::soft (
            uchar s )  [inline]
```

If "soft" is turned on, the user is allowed to drag the value outside the range.
If they drag the value to one of the ends, let go, then grab again and continue to drag, they can get to any value.
Default is one.

13.151.3.5 textcolor() [1/2]

```
Fl_Color Fl_Value_Output::textcolor ( ) const  [inline]
```

Sets the color of the text in the value box.

13.151.3.6 textcolor() [2/2]

```
void Fl_Value_Output::textcolor (
            Fl_Color s )  [inline]
```

Gets the color of the text in the value box.

13.151.3.7 textfont() [1/2]

```
Fl_Font Fl_Value_Output::textfont ( ) const  [inline]
```

Gets the typeface of the text in the value box.

13.151.3.8 textfont() [2/2]

```
void Fl_Value_Output::textfont (
            Fl_Font s )  [inline]
```

Sets the typeface of the text in the value box.

13.151.3.9 textsize()

```
Fl_Fontsize Fl_Value_Output::textsize ( ) const  [inline]
```

Gets the size of the text in the value box.

The documentation for this class was generated from the following files:

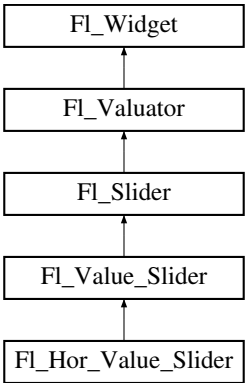
- Fl_Value_Output.H
- Fl_Value_Output.cxx

13.152 Fl_Value_Slider Class Reference

The Fl_Value_Slider widget is a Fl_Slider widget with a box displaying the current value.

```
#include <Fl_Value_Slider.H>
```

Inheritance diagram for Fl_Value_Slider:



Public Member Functions

- Fl_Value_Slider (int x, int y, int w, int h, const char *l=0)

  Creates a new Fl_Value_Slider widget using the given position, size, and label string.
- int handle (int) override

  Handles the specified event.
- Fl_Color **textcolor** () const

  Gets the color of the text in the value box.
- void **textcolor** (Fl_Color s)

  Sets the color of the text in the value box.
- Fl_Font **textfont** () const

  Gets the typeface of the text in the value box.
- void **textfont** (Fl_Font s)

  Sets the typeface of the text in the value box.
- Fl_Fontsize **textsize** () const

  Gets the size of the text in the value box.
- void **textsize** (Fl_Fontsize s)

  Sets the size of the text in the value box.
- int value_height () const

  Gets the height of the value box in pixels (vertical mode only).
- void value_height (int s)

  Sets the height of the value box in pixels (vertical mode only).
- int value_width () const

  Gets the width of the value box in pixels (horizontal mode only).
- void value_width (int s)

  Sets the width of the value box in pixels (horizontal mode only).

Public Member Functions inherited from Fl_Slider

- void bounds (double a, double b)

  Sets the minimum (a) and maximum (b) values for the valuator widget.
- Fl_Slider (int X, int Y, int W, int H, const char ∗L=0)

  Creates a new Fl_Slider widget using the given position, size, and label string.
- **Fl_Slider** (uchar t, int X, int Y, int W, int H, const char ∗L)

  Creates a new Fl_Slider widget using the given type, position, size, and label string.
- uchar num_ticks () const

  Gets the number of tick marks.
- Scale_Type scale () const

  Gets the scale type.
- void scale (Scale_Type s)

  Sets the scale type.
- int scrollvalue (int pos, int size, int first, int total)

  Sets the size and position of the sliding knob in the box.
- Fl_Boxtype **slider** () const

  Gets the slider box type.
- void **slider** (Fl_Boxtype c)

  Sets the slider box type.
- float **slider_size** () const

  Get the dimensions of the moving piece of slider.
- void slider_size (double v)

  Set the dimensions of the moving piece of slider.
- uchar ticks () const

  Gets the tick mark position.
- void ticks (uchar t, uchar num_ticks=11)

  Sets the tick mark position for Fl_Nice_Slider.

Public Member Functions inherited from Fl_Valuator

- void **bounds** (double a, double b)

  Sets the minimum (a) and maximum (b) values for the valuator widget.
- double **clamp** (double)

  Clamps the passed value to the valuator range.
- virtual int format (char ∗)

  Uses internal rules to format the fields numerical value into the character array pointed to by the passed parameter.
- virtual std::string format_str ()

  C++11 API for Fl_Valuator::format(char∗ buffer).
- double increment (double, int)

  Adds n times the step value to the passed value.
- double maximum () const

  Gets the maximum value for the valuator.
- void maximum (double a)

  Sets the maximum value for the valuator.
- double minimum () const

  Gets the minimum value for the valuator.
- void minimum (double a)

  Sets the minimum value for the valuator.
- void precision (int digits)

  Sets the step value to 1.0 / 10^{digits}.
- void range (double a, double b)

Sets the minimum and maximum values for the valuator.

- double round (double)

  Round the passed value to the nearest step increment.
- double step () const

  Gets or sets the step value.
- void **step** (double a, int b)

  See double Fl_Valuator::step() const
- void **step** (double s)

  See double Fl_Valuator::step() const.
- void **step** (int a)

  See double Fl_Valuator::step() const
- double value () const

  Gets the floating point(double) value.
- int value (double)

  Sets the current value.
- ~**Fl_Valuator** () override

  Destructor is accessible despite protected constructor.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Group * as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const * **as_group** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

Sets the image to use as part of the widget label when in the active state.

- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.

- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

Return the gap size between the label and the image.

- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.

- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

Gets the widget width.

- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void draw () override

  Draws the widget.

Protected Member Functions inherited from Fl_Slider

- void **draw** (int, int, int, int)
- void draw_ticks (const Fl_Rect &r, int S)

  Draw tick marks.
- int **handle** (int, int, int, int, int)
- double increment_lin_log (double v, int n, int range)

  Adds a sensible step value to the passed value.
- double position_to_value (double pos) const

  Converts a normalized position (0.0 to 1.0) to a value based on the current scale type.
- double value_to_position (double val) const

  Converts a value to a normalized position (0.0 to 1.0) based on the current scale type.

Protected Member Functions inherited from Fl_Valuator

- Fl_Valuator (int X, int Y, int W, int H, const char ∗L)

  Creates a new Fl_Valuator widget using the given position, size, and label string.
- void **handle_drag** (double newvalue)

  Called during a drag operation, after an FL_WHEN_CHANGED event is received and before the callback.
- void **handle_push** ()

  Stores the current value in the previous value.
- void **handle_release** ()

  Called after an FL_WHEN_RELEASE event is received and before the callback.
- int **horizontal** () const

  Tells if the valuator is an FL_HORIZONTAL one.
- double **previous_value** () const

  Gets the previous floating point value before an event changed it.
- void **set_value** (double v)

  Sets the current floating point value.
- double **softclamp** (double)

  Clamps the value, but accepts v if the previous value is not already out of range.
- virtual void value_damage ()

  Asks for partial redraw.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Additional Inherited Members

Public Types inherited from Fl_Slider

- enum Scale_Type : uchar { LINEAR_SCALE = 0 , LOG_SCALE }

  Bitset for tick mark positions for Fl_Nice_Slider.
- enum Tick_Position : uchar {
  TICKS_NONE = 0x00 , TICKS_BELOW = 0x01 , TICKS_ABOVE = 0x02 , TICKS_LEFT = 0x01 ,
  TICKS_RIGHT = 0x02 }

  Bitset for tick mark positions for Fl_Nice_Slider.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.152.1 Detailed Description

The Fl_Value_Slider widget is a Fl_Slider widget with a box displaying the current value.

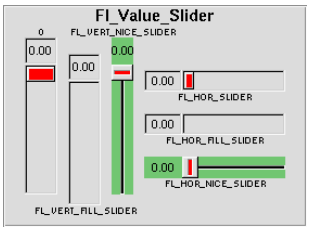


Figure 13.69 Fl_Value_Slider

13.152.2 Constructor & Destructor Documentation

13.152.2.1 Fl_Value_Slider()

```
Fl_Value_Slider::Fl_Value_Slider (
            int X,
            int Y,
            int W,
            int H,
            const char * l = 0 )
```

Creates a new Fl_Value_Slider widget using the given position, size, and label string.
The default boxtype is FL_DOWN_BOX.

13.152.3 Member Function Documentation

13.152.3.1 draw()

```
void Fl_Value_Slider::draw ( )  [override], [protected], [virtual]
```

Draws the widget.
Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.
Override this function to draw your own widgets.
If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();                 // calls Fl_Scrollbar::draw()
```

Reimplemented from Fl_Slider.

13.152.3.2 handle()

```
int Fl_Value_Slider::handle (
            int event )  [override], [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to *override* the behavior of the base class. This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to *filter* event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you *consumed* the event by returning 1 even if you didn't *do* anything with it.

Parameters

in	*event*	the kind of event received

Return values

0	if the event was not used or understood
1	if the event was used and can be deleted

See also

Fl_Event

Reimplemented from Fl_Slider.

13.152.3.3 value_height() [1/2]

```
int Fl_Value_Slider::value_height ( ) const  [inline]
```

Gets the height of the value box in pixels (vertical mode only).

Since

1.4.0

13.152.3.4 value_height() [2/2]

```
void Fl_Value_Slider::value_height (
            int s )  [inline]
```

Sets the height of the value box in pixels (vertical mode only).
Limited range checking is applied but drawing errors may occur if the size `s` is set too high or too low, particularly if the widget is resized (later).
The programmer is responsible for setting sensible values and widget sizes.

The default value set by the constructor is 25.

Parameters

in	*s*	new height of the value box

Since

1.4.0

13.152.3.5 value_width() [1/2]

```
int Fl_Value_Slider::value_width ( ) const  [inline]
```

Gets the width of the value box in pixels (horizontal mode only).

Since

1.4.0

13.152.3.6 value_width() [2/2]

```
void Fl_Value_Slider::value_width (
            int s )  [inline]
```

Sets the width of the value box in pixels (horizontal mode only).
Limited range checking is applied but drawing errors may occur if the size `s` is set too high or too low, particularly if the widget is resized (later).
The programmer is responsible for setting sensible values and widget sizes.
The default value set by the constructor is 35.

Parameters

in	*s*	new width of the value box

Since

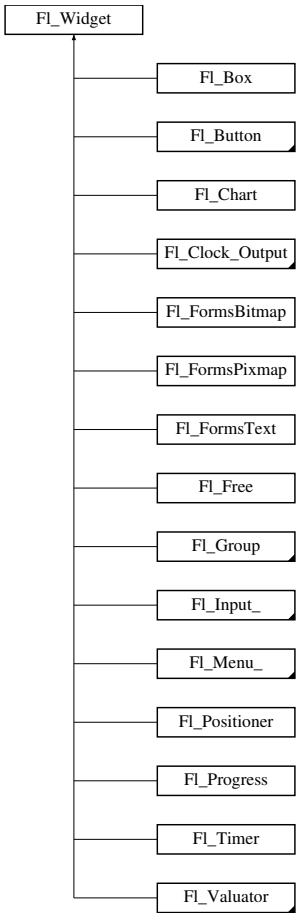
1.4.0

The documentation for this class was generated from the following files:

- Fl_Value_Slider.H
- Fl_Value_Slider.cxx

13.153 Fl_Widget Class Reference

Fl_Widget is the base class for all widgets in FLTK.
`#include <Fl_Widget.H>`
Inheritance diagram for Fl_Widget:



Public Member Functions

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window ∗ as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const ∗ **as_gl_window** () const
- virtual Fl_Group ∗ as_group ()

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- virtual Fl_Group const ∗ **as_group** () const
- virtual Fl_Window ∗ as_window ()

Returns an Fl_Window pointer if this widget is an Fl_Window.

- virtual Fl_Window const ∗ **as_window** () const
- void bind_deimage (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image ∗img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback ∗cb, Fl_Callback_User_Data ∗p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback ∗cb, void ∗p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 ∗cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 ∗cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget ∗w) const

  Checks if w is a child of this widget.
- void copy_label (const char ∗new_label)

  Sets the current label.
- void copy_tooltip (const char ∗text)

Sets the current tooltip text.

- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- virtual void draw ()=0

  Draws the widget.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual int handle (int event)

  Handles the specified event.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.

- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- virtual void resize (int x, int y, int w, int h)

  Changes the size or position of the widget.
- Fl_Color selection_color () const

Gets the selection color.

- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

  Gets the current tooltip text.
- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.

- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Static Public Member Functions

- static void default_callback (Fl_Widget ∗widget, void ∗data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char ∗t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char ∗, const bool require_alt=false)

  Returns true if the given text t contains the entered '&x' shortcut.

Protected Types

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

Protected Member Functions

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Friends

- class **Fl_Group**

13.153.1 Detailed Description

Fl_Widget is the base class for all widgets in FLTK.
You can't create one of these because the constructor is not public. However you can subclass it.
All "property" accessing methods, such as color(), parent(), or argument() are implemented as trivial inline functions and thus are as fast and small as accessing fields in a structure. Unless otherwise noted, the property setting methods such as color(n) or label(s) are also trivial inline functions, even if they change the widget's appearance. It is up to the user code to call redraw() after these.

13.153.2 Member Enumeration Documentation

13.153.2.1 anonymous enum

`anonymous enum [protected]`

flags possible values enumeration.

See activate(), output(), visible(), changed(), set_visible_focus()

Enumerator

INACTIVE	the widget can't receive focus, and is disabled but potentially visible
INVISIBLE	the widget is not drawn, but can receive a few special events
OUTPUT	for output only
NOBORDER	don't draw a decoration (Fl_Window)
FORCE_POSITION	don't let the window manager position the window (Fl_Window)
NON_MODAL	this is a hovering toolbar window (Fl_Window)
SHORTCUT_LABEL	the label contains a shortcut we need to draw
CHANGED	the widget value changed
OVERRIDE	position window on top (Fl_Window)
VISIBLE_FOCUS	accepts keyboard focus navigation if the widget can have the focus
COPIED_LABEL	the widget label is internally copied, its destruction is handled by the widget
CLIP_CHILDREN	all drawing within this widget will be clipped (Fl_Group)
MENU_WINDOW	a temporary popup window, dismissed by clicking outside (Fl_Window)
TOOLTIP_WINDOW	a temporary popup, transparent to events, and dismissed easily (Fl_Window)
MODAL	a window blocking input to all other windows (Fl_Window)
NO_OVERLAY	window not using a hardware overlay plane (Fl_Menu_Window)
GROUP_RELATIVE	Reserved, not implemented. DO NOT USE.
COPIED_TOOLTIP	the widget tooltip is internally copied, its destruction is handled by the widget
FULLSCREEN	a fullscreen window (Fl_Window)
MAC_USE_ACCENTS_MENU	On the macOS platform, pressing and holding a key on the keyboard opens an accented-character menu window (Fl_Input_, Fl_Text_Editor)
NEEDS_KEYBOARD	set on touch screen devices if a widget needs a keyboard when it gets the focus. Reserved, not yet used in 1.4.0. **See also** Fl_Widget::needs_keyboard()
IMAGE_BOUND	binding the image to the widget will transfer ownership, so that the widget will delete the image when it is no longer needed
DEIMAGE_BOUND	bind the inactive image to the widget, so the widget deletes the image when it is no longer needed
AUTO_DELETE_USER_DATA	automatically call `delete` on the user_data pointer when destroying this widget; if set, user_data must point to a class derived from the class Fl_Callback_User_Data
MAXIMIZED	a maximized Fl_Window
POPUP	popup window (i.e., positioned relatively to another mapped window)
USERFLAG3	reserved for 3rd party extensions
USERFLAG2	reserved for 3rd party extensions
USERFLAG1	reserved for 3rd party extensions

13.153.3 Constructor & Destructor Documentation

13.153.3.1 Fl_Widget()

```
Fl_Widget::Fl_Widget (
            int x,
            int y,
            int w,
            int h,
            const char * label = 0L )  [protected]
```

Creates a widget at the given position and size.
The Fl_Widget is a protected constructor, but all derived widgets have a matching public constructor. It takes a value for x(), y(), w(), h(), and an optional value for label().

Parameters

in	*x,y*	the position of the widget relative to the enclosing window
in	*w,h*	size of the widget in pixels
in	*label*	optional text for the widget label

13.153.3.2 ~Fl_Widget()

`Fl_Widget::~Fl_Widget ( ) [virtual]`
Destroys the widget.
Destroys the widget, taking care of throwing focus before if any.
Destroying single widgets is not very common. You almost always want to destroy the parent group instead, which will destroy all of the child widgets and groups in that group.

Since

FLTK 1.3, the widget's destructor removes the widget from its parent group, if it is member of a group.

Destruction removes the widget from any parent group! And groups when destroyed destroy all their children. This is convenient and fast. Also removes the widget from the Pen API list.

13.153.4 Member Function Documentation

13.153.4.1 activate()

`void Fl_Widget::activate ( )`
Activates the widget.
Changing this value will send FL_ACTIVATE to the widget if active_r() is true.

See also

active(), active_r(), deactivate()

13.153.4.2 active()

`unsigned int Fl_Widget::active ( ) const [inline]`
Returns whether the widget is active.

Return values

0	if the widget is inactive

See also

active_r(), activate(), deactivate()

13.153.4.3 active_r()

`int Fl_Widget::active_r ( ) const`
Returns whether the widget and all of its parents are active.

Return values

0	if this or any of the parent widgets are inactive

See also

active(), activate(), deactivate()

13.153.4.4 align() [1/2]

```
Fl_Align Fl_Widget::align ( ) const  [inline]
```

Gets the label alignment.

Returns

label alignment

See also

label(), align(Fl_Align), Fl_Align

13.153.4.5 align() [2/2]

```
void Fl_Widget::align (
            Fl_Align alignment )  [inline]
```

Sets the label alignment.

This controls how the label is displayed next to or inside the widget. The default value is FL_ALIGN_CENTER, which centers the label inside the widget.

Parameters

in	*alignment*	new label alignment

See also

align(), Fl_Align

13.153.4.6 argument() [1/2]

```
long Fl_Widget::argument ( ) const  [inline]
```

Gets the current user data (long) argument that is passed to the callback function.

Note

On platforms with `sizeof(long) < sizeof(void*)`, particularly on Windows 64-bit platforms, this method can truncate stored addresses (void*) to the size of a `long` value. Use with care and only if you are sure that the stored user_data value fits in a `long` value because it was stored with argument(long) or another method using only `long` values. You may want to use user_data() instead.

See also

user_data()

13.153.4.7 argument() [2/2]

```
void Fl_Widget::argument (
            long v )  [inline]
```

Sets the current user data (long) argument that is passed to the callback function.

See also

argument()

13.153.4.8 as_gl_window()

`virtual class Fl_Gl_Window * Fl_Widget::as_gl_window ( ) [inline], [virtual]`

Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.

Use this method if you have a widget (pointer) and need to know whether this widget is derived from Fl_Gl_Window. If it returns non-NULL, then the widget in question is derived from Fl_Gl_Window.

Return values

NULL	if this widget is not derived from Fl_Gl_Window.

Note

This method is provided to avoid dynamic_cast.

See also

Fl_Widget::as_group(), Fl_Widget::as_window()

Reimplemented in Fl_Gl_Window.

13.153.4.9 as_group()

`virtual Fl_Group * Fl_Widget::as_group ( ) [inline], [virtual]`

Returns an Fl_Group pointer if this widget is an Fl_Group.

Use this method if you have a widget (pointer) and need to know whether this widget is derived from Fl_Group. If it returns non-NULL, then the widget in question is derived from Fl_Group, and you can use the returned pointer to access its children or other Fl_Group-specific methods.

Example:

```
void my_callback (Fl_Widget *w, void *) {
  Fl_Group *g = w->as_group();
  if (g)
    printf ("This group has %d children\n", g->children());
  else
    printf ("This widget is not a group!\n");
}
```

Return values

NULL	if this widget is not derived from Fl_Group.

Note

This method is provided to avoid dynamic_cast.

See also

Fl_Widget::as_window(), Fl_Widget::as_gl_window()

Reimplemented in Fl_Group.

13.153.4.10 as_window()

`virtual Fl_Window * Fl_Widget::as_window ( ) [inline], [virtual]`

Returns an Fl_Window pointer if this widget is an Fl_Window.

Use this method if you have a widget (pointer) and need to know whether this widget is derived from Fl_Window. If it returns non-NULL, then the widget in question is derived from Fl_Window, and you can use the returned pointer to access its children or other Fl_Window-specific methods.

Return values

NULL	if this widget is not derived from Fl_Window.

Note

This method is provided to avoid dynamic_cast.

See also

Fl_Widget::as_group(), Fl_Widget::as_gl_window()

Reimplemented in Fl_Window.

13.153.4.11 bind_deimage() [1/2]

```
void Fl_Widget::bind_deimage (
            Fl_Image * img )
```

Sets the image to use as part of the widget label when in the inactive state.

Parameters

in	*img*	the new image for the deactivated widget

Note

The image will be bound to the widget. When the widget is deleted, the image will be deleted as well.

See also

void deimage(Fl_Image* img)

13.153.4.12 bind_deimage() [2/2]

```
void Fl_Widget::bind_deimage (
            int f )  [inline]
```

Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.

Parameters

f	1: mark the image as bound, 0: mark the image as managed by the user

See also

deimage_bound(), bind_image()

13.153.4.13 bind_image() [1/2]

```
void Fl_Widget::bind_image (
            Fl_Image * img )
```

Sets the image to use as part of the widget label when in the active state.
The image will be bound to the widget. When the widget is deleted, the image will be deleted as well.
Calling bind_image() with a new image will delete the old image if it was bound, and then set the new image, and bind that. If old and new image are the same, nothing happens.
Calling bind_image() with NULL will delete the old image if it was bound and not set a new image.

Parameters

in	*img*	the new image for the label

See also

void image(Fl_Image* img)

13.153.4.14 bind_image() [2/2]

```
void Fl_Widget::bind_image (
            int f ) [inline]
```

Bind the image to the widget, so the widget will delete the image when it is no longer needed.

Parameters

f	1: mark the image as bound, 0: mark the image as managed by the user

See also

image_bound(), bind_deimage()

13.153.4.15 box() [1/2]

```
Fl_Boxtype Fl_Widget::box (
            void ) const [inline]
```

Gets the box type of the widget.

Returns

the current box type

See also

box(Fl_Boxtype), Fl_Boxtype

13.153.4.16 box() [2/2]

```
void Fl_Widget::box (
            Fl_Boxtype new_box ) [inline]
```

Sets the box type for the widget.
This identifies a routine that draws the background of the widget. See Fl_Boxtype for the available types. The default depends on the widget, but is usually FL_NO_BOX or FL_UP_BOX.

Parameters

in	*new_box*	the new box type

See also

box(), Fl_Boxtype

13.153.4.17 callback() [1/6]

```
Fl_Callback_p Fl_Widget::callback ( ) const [inline]
```

Gets the current callback function for the widget.
Each widget has a single callback.

Returns

current callback

13.153.4.18 callback() [2/6]

```
void Fl_Widget::callback (
            Fl_Callback * cb )  [inline]
```

Sets the current callback function for the widget.
Each widget has a single callback.

Parameters

in	*cb*	new callback

13.153.4.19 callback() [3/6]

```
void Fl_Widget::callback (
            Fl_Callback * cb,
            Fl_Callback_User_Data * p,
            bool auto_free )  [inline]
```

Sets the current callback function and managed user data for the widget.
Setting auto_free will transfer ownership of the callback user data to the widget. Deleting the widget will then also delete the user data.

Parameters

in	*cb*	new callback
in	*p*	user data
in	*auto_free*	if set, the widget will free user data when destroyed

13.153.4.20 callback() [4/6]

```
void Fl_Widget::callback (
            Fl_Callback * cb,
            void * p )  [inline]
```

Sets the current callback function and data for the widget.
Each widget has a single callback.

Parameters

in	*cb*	new callback
in	*p*	user data

13.153.4.21 callback() [5/6]

```
void Fl_Widget::callback (
            Fl_Callback0 * cb )  [inline]
```

Sets the current callback function for the widget.
Each widget has a single callback.

Parameters

in	*cb*	new callback

13.153.4.22 callback() [6/6]

```
void Fl_Widget::callback (
```

```
            Fl_Callback1 * cb,
            long p = 0 )  [inline]
```

Sets the current callback function for the widget.
Each widget has a single callback.

Parameters

in	*cb*	new callback
in	*p*	user data

13.153.4.23 changed()

```
unsigned int Fl_Widget::changed ( ) const  [inline]
```

Checks if the widget value changed since the last callback.
"Changed" is a flag that is turned on when the user changes the value stored in the widget. This is only used by subclasses of Fl_Widget that store values, but is in the base class so it is easier to scan all the widgets in a panel and do_callback() on the changed ones in response to an "OK" button.
Most widgets turn this flag off when they do the callback, and when the program sets the stored value.

Note

do_callback() turns this flag off after the callback.

Return values

0	if the value did not change

See also

set_changed(), clear_changed()

do_callback(Fl_Widget ∗widget, void ∗data)

13.153.4.24 clear_active()

```
void Fl_Widget::clear_active ( )  [inline]
```

Marks the widget as inactive without sending events or changing focus.
This is mainly for specialized use, for normal cases you want deactivate().

See also

deactivate()

13.153.4.25 clear_changed()

```
void Fl_Widget::clear_changed ( )  [inline]
```

Marks the value of the widget as unchanged.

See also

changed(), set_changed()

13.153.4.26 clear_damage()

```
void Fl_Widget::clear_damage (
            uchar c = 0 )  [inline]
```

Clears or sets the damage flags.
Damage flags are cleared when parts of the widget drawing is repaired.
The optional argument `c` specifies the bits that **are set** after the call (default: 0) and **not** the bits that are cleared!

Note

Therefore it is possible to set damage bits with this method, but this should be avoided. Use damage(uchar) instead.

Parameters

in	*c*	new bitmask of damage flags (default: 0)

See also

damage(uchar), damage()

13.153.4.27 clear_output()

```
void Fl_Widget::clear_output ( ) [inline]
```

Sets a widget to accept input.

See also

set_output(), output()

13.153.4.28 clear_visible()

```
void Fl_Widget::clear_visible ( ) [inline]
```

Hides the widget.
You must still redraw the parent to see a change in the window. Normally you want to use the hide() method instead.

13.153.4.29 clear_visible_focus()

```
void Fl_Widget::clear_visible_focus ( ) [inline]
```

Disables keyboard focus navigation with this widget.
Normally, all widgets participate in keyboard focus navigation.

See also

set_visible_focus(), visible_focus(), visible_focus(int)

13.153.4.30 color() [1/3]

```
Fl_Color Fl_Widget::color (
            void ) const [inline]
```

Gets the background color of the widget.

Returns

current background color

See also

color(Fl_Color), color(Fl_Color, Fl_Color)

13.153.4.31 color() [2/3]

```
void Fl_Widget::color (
            Fl_Color bg ) [inline]
```

Sets the background color of the widget.
The color is passed to the box routine. The color is either an index into an internal table of RGB colors or an RGB color value generated using fl_rgb_color().
The default for most widgets is FL_BACKGROUND_COLOR. Use Fl::set_color() to redefine colors in the color map.

Parameters

in	*bg*	background color

See also

color(), color(Fl_Color, Fl_Color), selection_color(Fl_Color)

13.153.4.32 color() [3/3]

```
void Fl_Widget::color (
            Fl_Color bg,
            Fl_Color sel )  [inline]
```

Sets the background and selection color of the widget.
The two color form sets both the background and selection colors.

Parameters

in	*bg*	background color
in	*sel*	selection color

See also

color(unsigned), selection_color(unsigned)

13.153.4.33 contains()

```
int Fl_Widget::contains (
            const Fl_Widget * w ) const
```

Checks if w is a child of this widget.

Parameters

in	*w*	potential child widget

Returns

Returns 1 if `w` is a child of this widget, or is equal to this widget. Returns 0 if `w` is NULL.

13.153.4.34 copy_label()

```
void Fl_Widget::copy_label (
            const char * new_label )
```

Sets the current label.
Unlike label(), this method allocates a copy of the label string instead of using the original string pointer.
The internal copy will automatically be freed whenever you assign a new label or when the widget is destroyed.

Parameters

in	*new_label*	the new label text

See also

label()

13.153.4.35 copy_tooltip()

```
void Fl_Widget::copy_tooltip (
            const char * text )
```

Sets the current tooltip text.
Unlike tooltip(), this method allocates a copy of the tooltip string instead of using the original string pointer.
The internal copy will automatically be freed whenever you assign a new tooltip or when the widget is destroyed.
If no tooltip is set, the tooltip of the parent is inherited. Setting a tooltip for a group and setting no tooltip for a child will show the group's tooltip instead. To avoid this behavior, you can set the child's tooltip to an empty string ("").

Parameters

in	*text*	New tooltip text (an internal copy is made and managed)

See also

tooltip(const char*), tooltip()

13.153.4.36 damage() [1/3]

```
uchar Fl_Widget::damage ( ) const  [inline]
```

Returns non-zero if draw() needs to be called.
The damage value is actually a bit field that the widget subclass can use to figure out what parts to draw.

Returns

a bitmap of flags describing the kind of damage to the widget

See also

damage(uchar), clear_damage(uchar)

13.153.4.37 damage() [2/3]

```
void Fl_Widget::damage (
            uchar c )
```

Sets the damage bits for the widget.
Setting damage bits will schedule the widget for the next redraw.

Parameters

in	*c*	bitmask of flags to set

See also

damage(), clear_damage(uchar)

13.153.4.38 damage() [3/3]

```
void Fl_Widget::damage (
            uchar c,
            int x,
            int y,
            int w,
            int h )
```

Sets the damage bits for an area inside the widget.
Setting damage bits will schedule the widget for the next redraw.

Parameters

in	*c*	bitmask of flags to set
in	*x,y,w,h*	size of damaged area

See also

damage(), clear_damage(uchar)

13.153.4.39 deactivate()

```
void Fl_Widget::deactivate ( )
```

Deactivates the widget.

Inactive widgets will be drawn "grayed out", e.g. with less contrast than the active widget. Inactive widgets will not receive any keyboard or mouse button events. Other events (including FL_ENTER, FL_MOVE, FL_LEAVE, FL_SHORTCUT, and others) will still be sent. A widget is only active if active() is true on it *and all of its parents*.

Changing this value will send FL_DEACTIVATE to the widget if active_r() is true.

Currently you cannot deactivate Fl_Window widgets.

See also

activate(), active(), active_r()

13.153.4.40 default_callback()

```
void Fl_Widget::default_callback (
            Fl_Widget * widget,
            void * data )  [static]
```

The default callback for all widgets that don't set a callback.

This callback function puts a pointer to the widget on the queue returned by Fl::readqueue(). This is the default for all widgets if you don't set a callback.

You can avoid the overhead of this default handling if you set the callback to `NULL` explicitly.

Relying on the default callback and reading the callback queue with Fl::readqueue() is not recommended. If you need a callback, you should set one with Fl_Widget::callback(Fl_Callback *cb, void *data) or one of its variants.

Parameters

in	*widget*	the Fl_Widget given to the callback
in	*data*	user data associated with that callback

See also

callback(), Fl::readqueue()

do_callback(Fl_Widget *widget, void *data)

13.153.4.41 deimage() [1/4]

```
Fl_Image * Fl_Widget::deimage ( )  [inline]
```

Gets the image that is used as part of the widget label when in the inactive state.

Returns

the current image for the deactivated widget

13.153.4.42 deimage() [2/4]

```
const Fl_Image * Fl_Widget::deimage ( ) const  [inline]
```

Gets the image that is used as part of the widget label when in the inactive state.

Returns

the current image for the deactivated widget

13.153.4.43 deimage() [3/4]

```
void Fl_Widget::deimage (
            Fl_Image & img )
```

Sets the image to use as part of the widget label when in the inactive state.

Parameters

in	*img*	the new image for the deactivated widget

See also

void deimage(Fl_Image* img)

13.153.4.44 deimage() [4/4]

```
void Fl_Widget::deimage (
            Fl_Image * img )
```

Sets the image to use as part of the widget label when in the inactive state.

Parameters

in	*img*	the new image for the deactivated widget

Note

The caller is responsible for making sure `img` is not deleted while it's used by the widget, and, if appropriate, for deleting it after the widget's deletion.

See also

void bind_deimage(Fl_Image* img)

13.153.4.45 deimage_bound()

```
int Fl_Widget::deimage_bound ( ) const  [inline]
```

Returns whether the inactive image is managed by the widget.

Return values

0	if the image is not bound to the widget
1	if the image will be deleted when the widget is deleted

See also

image_bound(), bind_deimage()

13.153.4.46 do_callback() [1/3]

```
void Fl_Widget::do_callback (
            Fl_Callback_Reason reason = FL_REASON_UNKNOWN )  [inline]
```

Calls the widget callback function with default arguments.
This is the same as calling
`do_callback(this, user_data(), reason);`

Parameters

in	*reason*	give a reason to why this callback was called, defaults to FL_REASON_UNKNOWN

See also

callback()

do_callback(Fl_Widget *widget, void *data, Fl_Callback_Reason reason), Fl_Callback_Reason

13.153.4.47 do_callback() [2/3]

```
void Fl_Widget::do_callback (
            Fl_Widget * widget,
            long arg,
            Fl_Callback_Reason reason = FL_REASON_UNKNOWN )  [inline]
```

Calls the widget callback function with arbitrary arguments.

Parameters

in	*widget*	call the callback with `widget` as the first argument
in	*arg*	call the callback with `arg` as the user data (second) argument
in	*reason*	give a reason to why this callback was called, defaults to FL_REASON_UNKNOWN

See also

callback()

do_callback(Fl_Widget *widget, void *data), Fl_Callback_Reason

13.153.4.48 do_callback() [3/3]

```
void Fl_Widget::do_callback (
            Fl_Widget * widget,
            void * arg = 0,
            Fl_Callback_Reason reason = FL_REASON_UNKNOWN )
```

Calls the widget callback function with arbitrary arguments.
All overloads of do_callback() call this method. It does nothing if the widget's callback() is NULL. It clears the widget's *changed* flag **after** the callback was called unless the callback is the default callback. Hence it is not necessary to call clear_changed() after calling do_callback() in your own widget's handle() method.
A `reason` must be set for widgets if different actions can trigger the same callback.

Note

It is legal to delete the widget in the callback (i.e. in user code), but you must not access the widget in the handle() method after calling do_callback() if the widget was deleted in the callback. We recommend to use Fl_Widget_Tracker to check whether the widget was deleted in the callback.

Parameters

in	*widget*	call the callback with `widget` as the first argument
in	*arg*	use `arg` as the user data (second) argument
in	*reason*	give a reason to why this callback was called, defaults to FL_REASON_UNKNOWN

See also

default_callback()

callback()

class Fl_Widget_Tracker

Fl::callback_reason()

13.153.4.49 draw()

```
virtual void Fl_Widget::draw ( ) [pure virtual]
```

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();                 // calls Fl_Scrollbar::draw()
```

Implemented in Fl_Adjuster, Fl_Box, Fl_Browser_, Fl_Button, Fl_Cairo_Window, Fl_Chart, Fl_Choice, Fl_Clock_Output, Fl_Counter, Fl_Dial, Fl_File_Input, Fl_Flex, Fl_FormsBitmap, Fl_FormsPixmap, Fl_Free, Fl_Gl_Window, Fl_Grid, Fl_Group, Fl_Help_View, Fl_Input, Fl_Input_Choice, Fl_Light_Button, Fl_Menu_Bar, Fl_Menu_Button, Fl_Pack, Fl_Positioner, Fl_Progress, Fl_Return_Button, Fl_Roller, Fl_Scroll, Fl_Scrollbar, Fl_Shortcut_Button, Fl_Slider, Fl_Spinner, Fl_Sys_Menu_Bar, Fl_Table, Fl_Tabs, Fl_Text_Display, Fl_Timer, Fl_Tree, Fl_Value_Input, Fl_Value_Output, Fl_Value_Slider, Fl_Window, Fl_Wizard, Fl_FormsText, Fl_Glut_Window, and Fl_Terminal.

13.153.4.50 draw_focus() [1/3]

```
void Fl_Widget::draw_focus ( ) const [inline], [protected]
```

Draws a focus rectangle around the widget.

This method uses the widget's boxtype and coordinates and its background color color().

See also

Fl_Widget::draw_focus(Fl_Boxtype, int, int, int, int, Fl_Color) const

13.153.4.51 draw_focus() [2/3]

```
void Fl_Widget::draw_focus (
            Fl_Boxtype t,
            int X,
            int Y,
            int W,
            int H ) const [inline], [protected]
```

Draws a focus rectangle around the widget.

This method uses the given boxtype and coordinates and the widget's background color color().

See also

Fl_Widget::draw_focus(Fl_Boxtype, int, int, int, int, Fl_Color) const

13.153.4.52 draw_focus() [3/3]

```
void Fl_Widget::draw_focus (
            Fl_Boxtype bt,
            int X,
            int Y,
            int W,
```

```
            int H,
            Fl_Color bg ) const  [protected]
```

Draws a focus box for the widget at the given position and size.
This method does nothing if

- the global option Fl::visible_focus() or
- the per-widget option visible_focus() is false (off).

This means that Fl_Widget::draw_focus() or one of the more specialized methods can be called without checking these visible focus options.

Note

This method must only be called if the widget has the focus. This is not tested internally.

The boxtype `bt` is used to calculate the inset so the focus box is drawn inside the box borders.
The default focus box drawing color is black. The background color `bg` is used to determine a better visible color if necessary by using fl_contrast() with the given background color.

Parameters

in	*bt*	Boxtype that needs to be considered (frame width)
in	*X,Y,W,H*	Bounding box
in	*bg*	Background color

See also

Fl_Widget::draw_focus()

Fl_Widget::draw_focus(Fl_Boxtype, int, int, int, int) const

13.153.4.53 draw_label() [1/3]

```
void Fl_Widget::draw_label ( ) const  [protected]
```

Draws the widget's label at the defined label position.
This is the normal call for a widget's draw() method.

13.153.4.54 draw_label() [2/3]

```
void Fl_Widget::draw_label (
            int X,
            int Y,
            int W,
            int H ) const  [protected]
```

Draws the label in an arbitrary bounding box.
draw() can use this instead of draw_label(void) to change the bounding box

13.153.4.55 draw_label() [3/3]

```
void Fl_Widget::draw_label (
            int X,
            int Y,
            int W,
            int H,
            Fl_Align a ) const
```

Draws the label in an arbitrary bounding box with an arbitrary alignment.
Anybody can call this to force the label to draw anywhere.

13.153.4.56 h() [1/2]

```
int Fl_Widget::h ( ) const  [inline]
```

Gets the widget height.

Returns

the height of the widget in pixels.

13.153.4.57 h() [2/2]

```
void Fl_Widget::h (
            int v )  [inline], [protected]
```

Internal use only.
Use position(int,int), size(int,int) or resize(int,int,int,int) instead.

13.153.4.58 handle()

```
int Fl_Widget::handle (
            int event )  [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to *override* the behavior of the base class. This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to *filter* event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you *consumed* the event by returning 1 even if you didn't *do* anything with it.

Parameters

in	*event*	the kind of event received

Return values

0	if the event was not used or understood
1	if the event was used and can be deleted

See also

Fl_Event

Reimplemented in Fl_Color_Chooser, Fl_Free, Fl_Table, Fl_Terminal, Fl_Text_Display, Fl_Text_Editor, Fl_Tree, Fl_Browser_, Fl_File_Input, Fl_Scheme_Choice, Fl_Spinner::Fl_Spinner_Input, Fl_Spinner, Fl_Table_Row, Fl_Tile, Fl_Adjuster, Fl_Box, Fl_Button, Fl_Check_Browser, Fl_Choice, Fl_Clock, Fl_Counter, Fl_Dial, Fl_Gl_Window, Fl_Group, Fl_Help_View, Fl_Input, Fl_Light_Button, Fl_Menu_Bar, Fl_Menu_Button, Fl_Positioner, Fl_Repeat_Button, Fl_Return_Button, Fl_Roller, Fl_Scroll, Fl_Scrollbar, Fl_Secret_Input, Fl_Shortcut_Button, Fl_Slider, Fl_Tabs, Fl_Timer, Fl_Value_Input, Fl_Value_Output, Fl_Value_Slider, Fl_Window, and Fl_Glut_Window.

13.153.4.59 hide()

```
void Fl_Widget::hide ( )  [virtual]
```

Makes a widget invisible.

See also

show(), visible(), visible_r()

Reimplemented in Fl_Browser, Fl_Double_Window, Fl_Gl_Window, Fl_Overlay_Window, and Fl_Window.

13.153.4.60 horizontal_label_margin() [1/2]

```
int Fl_Widget::horizontal_label_margin ( ) [inline]
```

Get the spacing between the label and the horizontal edge of the widget.

Returns

px gap in pixels

13.153.4.61 horizontal_label_margin() [2/2]

```
void Fl_Widget::horizontal_label_margin (
            int px ) [inline]
```

Set the spacing between the label and the horizontal edge of the widget.

Parameters

in	*px*	gap in pixels

13.153.4.62 image() [1/4]

```
Fl_Image * Fl_Widget::image ( ) [inline]
```

Gets the image that is used as part of the widget label when in the active state.

Returns

the current image

13.153.4.63 image() [2/4]

```
const Fl_Image * Fl_Widget::image ( ) const [inline]
```

Gets the image that is used as part of the widget label when in the active state.

Returns

the current image

13.153.4.64 image() [3/4]

```
void Fl_Widget::image (
            Fl_Image & img )
```

Sets the image to use as part of the widget label when in the active state.

Parameters

in	*img*	the new image for the label

See also

void image(Fl_Image* img)

13.153.4.65 image() [4/4]

```
void Fl_Widget::image (
            Fl_Image * img )
```

Sets the image to use as part of the widget label when in the active state.
The caller is responsible for making sure `img` is not deleted while it's used by the widget, and, if appropriate, for deleting it after the widget's deletion.
Calling image() with a new image will delete the old image if it was bound, and set the new image without binding it.
If old and new are the same, the image will not be deleted, but it will be unbound.
Calling image() with NULL will delete the old image if it was bound and not set a new image.

Parameters

in	*img*	the new image for the label

See also

bind_image(Fl_Image* img)

13.153.4.66 image_bound()

```
int Fl_Widget::image_bound ( ) const  [inline]
```

Returns whether the image is managed by the widget.

Return values

0	if the image is not bound to the widget
1	if the image will be deleted when the widget is deleted

See also

deimage_bound(), bind_image()

13.153.4.67 inside()

```
int Fl_Widget::inside (
            const Fl_Widget * wgt ) const  [inline]
```

Checks if this widget is a child of `wgt`.
Returns 1 if this widget is a child of `wgt`, or is equal to `wgt`. Returns 0 if `wgt` is NULL.

Parameters

in	*wgt*	the possible parent widget.

See also

contains()

13.153.4.68 is_label_copied()

```
int Fl_Widget::is_label_copied ( ) const  [inline]
```

Returns whether the current label was assigned with copy_label().
This can be useful for temporarily overwriting the widget's label and restoring it later.

Return values

0	current label was assigned with label().
1	current label was assigned with copy_label().

13.153.4.69 label() [1/3]

```
const char * Fl_Widget::label ( ) const  [inline]
```

Gets the current label text.

Returns

a pointer to the current label text

See also

label(const char *), copy_label(const char *)

13.153.4.70 label() [2/3]

```
void Fl_Widget::label (
            const char * text )
```

Sets the current label pointer.
The label is shown somewhere on or next to the widget. See Labels and Label Types for details about what can be put in a label. The passed pointer is stored unchanged in the widget (the string is *not* copied), so if you need to set the label to a formatted value, make sure the buffer is static, global, or allocated. The copy_label() method can be used to make a copy of the label string automatically.

Parameters

in	*text*	pointer to new label text

See also

copy_label()

13.153.4.71 label() [3/3]

```
void Fl_Widget::label (
            Fl_Labeltype a,
            const char * b )  [inline]
```

Shortcut to set the label text and type in one call.

See also

label(const char *), labeltype(Fl_Labeltype)

13.153.4.72 label_image_spacing() [1/2]

```
int Fl_Widget::label_image_spacing ( )  [inline]
```

Return the gap size between the label and the image.

Returns

spacing in pixels

13.153.4.73 label_image_spacing() [2/2]

```
void Fl_Widget::label_image_spacing (
            int gap ) [inline]
```

Set the gap between the label and the image in pixels.
This value is limited to 0..255.

Parameters

in	*gap*	spacing in pixels

13.153.4.74 label_shortcut()

```
unsigned int Fl_Widget::label_shortcut (
            const char * t ) [static]
```

Returns the Unicode value of the '&x' shortcut in a given text.
The given text `t` (usually a widget's label or a menu text) is searched for a '&x' shortcut label, and if found, the Unicode value (code point) of the '&x' shortcut is returned.

Parameters

t	text or label to search for '&x' shortcut.

Returns

Unicode (UCS-4) value of shortcut in `t` or 0.

Note

Internal use only.

13.153.4.75 labelcolor() [1/2]

```
Fl_Color Fl_Widget::labelcolor ( ) const [inline]
```

Gets the label color.
The default color is FL_FOREGROUND_COLOR.

Returns

the current label color

13.153.4.76 labelcolor() [2/2]

```
void Fl_Widget::labelcolor (
            Fl_Color c ) [inline]
```

Sets the label color.
The default color is FL_FOREGROUND_COLOR.

Parameters

in	*c*	the new label color

13.153.4.77 labelfont() [1/2]

```
Fl_Font Fl_Widget::labelfont ( ) const [inline]
```

Gets the font to use.

Fonts are identified by indexes into a table. The default value uses a Helvetica typeface (Arial for Microsoft® Windows®). The function Fl::set_font() can define new typefaces.

Returns

current font used by the label

See also

Fl_Font

13.153.4.78 labelfont() [2/2]

```
void Fl_Widget::labelfont (
            Fl_Font f )  [inline]
```

Sets the font to use.
Fonts are identified by indexes into a table. The default value uses a Helvetica typeface (Arial for Microsoft® Windows®). The function Fl::set_font() can define new typefaces.

Parameters

in	*f*	the new font for the label

See also

Fl_Font

13.153.4.79 labelsize() [1/2]

```
Fl_Fontsize Fl_Widget::labelsize ( ) const  [inline]
```

Gets the font size in pixels.
The default size is 14 pixels.

Returns

the current font size

13.153.4.80 labelsize() [2/2]

```
void Fl_Widget::labelsize (
            Fl_Fontsize pix )  [inline]
```

Sets the font size in pixels.

Parameters

in	*pix*	the new font size

See also

Fl_Fontsize labelsize()

13.153.4.81 labeltype() [1/2]

```
Fl_Labeltype Fl_Widget::labeltype ( ) const  [inline]
```

Gets the label type.

Returns

the current label type.

See also

Fl_Labeltype

13.153.4.82 labeltype() [2/2]

```
void Fl_Widget::labeltype (
            Fl_Labeltype a ) [inline]
```

Sets the label type.
The label type identifies the function that draws the label of the widget. This is generally used for special effects such as embossing or for using the label() pointer as another form of data such as an icon. The value FL_NORMAL_↩ LABEL prints the label as plain text.

Parameters

in	*a*	new label type

See also

Fl_Labeltype

13.153.4.83 measure_label()

```
void Fl_Widget::measure_label (
            int & ww,
            int & hh ) const [inline]
```

Sets width ww and height hh accordingly with the label size.
Labels with images will return w() and h() of the image.
This calls fl_measure() internally. For more information about the arguments `ww` and `hh` and word wrapping

See also

fl_measure(const char*, int&, int&, int)

13.153.4.84 needs_keyboard() [1/2]

```
bool Fl_Widget::needs_keyboard ( ) const [inline]
```

Returns whether this widget needs a keyboard.

Returns

true or false

See also

needs_keyboard(bool)

13.153.4.85 needs_keyboard() [2/2]

```
void Fl_Widget::needs_keyboard (
            bool needs ) [inline]
```

Sets whether this widget needs a keyboard.
Set this on touch screen devices if a widget needs a keyboard when it gets the focus.

Note

This flag can be set but is not yet **used** in FLTK 1.4.0. It is intended to be used in the future on real touch devices.

Parameters

in	*needs*	Set this to true or false

13.153.4.86 output()

```
unsigned int Fl_Widget::output ( ) const  [inline]
```

Returns if a widget is used for output only.
output() means the same as !active() except it does not change how the widget is drawn. The widget will not receive any events. This is useful for making scrollbars or buttons that work as displays rather than input devices.

Return values

0	if the widget is used for input and output

See also

set_output(), clear_output()

13.153.4.87 parent() [1/2]

```
Fl_Group * Fl_Widget::parent ( ) const  [inline]
```

Returns a pointer to the parent widget.
Usually this is a Fl_Group or Fl_Window.

Return values

NULL	if the widget has no parent

See also

Fl_Group::add(Fl_Widget*)

13.153.4.88 parent() [2/2]

```
void Fl_Widget::parent (
            Fl_Group * p )  [inline]
```

Internal use only - "for hacks only".
It is ***STRONGLY*** *recommended* not to use this method, because it short-circuits Fl_Group's normal widget adding and removing methods, if the widget is already a child widget of another Fl_Group.
Use Fl_Group::add(Fl_Widget*) and/or Fl_Group::remove(Fl_Widget*) instead.

13.153.4.89 position()

```
void Fl_Widget::position (
            int X,
            int Y )  [inline]
```

Repositions the window or widget.
position(X, Y) is a shortcut for resize(X, Y, w(), h()).

Parameters

in	*X,Y*	new position relative to the parent window

See also

resize(int,int,int,int), size(int,int)

13.153.4.90 redraw()

```
void Fl_Widget::redraw ( )
```

Schedules the drawing of the widget.
Marks the widget as needing its draw() routine called.

13.153.4.91 redraw_label()

```
void Fl_Widget::redraw_label ( )
```

Schedules the drawing of the label.
Marks the widget or the parent as needing a redraw for the label area of a widget.

13.153.4.92 resize()

```
void Fl_Widget::resize (
            int x,
            int y,
            int w,
            int h )  [virtual]
```

Changes the size or position of the widget.
This is a virtual function so that the widget may implement its own handling of resizing. The default version does *not* call the redraw() method, but instead relies on the parent widget to do so because the parent may know a faster way to update the display, such as scrolling from the old position.
Some window managers under X11 call resize() a lot more often than needed. Please verify that the position or size of a widget did actually change before doing any extensive calculations.
position(X, Y) is a shortcut for resize(X, Y, w(), h()), and size(W, H) is a shortcut for resize(x(), y(), W, H).

Parameters

in	*x,y*	new position relative to the parent window
in	*w,h*	new size

See also

position(int,int), size(int,int)

Reimplemented in Fl_Browser_, Fl_Flex, Fl_Grid, Fl_Input_Choice, Fl_Pack, Fl_Scroll, Fl_Spinner, Fl_Table, Fl_Terminal, Fl_Text_Display, Fl_Tile, Fl_Window, Fl_Double_Window, Fl_Gl_Window, Fl_Group, Fl_Help_View, Fl_Input_, Fl_Overlay_Window, Fl_Tabs, Fl_Tree, and Fl_Value_Input.

13.153.4.93 selection_color() [1/2]

```
Fl_Color Fl_Widget::selection_color ( ) const  [inline]
```

Gets the selection color.

Returns

the current selection color

See also

selection_color(Fl_Color), color(Fl_Color, Fl_Color)

13.153.4.94 selection_color() [2/2]

```
void Fl_Widget::selection_color (
            Fl_Color a ) [inline]
```

Sets the selection color.

The selection color is defined for Forms compatibility and is usually used to color the widget when it is selected, although some widgets use this color for other purposes. You can set both colors at once with color(Fl_Color bg, Fl_Color sel).

Parameters

in	*a*	the new selection color

See also

selection_color(), color(Fl_Color, Fl_Color)

13.153.4.95 send()

```
int Fl_Widget::send (
            int event )
```

Sends the specified event to the widget.

This is a wrapper for handle(), used by groups to send events to child widgets. If the target is a subwindow, `send` updates mouse coordinates and does some additional event processing before calling handle().

Parameters

in	*event*	the kind of event to send

Returns

the return value of handle(event)

13.153.4.96 set_active()

```
void Fl_Widget::set_active ( ) [inline]
```

Marks the widget as active without sending events or changing focus.

This is mainly for specialized use, for normal cases you want activate().

See also

activate()

13.153.4.97 set_changed()

```
void Fl_Widget::set_changed ( ) [inline]
```

Marks the value of the widget as changed.

See also

changed(), clear_changed()

13.153.4.98 set_output()

```
void Fl_Widget::set_output ( ) [inline]
```

Sets a widget to output only.

See also

output(), clear_output()

13.153.4.99 set_visible()

```
void Fl_Widget::set_visible ( ) [inline]
```

Makes the widget visible.
You must still redraw the parent widget to see a change in the window. Normally you want to use the show() method instead.

13.153.4.100 set_visible_focus()

```
void Fl_Widget::set_visible_focus ( ) [inline]
```

Enables keyboard focus navigation with this widget.
Note, however, that this will not necessarily mean that the widget will accept focus, but for widgets that can accept focus, this method enables it if it has been disabled.

See also

visible_focus(), clear_visible_focus(), visible_focus(int)

13.153.4.101 shortcut_label() [1/2]

```
int Fl_Widget::shortcut_label ( ) const [inline]
```

Returns whether the widget's label uses '&' to indicate shortcuts.

See also

void shortcut_label(int value)

13.153.4.102 shortcut_label() [2/2]

```
void Fl_Widget::shortcut_label (
            int value ) [inline]
```

Sets whether the widget's label uses '&' to indicate shortcuts.
By default, all objects of classes Fl_Menu_ (and derivatives), Fl_Button (and derivatives), Fl_Text_Display, Fl_Value_Input, and Fl_Input_ (and derivatives) use character '&' in their label, unless '&' is repeated, to indicate shortcuts: '&' does not appear in the drawn label, the next character after '&' in the label is drawn underlined, and typing this character triggers the corresponding menu window, button, or other widget. If the label contains 2 consecutive '&', only one is drawn and the next character is not underlined and not used as a shortcut. If `value` is set to 0, all these labels don't process character '&' as indicating a shortcut: '&' is drawn in the label, the next character is not underlined and does not define a shortcut.

13.153.4.103 show()

```
void Fl_Widget::show ( ) [virtual]
```

Makes a widget visible.
An invisible widget never gets redrawn and does not get keyboard or mouse events, but can receive a few other events like FL_SHOW.
The visible() method returns true if the widget is set to be visible. The visible_r() method returns true if the widget and all of its parents are visible. A widget is only visible if visible() is true on it *and all of its parents*.
Changing it will send FL_SHOW or FL_HIDE events to the widget. *Do not change it if the parent is not visible, as this will send false FL_SHOW or FL_HIDE events to the widget.* redraw() is called if necessary on this or the parent.

See also

hide(), visible(), visible_r()

Reimplemented in Fl_Browser, Fl_Double_Window, Fl_Gl_Window, Fl_Overlay_Window, Fl_Single_Window, Fl_Tabs, and Fl_Window.

13.153.4.104 size()

```
void Fl_Widget::size (
            int W,
            int H )  [inline]
```

Changes the size of the widget.
size(W, H) is a shortcut for resize(x(), y(), W, H).

Parameters

in	*W,H*	new size

See also

position(int,int), resize(int,int,int,int)

13.153.4.105 take_focus()

```
int Fl_Widget::take_focus ( )
```

Gives the widget the keyboard focus.
Tries to make this widget be the Fl::focus() widget, by first sending it an FL_FOCUS event, and if it returns non-zero, setting Fl::focus() to this widget. You should use this method to assign the focus to a widget.

Returns

true if the widget accepted the focus.

13.153.4.106 takesevents()

```
unsigned int Fl_Widget::takesevents ( ) const  [inline]
```

Returns if the widget is able to take events.
This is the same as (active() && !output() && visible()) but is faster.

Return values

0	if the widget takes no events

13.153.4.107 test_shortcut() [1/2]

```
int Fl_Widget::test_shortcut ( )
```

Returns true if the widget's label contains the entered '&x' shortcut.
This method must only be called in handle() methods or callbacks after a keypress event (usually FL_KEYDOWN or FL_SHORTCUT). The widget's label is searched for a '&x' shortcut, and if found, this is compared with the entered key value.
Fl::event_text() is used to get the entered key value.

Returns

true, if the entered text matches the widget's'&x' shortcut, false (0) otherwise.

Note

Useful when a widget's handle(int) method needs dedicated processing of FL_SHORTCUT.

13.153.4.108 test_shortcut() [2/2]

```
int Fl_Widget::test_shortcut (
            const char * t,
            const bool require_alt = false )  [static]
```

Returns true if the given text `t` contains the entered '&x' shortcut.
This method must only be called in handle() methods or callbacks after a keypress event (usually FL_KEYDOWN or FL_SHORTCUT). The given text `t` (usually a widget's label or menu text) is searched for a '&x' shortcut, and if found, this is compared with the entered key value.
Fl::event_text() is used to get the entered key value. Fl::event_state() is used to get the Alt modifier, if `require_alt` is true.

Parameters

t	text or label to search for '&x' shortcut.
require_alt	if true: match only if Alt key is pressed.

Returns

true, if the entered text matches the '&x' shortcut in `t` false (0) otherwise.

Note

Useful when a widget's handle(int) method needs dedicated processing of FL_SHORTCUT.

13.153.4.109 tooltip() [1/2]

```
const char * Fl_Widget::tooltip ( ) const  [inline]
```

Gets the current tooltip text.

Returns

a pointer to the tooltip text or NULL

See also

tooltip(const char*), copy_tooltip(const char*)

13.153.4.110 tooltip() [2/2]

```
void Fl_Widget::tooltip (
            const char * text )
```

Sets the Tooltip Text for a Widget.
Assigns a tooltip string that appears in a popup when the user hovers over the widget. The provided string is not copied. The caller must eensure it remains valid by storing it in a static, global, or dynamically allocated buffer. If you need the tooltip string to be copied and managed automatically, use copy_tooltip().
By default, a widget inherits the tooltip of its parent if none is explicitly set. If you assign a tooltip to a group but not to its child widgets, the child widgets will display the group's tooltip. To prevent inheritance, set the child's tooltip to an empty string ("").
Tooltips can be updated dynamically before they are displayed. When a tooltip is about to be shown, FLTK sends an `FL_BEFORE_TOOLTIP` event to the widget's `handle()` method. Developers can override the tooltip text temporarily using `Fl_Tooltip::override_text(const char* new_text)` and returning 1 from `handle()` to apply the change.

Parameters

in	*text*	New tooltip text (no copy is made)

See also

copy_tooltip(const char*), tooltip()

13.153.4.111 top_window()

```
Fl_Window * Fl_Widget::top_window ( ) const
```

Returns a pointer to the top-level window for the widget.
In other words, the 'window manager window' that contains this widget. This method differs from window() in that it won't return sub-windows (if there are any).

Returns

the top-level window, or NULL if no top-level window is associated with this widget.

See also

window()

13.153.4.112 top_window_offset()

```
Fl_Window * Fl_Widget::top_window_offset (
            int & xoff,
            int & yoff ) const
```

Finds the x/y offset of the current widget relative to the top-level window.

Parameters

out	*xoff,yoff*	Returns the x/y offset

Returns

the top-level window (or NULL for a widget that's not in any window)

13.153.4.113 type() [1/2]

```
uchar Fl_Widget::type ( ) const  [inline]
```

Gets the widget type.
Returns the widget type value, which gives some information about the derived widget class to which the object belongs. Noticeably, the condition `type() >= FL_WINDOW` indicates a widget is an Fl_Window or derived object.

Note

Known values for type_ are listed at the top of src/Fl_Widget.cxx .

13.153.4.114 type() [2/2]

```
void Fl_Widget::type (
            uchar t )  [inline]
```

Sets the widget type.

See also

type()

13.153.4.115 user_data()

```
void * Fl_Widget::user_data ( ) const  [inline]
```

Gets the user data for this widget.
Gets the current user data (void *) argument that is passed to the callback function.

Returns

user data as a pointer

13.153.4.116 vertical_label_margin() [1/2]

```
int Fl_Widget::vertical_label_margin ( ) [inline]
```

Get the spacing between the label and the vertical edge of the widget.

Returns

px gap in pixels

13.153.4.117 vertical_label_margin() [2/2]

```
void Fl_Widget::vertical_label_margin (
            int px ) [inline]
```

Set the spacing between the label and the vertical edge of the widget.

Parameters

in	*px*	gap in pixels

13.153.4.118 visible()

```
unsigned int Fl_Widget::visible ( ) const [inline]
```

Returns whether a widget is visible.

Return values

0	if the widget is not drawn and hence invisible.

See also

show(), hide(), visible_r()

13.153.4.119 visible_focus() [1/2]

```
unsigned int Fl_Widget::visible_focus ( ) const [inline]
```

Checks whether this widget has a visible focus.

Return values

0	if this widget has no visible focus.

See also

visible_focus(int), set_visible_focus(), clear_visible_focus()

13.153.4.120 visible_focus() [2/2]

```
void Fl_Widget::visible_focus (
            int v ) [inline]
```

Modifies keyboard focus navigation.

Parameters

in	*v*	set or clear visible focus

See also

set_visible_focus(), clear_visible_focus(), visible_focus()

13.153.4.121 visible_r()

```
int Fl_Widget::visible_r ( ) const
```

Returns whether a widget and all its parents are visible.

Return values

0	if the widget or any of its parents are invisible.

See also

show(), hide(), visible()

13.153.4.122 w() [1/2]

```
int Fl_Widget::w ( ) const  [inline]
```

Gets the widget width.

Returns

the width of the widget in pixels.

13.153.4.123 w() [2/2]

```
void Fl_Widget::w (
            int v )  [inline], [protected]
```

Internal use only.
Use position(int,int), size(int,int) or resize(int,int,int,int) instead.

13.153.4.124 when() [1/2]

```
Fl_When Fl_Widget::when ( ) const  [inline]
```

Returns the conditions under which the callback is called.
You can set the flags with when(uchar), the default value is FL_WHEN_RELEASE.

Returns

set of flags

See also

when(uchar), Fl_When, do_callback(), Fl::callback_reason()

13.153.4.125 when() [2/2]

```
void Fl_Widget::when (
            uchar i )  [inline]
```

Sets the flags used to decide when a callback is called.
This controls when callbacks are done. The following values are useful, the default value is FL_WHEN_RELEASE:

- 0: The callback is not done, but changed() is turned on.
- FL_WHEN_CHANGED: The callback is done each time the text is changed by the user.

- FL_WHEN_RELEASE: The callback will be done when this widget loses the focus, including when the window is unmapped. This is a useful value for text fields in a panel where doing the callback on every change is wasteful. However the callback will also happen if the mouse is moved out of the window, which means it should not do anything visible (like pop up an error message). You might do better setting this to zero, and scanning all the items for changed() when the OK button on a panel is pressed.
- FL_WHEN_ENTER_KEY: If the user types the Enter key, the entire text is selected, and the callback is done if the text has changed. Normally the Enter key will navigate to the next field (or insert a newline for a Fl_Multiline_Input) - this changes the behavior.
- FL_WHEN_ENTER_KEY|FL_WHEN_NOT_CHANGED: The Enter key will do the callback even if the text has not changed. Useful for command fields.
- FL_WHEN_CLOSED : If the user requests that the widget is closed, the callback is called with FL_REASON_CLOSED. The Fl_Tabs widget checks this flag on its children to determine whether to display a close button on the tab of that widget.

Fl_Widget::when() is a set of bitflags used by subclasses of Fl_Widget to decide when to do the callback.
If the value is zero then the callback is never done. Other values are described in the individual widgets. This field is in the base class so that you can scan a panel and do_callback() on all the ones that don't do their own callbacks in response to an "OK" button.

Parameters

in	*i*	set of flags

See also

Fl_When, do_callback(), Fl::callback_reason()

13.153.4.126 window()

`Fl_Window * Fl_Widget::window ( ) const`

Returns a pointer to the nearest parent window up the widget hierarchy.
This will return sub-windows if there are any, or the parent window if there's no sub-windows. If this widget IS the top-level window, NULL is returned.

Return values

NULL	if no window is associated with this widget.

Note

for an Fl_Window widget, this returns its *parent* window (if any), not *this* window.

See also

top_window()

13.153.4.127 x() [1/2]

`int Fl_Widget::x ( ) const [inline]`

Gets the widget position in its window.

Returns

the x position relative to the window

13.153.4.128 x() [2/2]

```
void Fl_Widget::x (
            int v )  [inline], [protected]
```

Internal use only.
Use position(int,int), size(int,int) or resize(int,int,int,int) instead.

13.153.4.129 y() [1/2]

```
int Fl_Widget::y ( ) const  [inline]
```

Gets the widget position in its window.

Returns

the y position relative to the window

13.153.4.130 y() [2/2]

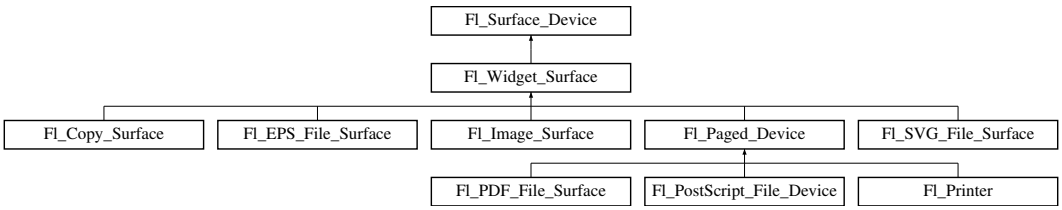
```
void Fl_Widget::y (
            int v )  [inline], [protected]
```

Internal use only.
Use position(int,int), size(int,int) or resize(int,int,int,int) instead.
The documentation for this class was generated from the following files:

- Fl_Widget.H
- Fl.cxx
- fl_boxtype.cxx
- Fl_Group.cxx
- fl_labeltype.cxx
- fl_shortcut.cxx
- Fl_Tooltip.cxx
- Fl_Widget.cxx
- Fl_Window.cxx

13.154 Fl_Widget_Surface Class Reference

A surface on which any FLTK widget can be drawn.
`#include <Fl_Widget_Surface.H>`
Inheritance diagram for Fl_Widget_Surface:



Public Member Functions

- void draw (Fl_Widget *widget, int delta_x=0, int delta_y=0)

  Draws the widget on the drawing surface.
- void draw_decorated_window (Fl_Window *win, int x_offset=0, int y_offset=0)

  Draws a window with its title bar and frame if any.
- virtual void origin (int *x, int *y)

  Computes the coordinates of the current origin of graphics functions.
- virtual void origin (int x, int y)

Sets the position of the origin of graphics in the drawable part of the drawing surface.

- void print_window_part (Fl_Window *win, int x, int y, int w, int h, int delta_x=0, int delta_y=0)

  Draws a rectangular part of an on-screen window.
- virtual int printable_rect (int *w, int *h)

  Computes the width and height of the drawable area of the drawing surface.
- virtual void translate (int x, int y)

  Translates the current graphics origin accounting for the current rotation.
- virtual void untranslate ()

  Undoes the effect of a previous translate() call.

Public Member Functions inherited from Fl_Surface_Device

- Fl_Graphics_Driver * **driver** ()

  Returns the graphics driver of this drawing surface.
- virtual bool is_current ()

  Is this surface the current drawing surface?
- virtual void set_current (void)

  Make this surface the current drawing surface.
- virtual ~**Fl_Surface_Device** ()

  The destructor.

Protected Member Functions

- Fl_Widget_Surface (Fl_Graphics_Driver *d)

  The constructor.

Protected Member Functions inherited from Fl_Surface_Device

- void **driver** (Fl_Graphics_Driver *graphics_driver)

  Sets the graphics driver of this drawing surface.
- virtual void end_current ()

  FLTK calls this each time a surface ceases to be the current drawing surface.
- **Fl_Surface_Device** (Fl_Graphics_Driver *graphics_driver)

  Constructor that sets the graphics driver to use for the created surface.

Protected Attributes

- int **x_offset**

  horizontal offset to the origin of graphics coordinates
- int **y_offset**

  vertical offset to the origin of graphics coordinates

Additional Inherited Members

Static Public Member Functions inherited from Fl_Surface_Device

- static Fl_Surface_Device * pop_current ()

  Removes the top element from the current drawing surface stack, and makes the new top element current.
- static void push_current (Fl_Surface_Device *new_current)

  Pushes `new_current` on top of the stack of current drawing surfaces, and makes it current.
- static Fl_Surface_Device * surface ()

  The current drawing surface.

13.154.1 Detailed Description

A surface on which any FLTK widget can be drawn.

13.154.2 Constructor & Destructor Documentation

13.154.2.1 Fl_Widget_Surface()

```
Fl_Widget_Surface::Fl_Widget_Surface (
            Fl_Graphics_Driver * d )  [protected]
```

The constructor.

Parameters

d	can be nul.

13.154.3 Member Function Documentation

13.154.3.1 draw()

```
void Fl_Widget_Surface::draw (
            Fl_Widget * widget,
            int delta_x = 0,
            int delta_y = 0 )
```

Draws the widget on the drawing surface.
The widget's position on the surface is determined by the last call to origin() and by the optional delta_x and delta_y arguments. Its dimensions are in points unless there was a previous call to scale().

Parameters

in	*widget*	Any FLTK widget (e.g., standard, custom, window).
in	*delta_x,delta↩_y*	Optional horizontal and vertical offsets for positioning the widget top left relatively to the current origin of graphics.

13.154.3.2 draw_decorated_window()

```
void Fl_Widget_Surface::draw_decorated_window (
            Fl_Window * win,
            int win_offset_x = 0,
            int win_offset_y = 0 )
```

Draws a window with its title bar and frame if any.
`win_offset_x` and `win_offset_y` are optional coordinates of where to position the window top left. Equivalent to draw() if `win` is a subwindow or has no border. Use Fl_Window::decorated_w() and Fl_Window::decorated_h() to get the size of the framed window.

13.154.3.3 origin() [1/2]

```
void Fl_Widget_Surface::origin (
            int * x,
            int * y )  [virtual]
```

Computes the coordinates of the current origin of graphics functions.

Parameters

out	*x,y*	If non-null, ∗x and ∗y are set to the horizontal and vertical coordinates of the graphics origin.

Reimplemented in Fl_EPS_File_Surface, Fl_Copy_Surface, Fl_Image_Surface, Fl_PDF_File_Surface, Fl_PostScript_File_Device,

Fl_Printer, and Fl_SVG_File_Surface.

13.154.3.4 origin() [2/2]

```
void Fl_Widget_Surface::origin (
            int x,
            int y )  [virtual]
```

Sets the position of the origin of graphics in the drawable part of the drawing surface.
Arguments should be expressed relatively to the result of a previous printable_rect() call. That is, `printable_rect(&w, &h); origin(w/2, 0);` sets the graphics origin at the top center of the drawable area. Successive origin() calls don't combine their effects. Origin() calls are not affected by rotate() calls (for classes derived from Fl_Paged_Device).

Parameters

in	*x,y*	Horizontal and vertical positions in the drawing surface of the desired origin of graphics.

Reimplemented in Fl_Copy_Surface, Fl_Image_Surface, Fl_PDF_File_Surface, Fl_PostScript_File_Device, Fl_EPS_File_Surface, Fl_Printer, and Fl_SVG_File_Surface.

13.154.3.5 print_window_part()

```
void Fl_Widget_Surface::print_window_part (
            Fl_Window * win,
            int x,
            int y,
            int w,
            int h,
            int delta_x = 0,
            int delta_y = 0 )
```

Draws a rectangular part of an on-screen window.

Parameters

win	The window from where to capture. Can be an Fl_Gl_Window. Sub-windows that intersect the rectangle are also captured.
x	The rectangle left
y	The rectangle top
w	The rectangle width
h	The rectangle height
delta_x,delta_y	Optional horizontal and vertical offsets from current graphics origin where to draw the top left of the captured rectangle.

13.154.3.6 printable_rect()

```
int Fl_Widget_Surface::printable_rect (
            int * w,
            int * h )  [virtual]
```

Computes the width and height of the drawable area of the drawing surface.
Values are in the same unit as that used by FLTK drawing functions and are unchanged by calls to origin(). If the object is derived from class Fl_Paged_Device, values account for the user-selected paper type and print orientation and are changed by scale() calls.

Returns

0 if OK, non-zero if any error

Reimplemented in Fl_Copy_Surface, Fl_Image_Surface, Fl_PDF_File_Surface, Fl_PostScript_File_Device, Fl_EPS_File_Surface, Fl_Printer, and Fl_SVG_File_Surface.

13.154.3.7 translate()

```
void Fl_Widget_Surface::translate (
            int x,
            int y )  [virtual]
```

Translates the current graphics origin accounting for the current rotation.
Each translate() call must be matched by an untranslate() call. Successive translate() calls add up their effects.
Reimplemented in Fl_Copy_Surface, Fl_Image_Surface, Fl_PDF_File_Surface, Fl_PostScript_File_Device, Fl_EPS_File_Surface, Fl_Printer, and Fl_SVG_File_Surface.

13.154.3.8 untranslate()

```
void Fl_Widget_Surface::untranslate (
            void  )  [virtual]
```

Undoes the effect of a previous translate() call.
Reimplemented in Fl_Copy_Surface, Fl_Image_Surface, Fl_PDF_File_Surface, Fl_EPS_File_Surface, Fl_SVG_File_Surface, Fl_PostScript_File_Device, and Fl_Printer.
The documentation for this class was generated from the following files:

- Fl_Widget_Surface.H
- Fl_Widget_Surface.cxx

13.155 Fl_Widget_Tracker Class Reference

This class should be used to control safe widget deletion.
`#include <Fl_Widget_Tracker.H>`

Public Member Functions

- void **clear** ()

  Clear the widget pointer.
- int deleted ()

  Check if the widget was deleted since the tracker was created.
- int exists ()

  Check if the widget exists and was not deleted since the tracker was created.
- **Fl_Widget_Tracker** (const Fl_Widget_Tracker &)=delete
- **Fl_Widget_Tracker** (Fl_Widget *wi)

  The constructor adds a widget to the watch list.
- **Fl_Widget_Tracker** (Fl_Widget_Tracker &&)=delete
- Fl_Widget_Tracker & **operator=** (const Fl_Widget_Tracker &)=delete
- Fl_Widget_Tracker & **operator=** (Fl_Widget_Tracker &&)=delete
- Fl_Widget * widget ()

  Returns a pointer to the watched widget.
- ~**Fl_Widget_Tracker** ()

  The destructor removes a widget from the watch list.

13.155.1 Detailed Description

This class should be used to control safe widget deletion.
You can use an Fl_Widget_Tracker object to watch another widget, if you need to know whether this widget has been deleted during a callback.
This simplifies the use of the "safe widget deletion" methods Fl::watch_widget_pointer() and Fl::release_widget_pointer() and makes their use more reliable, because the destructor automatically releases the widget pointer from the widget watch list.
Fl_Widget_Tracker is intended to be used as an automatic (local/stack) variable, such that its destructor is called when the object's scope is left. This ensures that no stale widget pointers are left in the widget watch list (see example below).
You can also create Fl_Widget_Tracker objects with `new`, but then it is your responsibility to delete the object (and thus remove the widget pointer from the watch list) when it is no longer needed.
Example:

```
int MyClass::handle (int event) {

  if (...) {
    Fl_Widget_Tracker wp(this);           // watch myself
    do_callback();                        // call the callback

    if (wp.deleted()) return 1;           // exit, if deleted

    // Now we are sure that the widget has not been deleted,
    // and it is safe to access the widget:

    box(FL_FLAT_BOX);
    color(FL_WHITE);
    redraw();
  }
}
```

13.155.2 Member Function Documentation

13.155.2.1 deleted()

`int Fl_Widget_Tracker::deleted ( ) [inline]`
Check if the widget was deleted since the tracker was created.

Returns

1 if the watched widget has been deleted, otherwise 0

13.155.2.2 exists()

`int Fl_Widget_Tracker::exists ( ) [inline]`
Check if the widget exists and was not deleted since the tracker was created.

Returns

1 if the watched widget exists, otherwise 0

13.155.2.3 widget()

`Fl_Widget * Fl_Widget_Tracker::widget ( ) [inline]`
Returns a pointer to the watched widget.

Returns

nullptr if the widget was deleted.

The documentation for this class was generated from the following files:

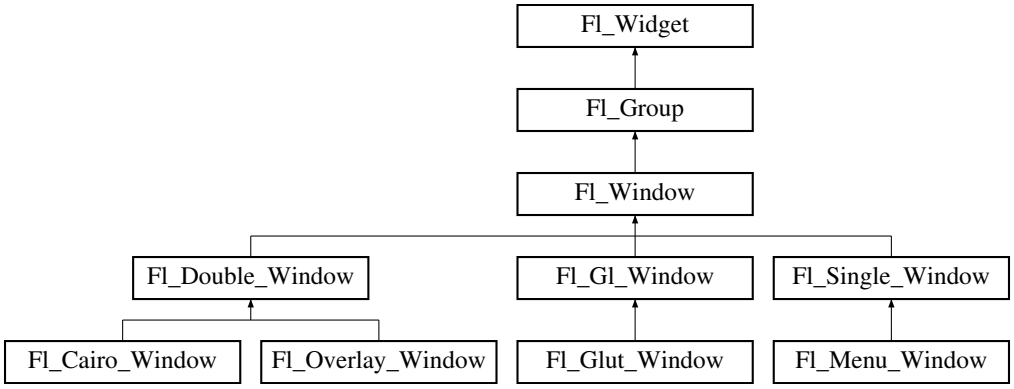
- Fl_Widget_Tracker.H
- Fl.cxx

13.156 Fl_Window Class Reference

This widget produces an actual window.

`#include <Fl_Window.H>`

Inheritance diagram for Fl_Window:



Public Types

- typedef struct HICON__ ∗ **HICON**

Public Member Functions

- void allow_expand_outside_parent ()

  Allow this subwindow to expand outside the area of its parent window.
- virtual class Fl_Double_Window ∗ as_double_window ()

  Return non-null if this is an Fl_Double_Window object.
- virtual class Fl_Overlay_Window ∗ as_overlay_window ()

  Return non-null if this is an Fl_Overlay_Window object.
- Fl_Window const ∗ as_window () const override
- Fl_Window ∗ as_window () override

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- unsigned int **border** () const

  Returns whether the window possesses a border.
- void border (int b)

  Sets whether or not the window manager border is around the window.
- void clear_border ()

  Fast inline function to turn the window manager border off.
- void clear_modal_states ()

  Clears the "modal" flags and converts a "modal" or "non-modal" window back into a "normal" window.
- void **copy_label** (const char ∗a)

  Sets the window titlebar label to a copy of a character string.
- void cursor (const Fl_RGB_Image ∗, int, int)

  Changes the cursor for this window using the provided image as cursor's shape.
- void cursor (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)

  For back compatibility only.
- void cursor (Fl_Cursor)

  Changes the cursor for this window.
- int decorated_h () const

  Returns the window height including any window title bar and any frame added by the window manager.
- int decorated_w () const

Returns the window width including any frame added by the window manager.

- void default_cursor (Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE)

  For back compatibility only.
- void default_cursor (Fl_Cursor)

  Sets the default window cursor.
- void **draw_backdrop** ()

  Draw the background image if one is set and is aligned inside.
- Fl_Window (int w, int h, const char *title=0)

  Creates a window from the given width `w`, height `h`, and `title`.
- Fl_Window (int x, int y, int w, int h, const char *title=0)

  Creates a window from the given position (x, y), size (w, h) and title.
- virtual void flush ()

  Forces the window to be drawn, this window is also made current and calls draw().
- void free_position ()

  Undoes the effect of a previous resize() or show() so that the next time show() is called the window manager is free to position the window.
- void fullscreen ()

  Makes the window completely fill one or more screens, without any window manager border visible.
- unsigned int **fullscreen_active** () const

  Returns non zero if FULLSCREEN flag is set, 0 otherwise.
- void **fullscreen_off** ()

  Turns off any side effects of fullscreen()
- void **fullscreen_off** (int X, int Y, int W, int H)

  Turns off any side effects of fullscreen() and does resize(X,Y,W,H).
- void fullscreen_screens (int top, int bottom, int left, int right)

  Sets which screens should be used when this window is in fullscreen mode.
- uchar get_size_range (int *minw, int *minh, int *maxw=NULL, int *maxh=NULL, int *dw=NULL, int *dh=NULL, int *aspect=NULL) const

  Gets the allowable range to which the user can resize this window.
- int handle (int) override

  Handles the specified event.
- void hide () override

  Removes the window from the screen.
- void **hotspot** (const Fl_Widget &p, int offscreen=0)

  See void Fl_Window::hotspot(int x, int y, int offscreen = 0)
- void **hotspot** (const Fl_Widget *, int offscreen=0)

  See void Fl_Window::hotspot(int x, int y, int offscreen = 0)
- void hotspot (int x, int y, int offscreen=0)

  Positions the window so that the mouse is pointing at the given position, or at the center of the given widget, which may be the window itself.
- const void * icon () const

  Gets the current icon window target dependent data.
- void icon (const Fl_RGB_Image *)

  Sets or resets a single window icon.
- void icon (const void *ic)

  Platform-specific method to set the window icon usable on Windows and X11 only.
- void iconize ()

  Iconifies the window.
- const char * **iconlabel** () const

  See void Fl_Window::iconlabel(const char)*

- void **iconlabel** (const char ∗)

  Sets the icon label.
- void icons (const Fl_RGB_Image ∗[ ], int)

  Sets the window icons.
- void icons (HICON big_icon, HICON small_icon)

  Sets the window icons using HICON handles (Windows platform only).
- const char ∗ **label** () const

  See void Fl_Window::label(const char∗)

- void **label** (const char ∗)

  Sets the window title bar label.
- void **label** (const char ∗label, const char ∗iconlabel)

  Sets the icon label.
- void make_current ()

  Sets things up so that the drawing functions in <FL/fl_draw.H> will go into this window.
- void maximize ()

  Maximizes a top-level window to its current screen.
- unsigned int **maximize_active** () const

  Returns whether the window is currently maximized.
- unsigned int **menu_window** () const

  Returns true if this window is a menu window.
- unsigned int modal () const

  Returns true if this window is modal.
- unsigned int **non_modal** () const

  Returns true if this window is modal or non-modal.
- fl_uintptr_t os_id () const

  Returns a platform-specific identification of a shown window, or 0 if not shown.
- unsigned int **override** () const

  Returns non zero if OVERRIDE flag is set, 0 otherwise.
- void resize (int X, int Y, int W, int H) override

  Changes the size and position of the window.
- int screen_num () const

  The number of the screen containing the mapped window.
- void screen_num (int screen_num)

  Set the number of the screen where to map the window.
- void set_menu_window ()

  Marks the window as a menu window.
- void set_modal ()

  A "modal" window, when shown(), will prevent any events from being delivered to other windows in the same program, and will also remain on top of the other windows (if the X window manager supports the "transient for" property).
- void set_non_modal ()

  A "non-modal" window (terminology borrowed from Microsoft Windows) acts like a modal() one in that it remains on top, but it has no effect on event delivery.
- void **set_override** ()

  Activates the flags NOBORDER|OVERRIDE.
- void set_tooltip_window ()

  Marks the window as a tooltip window.
- const Fl_Image ∗ shape () const

  Returns the image controlling the window shape or NULL.
- void shape (const Fl_Image &b)

  Set the window's shape with an Fl_Image.

- void shape (const Fl_Image *img)

  Assigns a non-rectangular shape to the window.
- void show () override

  Puts the window on the screen.
- void show (int argc, char **argv)

  Puts the window on the screen with show() and parses command-line arguments.
- int shown () const

  Returns non-zero if show() has been called (but not hide()).
- void size_range (int minw, int minh, int maxw=0, int maxh=0, int dw=0, int dh=0, int aspect=0)

  Sets the allowable range to which the user can resize this window.
- unsigned int **tooltip_window** () const

  Returns true if this window is a tooltip window.
- void un_maximize ()

  Returns a previously maximized top-level window to its previous size.
- void wait_for_expose ()

  Waits for the window to be displayed after calling show().
- int **x_root** () const

  Gets the x position of the window on the screen.
- const char * xclass () const

  Returns the xclass for this window, or a default.
- void xclass (const char *c)

  Sets the xclass for this window.
- int **y_root** () const

  Gets the y position of the window on the screen.
- virtual ~Fl_Window ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Group

- Fl_Widget *& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget *o)

  See void Fl_Group::add(Fl_Widget &w)
- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget *const * array () const

  Returns a pointer to the internal array of children.
- Fl_Group const * as_group () const override
- Fl_Group * as_group () override

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const

  Returns the n'th child.
- int **children** () const

  Returns how many child widgets the group has.
- void clear ()

  Deletes all child widgets from memory recursively.
- unsigned int clip_children () const

Returns the current clipping mode.

- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)

  Removes the widget at `index` from the group and deletes it.
- void end ()

  Exactly the same as current(this->parent()).
- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*
- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.
- void **forms_end** ()

  This is for forms compatibility only.
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

  This does insert(w, find(before)).
- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.
- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.
- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

Gets the current user data (long) argument that is passed to the callback function.

- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

Checks if w is a child of this widget.

- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.

- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.
- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

Sends the specified event to the widget.

- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char ∗ tooltip () const

  Gets the current tooltip text.
- void tooltip (const char ∗text)

  Sets the Tooltip Text for a Widget.
- Fl_Window ∗ top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window ∗ top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void ∗ user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data ∗v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void ∗v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.

- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window ∗ window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ∼Fl_Widget ()

  Destroys the widget.

Static Public Member Functions

- static Fl_Window ∗ current ()

  Returns the last window that was made current.
- static void **default_callback** (Fl_Window ∗, void ∗v)

  Back compatibility: Sets the default callback v for win to call on close event.
- static void default_icon (const Fl_RGB_Image ∗)

  Sets a single default window icon.
- static void default_icons (const Fl_RGB_Image ∗[ ], int)

  Sets the default window icons.
- static void default_icons (HICON big_icon, HICON small_icon)

  Sets the default window icons (Windows platform only).
- static const char ∗ default_xclass ()

  Returns the default xclass.
- static void default_xclass (const char ∗)

  Sets the default window xclass.
- static bool is_a_rescale ()

  Returns true when a window is being rescaled.
- static char show_next_window_iconic ()

  Returns the static flag whether the next window should be opened iconified.
- static void show_next_window_iconic (char stat)

  Sets a static flag whether the next window should be opened iconified.

Static Public Member Functions inherited from Fl_Group

- static Fl_Group ∗ current ()

  Returns the currently active group.
- static void current (Fl_Group ∗g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.
- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.
- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Member Functions

- void default_size_range ()

  Protected method to calculate the default size range of a window.
- void draw () override

  Draws the widget.
- int force_position () const

  Returns the internal state of the window's `FORCE_POSITION` flag.
- void force_position (int force)

  Sets an internal flag that tells FLTK and the window manager to honor position requests.
- void free_icons ()

  Deletes all icons previously attached to the window.
- int is_resizable ()

  Protected method to determine whether a window is resizable.

Protected Member Functions inherited from Fl_Group

- Fl_Rect * bounds ()

  Returns the internal array of widget sizes and positions.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget *, int)

  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)

  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)

  Allow derived groups to act when a child widget is removed from the group.
- int * sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.

- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char *label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

  Internal use only.
- void x (int v)

  Internal use only.
- void y (int v)

  Internal use only.

Static Protected Attributes

- static Fl_Window * current_

  Stores the last window that was made current.

Friends

- class **Fl_Window_Driver**
- class **Fl_X**

Additional Inherited Members

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,

AUTO_DELETE_USER_DATA = 1<<23 ,
MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
USERFLAG1 = 1<<31 }

flags possible values enumeration.

13.156.1 Detailed Description

This widget produces an actual window.
This can either be a main window, with a border and title and all the window management controls, or a "subwindow" inside a window. This is controlled by whether or not the window has a parent().
Once you create a window, you usually add children Fl_Widget's to it by using window->add(child) for each new widget. See Fl_Group for more information on how to add and remove children.
There are several subclasses of Fl_Window that provide double-buffering, overlay, menu, and OpenGL support.
The window's callback is done if the user tries to close a window using the window manager and Fl::modal() is zero or equal to the window. Fl_Window has a default callback that calls Fl_Window::hide(). Callback reasons can be `FL_REASON_CANCELLED` if the Escape key was pressed, or `FL_REASON_CLOSED` when the close button is clicked. `FL_WHEN_...` flags are ignored.

13.156.2 Constructor & Destructor Documentation

13.156.2.1 Fl_Window() [1/2]

```
Fl_Window::Fl_Window (
            int w,
            int h,
            const char * title = 0 )
```

Creates a window from the given width `w`, height `h`, and `title`.
If Fl_Group::current() is not NULL, the window is created as a subwindow of the parent window.
The (w, h) form of the constructor creates a top-level window and asks the window manager to position the window.
The (x, y, w, h) form of the constructor either creates a subwindow or a top-level window at the specified location (x, y), subject to window manager configuration. If you do not specify the position of the window, the window manager will pick a place to show the window or allow the user to pick a location. Use position(x, y) or hotspot() before calling show() to request a position on the screen. See Fl_Window::resize() for some more details on positioning windows.
Top-level windows initially have visible() set to 0 and parent() set to NULL. Subwindows initially have visible() set to 1 and parent() set to the parent window pointer.
Fl_Widget::box() defaults to FL_FLAT_BOX. If you plan to completely fill the window with children widgets you should change this to FL_NO_BOX. If you turn the window border off you may want to change this to FL_UP_BOX.

See also

Fl_Window(int x, int y, int w, int h, const char *title)

13.156.2.2 Fl_Window() [2/2]

```
Fl_Window::Fl_Window (
            int x,
            int y,
            int w,
            int h,
            const char * title = 0 )
```

Creates a window from the given position (x, y), size (w, h) and title.
On a multi-screen system, the values computed by Fl::screen_xywh(int &X, int &Y, int &W, int &H, int n) can be used to discover the coordinates of the area of screen #n. When these screens have various scale factor values, an (x, y) pair may not be enough to specify the targeted screen for the window, because the same (x,y) pair can belong to several screens. In that situation, a call to Fl_Window::screen_num(int) is to be used to identify unambiguously the targeted screen.

See also

Fl_Window(int w, int h, const char *title)

Fl::screen_xywh(int &X, int &Y, int &W, int &H, int n)

Note

Under Wayland, it's generally not possible for the client app to control the position of a window in the system. It's possible to specify on what screen should the compositor place a fullscreen window. It's also possible to make an Fl_Window the child of another window or group and control with `x` and `y` its screen position relatively to the enclosing window. Apply member function Fl_Window::allow_expand_outside_parent() to the child window to allow it to expand partially or totally outside its parent.

13.156.2.3 ~Fl_Window()

`Fl_Window::~Fl_Window ( ) [virtual]`

The destructor *also deletes all the children*.

This allows a whole tree to be deleted at once, without having to keep a pointer to all the children in the user code. A kludge has been done so the Fl_Window and all of its children can be automatic (local) variables, but you must declare the Fl_Window *first* so that it is destroyed last.

13.156.3 Member Function Documentation

13.156.3.1 allow_expand_outside_parent()

`void Fl_Window::allow_expand_outside_parent ( )`

Allow this subwindow to expand outside the area of its parent window.

This is presently implemented only for the Wayland platform to help support window docking.

Since

1.4.0

13.156.3.2 as_double_window()

`virtual class Fl_Double_Window * Fl_Window::as_double_window ( ) [inline], [virtual]`

Return non-null if this is an Fl_Double_Window object.

Reimplemented in Fl_Double_Window.

13.156.3.3 as_overlay_window()

`virtual class Fl_Overlay_Window * Fl_Window::as_overlay_window ( ) [inline], [virtual]`

Return non-null if this is an Fl_Overlay_Window object.

Reimplemented in Fl_Overlay_Window.

13.156.3.4 as_window() [1/2]

`Fl_Window const * Fl_Window::as_window ( ) const [inline], [override], [virtual]`

Reimplemented from Fl_Widget.

13.156.3.5 as_window() [2/2]

`Fl_Window * Fl_Window::as_window ( ) [inline], [override], [virtual]`

Returns an Fl_Window pointer if this widget is an Fl_Window.

Use this method if you have a widget (pointer) and need to know whether this widget is derived from Fl_Window. If it returns non-NULL, then the widget in question is derived from Fl_Window, and you can use the returned pointer to access its children or other Fl_Window-specific methods.

Return values

NULL	if this widget is not derived from Fl_Window.

Note

This method is provided to avoid dynamic_cast.

See also

Fl_Widget::as_group(), Fl_Widget::as_gl_window()

Reimplemented from Fl_Widget.

13.156.3.6 border()

```
void Fl_Window::border (
            int b )
```

Sets whether or not the window manager border is around the window.
The default value is true. The macOS platform ignores requests to change the border state of a fullscreen or maximized window. *With some X window managers, this does not work after show() has been called.*

13.156.3.7 clear_border()

```
void Fl_Window::clear_border ( ) [inline]
```

Fast inline function to turn the window manager border off.
It only works before show() is called.

13.156.3.8 clear_modal_states()

```
void Fl_Window::clear_modal_states ( ) [inline]
```

Clears the "modal" flags and converts a "modal" or "non-modal" window back into a "normal" window.
Note that there are *three* states for a window: modal, non-modal, and normal.
You can not change the "modality" of a window whilst it is shown, so it is necessary to first hide() the window, change its "modality" as required, then re-show the window for the new state to take effect.
This method can also be used to change a "modal" window into a "non-modal" one. On several supported platforms, the "modal" state over-rides the "non-modal" state, so the "modal" state must be cleared before the window can be set into the "non-modal" state. In general, the following sequence should work:

```
win->hide();
win->clear_modal_states();
// Set win to new state as desired, or leave "normal", e.g...
win->set_non_modal();
win->show();
```

Note

Under some window managers, the sequence of hiding the window and changing its modality will often cause it to be re-displayed at a different position when it is subsequently shown. This is an irritating feature but appears to be unavoidable at present. As a result we would advise to use this method only when absolutely necessary.

See also

void set_modal(), void set_non_modal()

13.156.3.9 current()

```
Fl_Window * Fl_Window::current ( ) [static]
```

Returns the last window that was made current.

See also

Fl_Window::make_current()

13.156.3.10 cursor() [1/3]

```
void Fl_Window::cursor (
            const Fl_RGB_Image * image,
            int hotx,
            int hoty )
```

Changes the cursor for this window using the provided image as cursor's shape.
The window must be show()'n for this function to have any effect. This always calls the system. If you are changing the cursor a lot you may want to keep track of how you set it in a static variable and call this only if the new cursor is different.
The default cursor will be used if the provided image cannot be used as a cursor.

Parameters

image	Sets the cursor size and shape
hotx,hoty	Sets the cursor's active location relatively to top-left of `image` when clicking

See also

cursor(Fl_Cursor), default_cursor()

13.156.3.11 cursor() [2/3]

```
void Fl_Window::cursor (
            Fl_Cursor c,
            Fl_Color ,
            Fl_Color  = FL_WHITE )
```

For back compatibility only.
Same as Fl_Window::cursor(Fl_Cursor)

13.156.3.12 cursor() [3/3]

```
void Fl_Window::cursor (
            Fl_Cursor c )
```

Changes the cursor for this window.
The window must be show()'n for this function to have any effect. This always calls the system. If you are changing the cursor a lot you may want to keep track of how you set it in a static variable and call this only if the new cursor is different.
The type Fl_Cursor is an enumeration defined in <FL/Enumerations.H>.

See also

cursor(const Fl_RGB_Image*, int, int), default_cursor()

13.156.3.13 decorated_h()

```
int Fl_Window::decorated_h ( ) const
```

Returns the window height including any window title bar and any frame added by the window manager.
Same as h() if applied to a subwindow, or if window is not yet mapped.

Note

Under X11, FLTK is able to compute the size of window titlebars and borders only if these decoration elements are strictly X11-based. When that's not the case, decorated_h() returns the same value as h() and decorated_w() as w(), and FLTK cannot access window decorations.

Under X11 again, the values returned by decorated_h() and decorated_w() may not be reliable **during a resize operation**. The size of decoration elements of a window is best computed when the window is first mapped.

13.156.3.14 decorated_w()

```
int Fl_Window::decorated_w ( ) const
```

Returns the window width including any frame added by the window manager.
Same as w() if applied to a subwindow, or if window is not yet mapped.

See also

decorated_h().

13.156.3.15 default_cursor() [1/2]

```
void Fl_Window::default_cursor (
            Fl_Cursor c,
            Fl_Color ,
            Fl_Color  = FL_WHITE )
```

For back compatibility only.
same as Fl_Window::default_cursor(Fl_Cursor)

13.156.3.16 default_cursor() [2/2]

```
void Fl_Window::default_cursor (
            Fl_Cursor c )
```

Sets the default window cursor.
This is the cursor that will be used after the mouse pointer leaves a widget with a custom cursor set.

See also

cursor(const Fl_RGB_Image∗, int, int), default_cursor()

13.156.3.17 default_icon()

```
void Fl_Window::default_icon (
            const Fl_RGB_Image * icon )  [static]
```

Sets a single default window icon.
If `icon` is NULL the current default icons are removed.

Parameters

in	*icon*	default icon for all windows subsequently created or NULL

See also

Fl_Window::default_icons(const Fl_RGB_Image ∗[], int)

Fl_Window::icon(const Fl_RGB_Image ∗)

Fl_Window::icons(const Fl_RGB_Image ∗[], int)

Note

See Window icons for the Wayland platform.

13.156.3.18 default_icons() [1/2]

```
void Fl_Window::default_icons (
            const Fl_RGB_Image * icons[],
            int count )  [static]
```

Sets the default window icons.

The default icons are used for all windows that don't have their own icons set before show() is called. You can change the default icons whenever you want, but this only affects windows that are created (and shown) after this call.
The given images in `icons` are copied. You can use a local variable or free the images immediately after this call.

Parameters

in	*icons*	default icons for all windows subsequently created
in	*count*	number of images in `icons`. Set to 0 to remove the current default icons

See also

Fl_Window::default_icon(const Fl_RGB_Image *)

Fl_Window::icon(const Fl_RGB_Image *)

Fl_Window::icons(const Fl_RGB_Image *[], int)

Note

See Window icons for the Wayland platform.

13.156.3.19 default_icons() [2/2]

```
static void Fl_Window::default_icons (
            HICON big_icon,
            HICON small_icon )  [static]
```

Sets the default window icons (Windows platform only).
Convenience function to set the default icons using Windows' native HICON icon handles.
The given icons are copied. You can free the icons immediately after this call.

Parameters

in	*big_icon*	default large icon for all windows subsequently created
in	*small_icon*	default small icon for all windows subsequently created

See also

Fl_Window::default_icon(const Fl_RGB_Image *)

Fl_Window::default_icons(const Fl_RGB_Image *[], int)

Fl_Window::icon(const Fl_RGB_Image *)

Fl_Window::icons(const Fl_RGB_Image *[], int)

Fl_Window::icons(HICON, HICON)

13.156.3.20 default_size_range()

```
void Fl_Window::default_size_range ( )  [protected]
```

Protected method to calculate the default size range of a window.
This method is called internally prior to showing a window to ensure that the window's size range values are calculated if a resizable() widget has been set but size_range() has not been called explicitly.
This method does nothing if size_range() has been called before.
Otherwise FLTK tries to figure out the window's size range from the setting of the window's resizable() widget as follows and roughly in the given order.

1. If resizable() is NULL (this is the default) then the window cannot be resized and the resize border and max-size control will not be displayed for the window.

2. If either dimension of resizable() is zero, then the window cannot resize in that direction.
3. The resizable() widget is clipped to the window area.
4. The non-resizable portion of the window is calculated as the difference of the window's size and the clipped resizable() widget's size.
5. If either dimension of the clipped resizable() widget is greater than 100, then 100 is considered its minimum width/height. This allows the resizable widget to shrink below its original size.
6. Finally the minimum width/height of the window is set to the non-resizable portion plus the width/height of the resizable() widget as calculated above.

In simple words:

- It is assumed that the resizable() widget can be indefinitely enlarged and/or shrunk to a minimum width/height of 100 unless it is smaller than that, which is then considered the minimum.
- The window's size_range() minimum values are set to the sum of the non-resizable portion of the window and the previously calculated minimum size of the resizable() widget.

Examples:

```
Fl_Window win(400, 400);
win.resizable(win);
// win.size_range(100, 100, 0, 0);
```

The minimum size of the resizable is 100, hence the minimum size of the total window is also 100 in both directions.

```
Fl_Window win(400, 400);
Fl_Box box(20, 20, 360, 360);
win.resizable(box);
// win.size_range(140, 140, 0, 0);
```

The calculated minimum width and height would be 20 + 100 + 20 in both dimensions.

```
Fl_Window win(400, 400);
Fl_Box box(200, 0, 500, 300); // note: width 500 too large: clipped
win.resizable(box);
// win.size_range(300, 200, 0, 0);
```

The width of the resizable is clipped to 200, hence the minimum size of the total window is also 200 (fix) + 100 (min. resizable) in x direction. The minimum value in y direction is 100 (resizable) + 100 (fixed part).
The calculation is based on clipping the resizable widget to the window area to prevent programming errors and the assumption that the resizable widget can be shrunk to 100x100 or its original size, whichever is smaller.
If this is not what you want, please use Fl_Window::size_range() explicitly so you can set any appropriate range.

Since

1.4.0

13.156.3.21 default_xclass() [1/2]

```
const char * Fl_Window::default_xclass ( ) [static]
```

Returns the default xclass.

See also

Fl_Window::default_xclass(const char *)

13.156.3.22 default_xclass() [2/2]

```
void Fl_Window::default_xclass (
            const char * xc ) [static]
```

Sets the default window xclass.
The default xclass is used for all windows that don't have their own xclass set before show() is called. You can change the default xclass whenever you want, but this only affects windows that are created (and shown) after this call.
The given string `xc` is copied. You can use a local variable or free the string immediately after this call.
If you don't call this, the default xclass for all windows will be "FLTK". You can reset the default xclass by specifying NULL for `xc`.
If you call Fl_Window::xclass(const char *) for any window, then this also sets the default xclass, unless it has been set before.

Parameters

in	*xc*	default xclass for all windows subsequently created

See also

Fl_Window::xclass(const char *)

13.156.3.23 draw()

`void Fl_Window::draw ( ) [override], [protected], [virtual]`

Draws the widget.

Never call this function directly. FLTK will schedule redrawing whenever needed. If your widget must be redrawn as soon as possible, call redraw() instead.

Override this function to draw your own widgets.

If you ever need to call another widget's draw method *from within your own draw() method*, e.g. for an embedded scrollbar, you can do it (because draw() is virtual) like this:

```
Fl_Widget *s = &scrollbar; // scrollbar is an embedded Fl_Scrollbar
s->draw();                 // calls Fl_Scrollbar::draw()
```

Reimplemented from Fl_Group.

13.156.3.24 flush()

`void Fl_Window::flush ( ) [virtual]`

Forces the window to be drawn, this window is also made current and calls draw().

Reimplemented in Fl_Double_Window, Fl_Gl_Window, Fl_Overlay_Window, and Fl_Single_Window.

13.156.3.25 force_position() [1/2]

`int Fl_Window::force_position ( ) const [inline], [protected]`

Returns the internal state of the window's `FORCE_POSITION` flag.

Return values

1	if flag is set
0	otherwise

See also

force_position(int)

13.156.3.26 force_position() [2/2]

```
void Fl_Window::force_position (
            int force ) [inline], [protected]
```

Sets an internal flag that tells FLTK and the window manager to honor position requests.

This is used internally and should not be needed by user code.

Parameters

in	*force*	1 to set the FORCE_POSITION flag, 0 to clear it

13.156.3.27 free_icons()

`void Fl_Window::free_icons ( ) [protected]`

Deletes all icons previously attached to the window.

See also

Fl_Window::icons(const Fl_RGB_Image *icons[], int count)

13.156.3.28 free_position()

```
void Fl_Window::free_position ( ) [inline]
```

Undoes the effect of a previous resize() or show() so that the next time show() is called the window manager is free to position the window.
This is the same as the *protected* method Fl_Window::force_position(0).

13.156.3.29 fullscreen()

```
void Fl_Window::fullscreen ( )
```

Makes the window completely fill one or more screens, without any window manager border visible.
You must use fullscreen_off() to undo this.

Note

On some platforms, this can result in the keyboard being grabbed. The window may also be recreated, meaning hide() and show() will be called.

See also

void Fl_Window::fullscreen_screens()

13.156.3.30 fullscreen_screens()

```
void Fl_Window::fullscreen_screens (
            int top,
            int bottom,
            int left,
            int right )
```

Sets which screens should be used when this window is in fullscreen mode.
The window will be resized to the top of the screen with index `top`, the bottom of the screen with index `bottom`, etc.
If this method is never called, or if any argument is < 0, then the window will be resized to fill the screen it is currently on.
Wayland compositors may support only partially or not at all multi-screen fullscreen windows.

See also

void Fl_Window::fullscreen()

13.156.3.31 get_size_range()

```
uchar Fl_Window::get_size_range (
            int * minWidth,
            int * minHeight,
            int * maxWidth = NULL,
            int * maxHeight = NULL,
            int * deltaX = NULL,
            int * deltaY = NULL,
            int * aspectRatio = NULL ) const
```

Gets the allowable range to which the user can resize this window.

Parameters

out	*minWidth,minHeight,maxWidth,max↩ Height,deltaX,deltaY,aspectRatio*	are all pointers to integers that will receive the current respective value during the call. Every pointer can be NULL if that value is not needed.

Return values

0	if size range not set
1	if the size range was explicitly set by a call to Fl_Window::size_range() or has been calculated

See also

Fl_Window::size_range(int minWidth, int minHeight, int maxWidth, int maxHeight, int deltaX, int deltaY, int aspectRatio)

Since

1.4.0

13.156.3.32 handle()

```
int Fl_Window::handle (
            int event )  [override], [virtual]
```

Handles the specified event.
You normally don't call this method directly, but instead let FLTK do it when the user interacts with the widget.
When implemented in a widget, this function must return 0 if the widget does not use the event or 1 otherwise.
Most of the time, you want to call the inherited handle() method in your overridden method so that you don't short-circuit events that you don't handle. In this last case you should return the callee retval.
One exception to the rule in the previous paragraph is if you really want to *override* the behavior of the base class. This requires knowledge of the details of the inherited class.
In rare cases you may want to return 1 from your handle() method although you don't really handle the event. The effect would be to *filter* event processing, for instance if you want to dismiss non-numeric characters (keypresses) in a numeric input widget. You may "ring the bell" or show another visual indication or drop the event silently. In such a case you must not call the handle() method of the base class and tell FLTK that you *consumed* the event by returning 1 even if you didn't *do* anything with it.

Parameters

in	*event*	the kind of event received

Return values

0	if the event was not used or understood
1	if the event was used and can be deleted

See also

Fl_Event

Reimplemented from Fl_Group.

13.156.3.33 hide()

```
void Fl_Window::hide ( )  [override], [virtual]
```

Removes the window from the screen.

If the window is already hidden or has not been shown then this does nothing and is harmless.
Reimplemented from Fl_Widget.

13.156.3.34 hotspot()

```
void Fl_Window::hotspot (
            int x,
            int y,
            int offscreen = 0 )
```

Positions the window so that the mouse is pointing at the given position, or at the center of the given widget, which may be the window itself.
If the optional offscreen parameter is non-zero, then the window is allowed to extend off the screen (this does not work with some X window managers).

See also

position()

13.156.3.35 icon() [1/3]

```
const void * Fl_Window::icon ( ) const
```

Gets the current icon window target dependent data.

Deprecated in 1.3.3

13.156.3.36 icon() [2/3]

```
void Fl_Window::icon (
            const Fl_RGB_Image * icon )
```

Sets or resets a single window icon.
A window icon *can* be changed while the window is shown, but this *may* be platform and/or window manager dependent. To be sure that the window displays the correct window icon you should always set the icon before the window is shown.
If a window icon has not been set for a particular window, then the default window icon (see links below) or the system default icon will be used.
This method makes an internal copy of the `icon` pixel buffer, so once set, the Fl_RGB_Image instance can be freed by the caller.

Parameters

in	*icon*	icon for this window, NULL to reset window icon.

See also

Fl_Window::default_icon(const Fl_RGB_Image *)

Fl_Window::default_icons(const Fl_RGB_Image *[], int)

Fl_Window::icons(const Fl_RGB_Image *[], int)

Note

See Window icons for the Wayland platform.

13.156.3.37 icon() [3/3]

```
void Fl_Window::icon (
            const void * ic )
```

Platform-specific method to set the window icon usable on Windows and X11 only.
See Setting the Icon of a Window for its use under X11, and Setting the Icon of a Window under Windows.

Deprecated in 1.3.3 in favor of platform-independent methods Fl_Window::icon(const Fl_RGB_Image *icon) and Fl_Window::icons(const Fl_RGB_Image *icons[], int count).

13.156.3.38 iconize()

```
void Fl_Window::iconize ( )
```

Iconifies the window.
If you call this when shown() is false it will show() it as an icon. If the window is already iconified this does nothing.
Call show() to restore the window.
When a window is iconified/restored (either by these calls or by the user) the handle() method is called with FL_HIDE and FL_SHOW events and visible() is turned on and off.
There is no way to control what is drawn in the icon except with the string passed to Fl_Window::xclass(). You should not rely on window managers displaying the icons.

13.156.3.39 icons() [1/2]

```
void Fl_Window::icons (
            const Fl_RGB_Image * icons[],
            int count )
```

Sets the window icons.
You may set multiple window icons with different sizes. Dependent on the platform and system settings the best (or the first) icon will be chosen.
The given images in `icons` are copied. You can use a local variable or free the images immediately after this call.
If `count` is zero, current icons are removed. If `count` is greater than zero (must not be negative), then `icons`[] must contain at least `count` valid image pointers (not NULL). Otherwise the behavior is undefined.

Parameters

in	*icons*	icons for this window
in	*count*	number of images in `icons`. Set to 0 to remove the current icons

See also

Fl_Window::default_icon(const Fl_RGB_Image *)

Fl_Window::default_icons(const Fl_RGB_Image *[], int)

Fl_Window::icon(const Fl_RGB_Image *)

Note

See Window icons for the Wayland platform.

13.156.3.40 icons() [2/2]

```
void Fl_Window::icons (
            HICON big_icon,
            HICON small_icon )
```

Sets the window icons using HICON handles (Windows platform only).
The given icons are copied. You can free the icons immediately after this call.

Parameters

in	*big_icon*	large window icon
in	*small_icon*	small window icon

13.156.3.41 is_a_rescale()

`bool Fl_Window::is_a_rescale ( ) [static]`

Returns true when a window is being rescaled.

Since

1.4.0

13.156.3.42 is_resizable()

`int Fl_Window::is_resizable ( ) [protected]`

Protected method to determine whether a window is resizable.

If size_range() has not yet been called this method calculates the default size range values by calling default_size_range().

This method is for internal use only. The returned value is a bit mask and non-zero if the window is resizable in at least one direction.

Returns

non-zero if the window is resizable

Return values

0	the window is not resizable
1	the window is resizable in horizontal direction (w)
2	the window is resizable in vertical direction (h)
3	the window is resizable in both directions (w and h)

See also

default_size_range()

Since

1.4.0

13.156.3.43 make_current()

`void Fl_Window::make_current ( )`

Sets things up so that the drawing functions in <FL/fl_draw.H> will go into this window.

This is useful for incremental update of windows, such as in an idle callback, which will make your program behave much better if it draws a slow graphic. **Danger: incremental update is very hard to debug and maintain!**

This method only works for the Fl_Window and Fl_Gl_Window derived classes.

13.156.3.44 maximize()

`void Fl_Window::maximize ( )`

Maximizes a top-level window to its current screen.

This function is effective only with a show()'n, resizable, top-level window. Bordered and borderless windows can be used. Fullscreen windows can't be used.

See also

Fl_Window::un_maximize(), Fl_Window::maximize_active()

Since

1.4.0

13.156.3.45 modal()

```
unsigned int Fl_Window::modal ( ) const  [inline]
```

Returns true if this window is modal.

13.156.3.46 os_id()

```
fl_uintptr_t Fl_Window::os_id ( ) const
```

Returns a platform-specific identification of a shown window, or 0 if not shown.

Note

This identification may differ from the platform-specific reference of an Fl_Window object used by functions fl_x11_xid(), fl_mac_xid(), fl_x11_find(), and fl_mac_find().

- X11 platform: the window's XID.
- macOS platform: The window number of the window's window device.
- other platforms: 0.

Since

1.4.0

13.156.3.47 resize()

```
void Fl_Window::resize (
            int X,
            int Y,
            int W,
            int H )  [override], [virtual]
```

Changes the size and position of the window.
If shown() is true, these changes are communicated to the window server (which may refuse that size and cause a further resize). If shown() is false, the size and position are used when show() is called. See Fl_Group for the effect of resizing on the child widgets.
You can also call the Fl_Widget methods position(x, y) and size(w, h) which are inline wrappers for this virtual function.
A top-level window can not force, but merely suggest a position and size to the operating system. The window manager may not be willing or able to display a window at the desired position or with the given dimensions. It is up to the application developer to verify window parameters after the resize request.
Reimplemented from Fl_Group.

13.156.3.48 screen_num() [1/2]

```
int Fl_Window::screen_num ( ) const
```

The number of the screen containing the mapped window.
This method returns the screen number (0 .. n) of the window if it is shown, otherwise the result value is undefined.

Note

The return value is undefined before the window is shown and if the window has been hidden after it was previously shown.

To position a window on a particular screen, use Fl_Window::screen_num(int) **before** you show() it.

Returns

screen number of the window if it is shown

See also

Fl_Window::screen_num(int)

Since

1.4.0

13.156.3.49 screen_num() [2/2]

```
void Fl_Window::screen_num (
            int screen_num )
```

Set the number of the screen where to map the window.
Call this and set also the window's desired x/y position before show()'ing the window. This can be necessary if a system has several screens with distinct scaling factors because the window's x() and y() may not suffice to uniquely identify one screen.
Consider a system with two screens where the left screen is A pixels wide and has a scaling factor of 1 whereas the right screen has a scaling factor of 2. For the sake of simplicity, consider only the X coordinate of the window. FLTK coordinates translate directly to pixel coordinates on the left screen, whereas FLTK coordinates multiplied by 2 correspond to pixel coordinates on the right screen. Consequently, FLTK coordinates between A/2 + 1 and A-1 can map to both screens. Therefore both the window coordinates and the screen number are necessary to uniquely identify where a window is to be mapped.
The valid range of `screen_num` is 0 .. Fl::screen_count() - 1.
This method does nothing if

- the window is already shown() or
- the given screen number is out of range.

Parameters

in	*screen_num*	screen number where the window is to be mapped

See also

Fl_Window::screen_num()

Fl::screen_count()

Since

1.4.0

13.156.3.50 set_menu_window()

```
void Fl_Window::set_menu_window ( ) [inline]
```

Marks the window as a menu window.
This is intended for internal use, but it can also be used if you write your own menu handling. However, this is not recommended.
This flag is used for correct "parenting" of windows in communication with the windowing system. Modern X window managers can use different flags to distinguish menu and tooltip windows from normal windows.
This must be called before the window is shown and cannot be changed later.

13.156.3.51 set_modal()

```
void Fl_Window::set_modal ( ) [inline]
```

A "modal" window, when shown(), will prevent any events from being delivered to other windows in the same program, and will also remain on top of the other windows (if the X window manager supports the "transient for" property).

Several modal windows may be shown at once, in which case only the last one shown gets events. You can see which window (if any) is modal by calling Fl::modal().

13.156.3.52 set_non_modal()

`void Fl_Window::set_non_modal ( ) [inline]`

A "non-modal" window (terminology borrowed from Microsoft Windows) acts like a modal() one in that it remains on top, but it has no effect on event delivery.
There are *three* states for a window: modal, non-modal, and normal.

13.156.3.53 set_tooltip_window()

`void Fl_Window::set_tooltip_window ( ) [inline]`

Marks the window as a tooltip window.
This is intended for internal use, but it can also be used if you write your own tooltip handling. However, this is not recommended.
This flag is used for correct "parenting" of windows in communication with the windowing system. Modern X window managers can use different flags to distinguish menu and tooltip windows from normal windows.
This must be called before the window is shown and cannot be changed later.

Note

Since Fl_Tooltip_Window is derived from Fl_Menu_Window, this also **clears** the menu_window() state.

13.156.3.54 shape() [1/3]

`const Fl_Image * Fl_Window::shape ( ) const`

Returns the image controlling the window shape or NULL.

Since

1.4.0

13.156.3.55 shape() [2/3]

```
void Fl_Window::shape (
            const Fl_Image & img )
```

Set the window's shape with an Fl_Image.

See also

void shape(const Fl_Image* img)

13.156.3.56 shape() [3/3]

```
void Fl_Window::shape (
            const Fl_Image * img )
```

Assigns a non-rectangular shape to the window.
This function gives an arbitrary shape (not just a rectangular region) to an Fl_Window. An Fl_Image of any dimension can be used as mask; it is rescaled to the window's dimension as needed.
The layout and widgets inside are unaware of the mask shape, and most will act as though the window's rectangular bounding box is available to them. It is up to you to make sure they adhere to the bounds of their masking shape.
The `img` argument can be an Fl_Bitmap, Fl_Pixmap, Fl_RGB_Image or Fl_Shared_Image:

- With Fl_Bitmap or Fl_Pixmap, the shaped window covers the image part where bitmap bits equal one, or where the pixmap is not fully transparent.
- With an Fl_RGB_Image with an alpha channel (depths 2 or 4), the shaped window covers the image part that is not fully transparent.

- With an Fl_RGB_Image of depth 1 (gray-scale) or 3 (RGB), the shaped window covers the non-black image part.
- With an Fl_Shared_Image, the shape is determined by rules above applied to the underlying image. The shared image should not have been scaled through Fl_Image::scale().

Platform details:

- On the unix/linux platform, the SHAPE extension of the X server is required. This function does control the shape of Fl_Gl_Window instances.
- On the Windows platform, this function does nothing with class Fl_Gl_Window.
- On the Mac platform, OS version 10.4 or above is required. An 8-bit shape-mask is used when `img` is an Fl_RGB_Image: with depths 2 or 4, the image alpha channel becomes the shape mask such that areas with alpha = 0 are out of the shaped window; with depths 1 or 3, white and black are in and out of the shaped window, respectively, and other colors give intermediate masking scores. This function does nothing with class Fl_Gl_Window.

The window borders and caption created by the window system are turned off by default. They can be re-enabled by calling Fl_Window::border(1).
A usage example is found at example/shapedwindow.cxx.

Version

1.3.3

13.156.3.57 show() [1/2]

```
void Fl_Window::show ( ) [override], [virtual]
```

Puts the window on the screen.
This has the side effect of opening the display, if not done before.
If the window is already shown then it is restored and raised to the top. This is really convenient because your program can call show() at any time, even if the window is already up. It also means that show() serves the purpose of raise() in other toolkits.
Fl_Window::show(int argc, char **argv) is used for top-level windows and allows standard arguments to be parsed from the command-line.

Note

For some obscure reasons Fl_Window::show() resets the current group by calling Fl_Group::current(0). The comments in the code say "get rid of very common user bug: forgot end()". Although this is true it may have unwanted side effects if you show() an unrelated window (maybe for an error message or warning) while building a window or any other group widget.

Todo Check if we can remove resetting the current group in a later FLTK version (after 1.3.x). This may break "already broken" programs though if they rely on this "feature".

See also

Fl_Window::show(int argc, char **argv)

Reimplemented from Fl_Widget.

13.156.3.58 show() [2/2]

```
void Fl_Window::show (
            int argc,
            char ** argv )
```

Puts the window on the screen with show() and parses command-line arguments.
This call should be used for top-level windows, at least for the first (main) window. It allows standard arguments to be parsed, as done by Fl::args(int, char **), from the command-line. You can use `argc` and `argv` from main(int argc, char **argv) for this call.

This call also sets up some system-specific internal variables, that is, it sets FL_SELECTION_COLOR and calls Fl::background(), Fl::background2(), Fl::foreground() with default or X resources-given values, and calls Fl::scheme(const char *) for the current scheme. On X11, it also calls Fl::dnd_text_ops(int), Fl_Tooltip::enable(int), Fl::visible_focus(int) with X resources-given values.

Parameters

argc	command-line argument count, usually from main()
argv	command-line argument vector, usually from main()

See also

virtual void Fl_Window::show()

Fl::args(int, char **)

13.156.3.59 show_next_window_iconic() [1/2]

```
static char Fl_Window::show_next_window_iconic ( ) [inline], [static]
```

Returns the static flag whether the next window should be opened iconified.

Note

This is an **internal function**, you should not use this in user code.

Please use Fl_Window::iconize() to iconify a window.

13.156.3.60 show_next_window_iconic() [2/2]

```
static void Fl_Window::show_next_window_iconic (
            char stat ) [inline], [static]
```

Sets a static flag whether the next window should be opened iconified.

Note

This is an **internal function**, you should not use this in user code.

Please use Fl_Window::iconize() instead.

13.156.3.61 shown()

```
int Fl_Window::shown ( ) const [inline]
```

Returns non-zero if show() has been called (but not hide()).
You can tell if a window is iconified with (w->shown() && !w->visible()).

13.156.3.62 size_range()

```
void Fl_Window::size_range (
            int minWidth,
            int minHeight,
            int maxWidth = 0,
            int maxHeight = 0,
            int deltaX = 0,
            int deltaY = 0,
            int aspectRatio = 0 )
```

Sets the allowable range to which the user can resize this window.
We recommend to call size_range() if you have a resizable() widget in a main window, and to call it after setting the resizable() and before show()'ing the window for best cross platform compatibility.
If this function is **not** called, FLTK tries to figure out the range when the window is shown. Please see the protected method default_size_range() for details.
It is undefined what happens if the current window size does not fit in the constraints passed to size_range().

Note

This only works for top-level windows and the exact behavior can be platform specific. To work correctly across all platforms size_range() must be called after setting the resizable() widget of the window and before the window is show()'n.

Calling size_range() after the window has been shown may work on some but not all platforms. If you need to change the size_range() after the window has been shown, then you should consider to hide() and show() the window again, i.e. call hide(), size_range(), and show() in this order.
Typical usage: call
`size_range(minWidth, minHeight);`
after setting the resizable widget and before calling show(). This ensures that the window cannot be resized smaller than the given values by user interaction.
`maxWidth` and `maxHeight` might be useful in some special cases but less frequently used.
The other optional parameters `deltaX`, `deltaY`, and `aspectRatio` are not recommended because they may not work on all platforms and may even under X11 not be supported by all Window Managers.

Parameters

in	*minWidth,minHeight*	The smallest the window can be. Either value must be greater than 0.
in	*maxWidth,maxHeight*	The largest the window can be. If either is equal to the minimum then you cannot resize in that direction. If either is zero then FLTK picks a maximum size in that direction such that the window will fill the screen.
in	*deltaX,deltaY*	These are size increments. The window will be constrained to widths of `minWidth + N * deltaX`, where N is any non-negative integer. If these are less or equal to 1 they are ignored (this is always ignored on Windows).
in	*aspectRatio*	A flag that indicates that the window should preserve its aspect ratio. This only works if both the maximum and minimum have the same aspect ratio (ignored on Windows and by many X window managers).

13.156.3.63 un_maximize()

`void Fl_Window::un_maximize ( )`
Returns a previously maximized top-level window to its previous size.

See also

Fl_Window::maximize()

Since

1.4.0

13.156.3.64 wait_for_expose()

`void Fl_Window::wait_for_expose ( )`
Waits for the window to be displayed after calling show().
Fl_Window::show() is not guaranteed to show and draw the window on all platforms immediately. Instead this is done in the background; particularly on X11 it will take a few messages (client server roundtrips) to display the window. Usually this small delay doesn't matter, but in some cases you may want to have the window instantiated and displayed synchronously.
Currently (as of FLTK 1.3.4) this method has an effect on X11 and Mac OS. On Windows, show() is always synchronous. The effect of show() varies with versions of Mac OS X: early versions have the window appear on the screen when show() returns, later versions don't. If you want to write portable code and need this synchronous show() feature, add win->wait_for_expose() on all platforms, and FLTK will just do the right thing.
This method can be used for displaying splash screens before calling Fl::run() or for having exact control over which window has the focus after calling show().
If the window is not shown(), this method does nothing.

Note

Depending on the platform and window manager wait_for_expose() may not guarantee that the window is fully drawn when it is called. Under X11 it may only make sure that the window is **mapped**, i.e. the internal (OS dependent) window object was created (and maybe shown on the desktop as an empty frame or something like that). You may need to call Fl::flush() after wait_for_expose() to make sure the window and all its widgets are drawn and thus visible.

FLTK does the best it can do to make sure that all widgets get drawn if you call wait_for_expose() and Fl::flush(). However, dependent on the window manager it can not be guaranteed that this does always happen synchronously. The only guaranteed behavior that all widgets are eventually drawn is if the FLTK event loop is run continuously, for instance with Fl::run().

See also

virtual void Fl_Window::show()

Example code for displaying a window before calling Fl::run()

```
Fl_Double_Window win = new Fl_Double_Window(...);

// do more window initialization here ...

win->show();                // show window
win->wait_for_expose();     // wait, until displayed
Fl::flush();                // make sure everything gets drawn

// do more initialization work that needs some time here ...

Fl::run();                  // start FLTK event loop
```

Note that the window will not be responsive until the event loop is started with Fl::run().

13.156.3.65 xclass() [1/2]

```
const char * Fl_Window::xclass ( ) const
```

Returns the xclass for this window, or a default.

See also

Fl_Window::default_xclass(const char *)

Fl_Window::xclass(const char *)

13.156.3.66 xclass() [2/2]

```
void Fl_Window::xclass (
            const char * xc )
```

Sets the xclass for this window.

A string used to tell the system what type of window this is. Mostly this identifies the picture to draw in the icon. This only works if called *before* calling show().

Under X, this is turned into a XA_WM_CLASS pair by truncating at the first non-alphanumeric character and capitalizing the first character, and the second one if the first is 'x'. Thus "foo" turns into "foo, Foo", and "xprog.1" turns into "xprog, XProg".

Under Wayland, this is used to set the "application identifier" of toplevel windows. If not set, the executable name is used.

Under Microsoft Windows, this string is used as the name of the WNDCLASS structure. The user should avoid names that may collide with system-reserved window class names (e.g., anything beginning with "edit").

Since

FLTK 1.3 the passed string is copied. You can use a local variable or free the string immediately after this call. Note that FLTK 1.1 stores the *pointer* without copying the string.

If the default xclass has not yet been set, this also sets the default xclass for all windows created subsequently.

See also

Fl_Window::default_xclass(const char *)

13.156.4 Member Data Documentation

13.156.4.1 current_

`Fl_Window * Fl_Window::current_ [static], [protected]`

Stores the last window that was made current.

See current() const

The documentation for this class was generated from the following files:

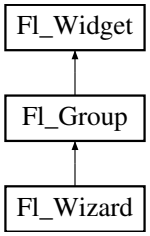
- Fl_Window.H
- Fl_arg.cxx
- fl_cursor.cxx
- Fl_Window.cxx
- Fl_Window_fullscreen.cxx
- Fl_Window_hotspot.cxx
- Fl_Window_iconize.cxx

13.157 Fl_Wizard Class Reference

This widget is based off the Fl_Tabs widget, but instead of displaying tabs it only changes "tabs" under program control.

`#include <Fl_Wizard.H>`

Inheritance diagram for Fl_Wizard:



Public Member Functions

- Fl_Wizard (int, int, int, int, const char *=0)

  The constructor creates the Fl_Wizard widget at the specified position and size.
- void next ()

  This method shows the next child of the wizard.
- void **prev** ()

  Shows the previous child.
- Fl_Widget * **value** ()

  Gets the current visible child widget.
- void **value** (Fl_Widget *)

  Sets the child widget that is visible.
- void value (int)

  Sets the child widget that is visible.

Public Member Functions inherited from Fl_Group

- Fl_Widget *& **_ddfdesign_kludge** ()

  This is for forms compatibility only.
- void **add** (Fl_Widget &)

  The widget is removed from its current group (if any) and then added to the end of this group.
- void **add** (Fl_Widget *o)

  See void Fl_Group::add(Fl_Widget &w)

- void **add_resizable** (Fl_Widget &o)

  Adds a widget to the group and makes it the resizable widget.
- Fl_Widget *const * array () const

  Returns a pointer to the internal array of children.
- Fl_Group const * as_group () const override
- Fl_Group * as_group () override

  Returns an Fl_Group pointer if this widget is an Fl_Group.
- void begin ()

  Sets the current group so you can build the widget tree by just constructing the widgets.
- Fl_Widget * child (int n) const

  Returns the n'th child.
- int **children** () const

  Returns how many child widgets the group has.
- void clear ()

  Deletes all child widgets from memory recursively.
- unsigned int clip_children () const

  Returns the current clipping mode.
- void clip_children (int c)

  Controls whether the group widget clips the drawing of child widgets to its bounding box.
- virtual int delete_child (int n)

  Removes the widget at `index` from the group and deletes it.
- void end ()

  Exactly the same as current(this->parent()).
- int **find** (const Fl_Widget &o) const

  *See int Fl_Group::find(const Fl_Widget *w) const.*
- int find (const Fl_Widget *) const

  Searches the children for the widget and returns the index.
- Fl_Group (int, int, int, int, const char *=0)

  Creates a new Fl_Group widget using the given position, size, and label string.
- void **forms_end** ()

  This is for forms compatibility only.
- int handle (int) override

  Handles the specified event.
- void init_sizes ()

  Resets the internal array of widget sizes and positions.
- void insert (Fl_Widget &, int i)

  The widget is removed from its current group (if any) and then inserted into this group.
- void insert (Fl_Widget &o, Fl_Widget *before)

  This does insert(w, find(before)).
- void remove (Fl_Widget &)

  Removes a widget from the group but does not delete it.
- void remove (Fl_Widget *o)

  Removes the widget `o` from the group.
- void remove (int index)

  Removes the widget at `index` from the group but does not delete it.
- Fl_Widget * resizable () const

  Returns the group's resizable widget.
- void resizable (Fl_Widget &o)

  Sets the group's resizable widget.
- void resizable (Fl_Widget *o)

  The resizable widget defines both the resizing box and the resizing behavior of the group and its children.

- void resize (int, int, int, int) override

  Resizes the Fl_Group widget and all of its children.
- virtual ~Fl_Group ()

  The destructor also deletes all the children.

Public Member Functions inherited from Fl_Widget

- void **_clear_fullscreen** ()
- void **_set_fullscreen** ()
- void activate ()

  Activates the widget.
- unsigned int active () const

  Returns whether the widget is active.
- int active_r () const

  Returns whether the widget and all of its parents are active.
- Fl_Align align () const

  Gets the label alignment.
- void align (Fl_Align alignment)

  Sets the label alignment.
- long argument () const

  Gets the current user data (long) argument that is passed to the callback function.
- void argument (long v)

  Sets the current user data (long) argument that is passed to the callback function.
- virtual class Fl_Gl_Window * as_gl_window ()

  Returns an Fl_Gl_Window pointer if this widget is an Fl_Gl_Window.
- virtual class Fl_Gl_Window const * **as_gl_window** () const
- virtual Fl_Window * as_window ()

  Returns an Fl_Window pointer if this widget is an Fl_Window.
- virtual Fl_Window const * **as_window** () const
- void bind_deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- void bind_deimage (int f)

  Bind the inactive image to the widget, so the widget will delete the image when it is no longer needed.
- void bind_image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- void bind_image (int f)

  Bind the image to the widget, so the widget will delete the image when it is no longer needed.
- Fl_Boxtype box () const

  Gets the box type of the widget.
- void box (Fl_Boxtype new_box)

  Sets the box type for the widget.
- Fl_Callback_p callback () const

  Gets the current callback function for the widget.
- void callback (Fl_Callback *cb)

  Sets the current callback function for the widget.
- void callback (Fl_Callback *cb, Fl_Callback_User_Data *p, bool auto_free)

  Sets the current callback function and managed user data for the widget.
- void callback (Fl_Callback *cb, void *p)

  Sets the current callback function and data for the widget.
- void callback (Fl_Callback0 *cb)

  Sets the current callback function for the widget.

- void callback (Fl_Callback1 *cb, long p=0)

  Sets the current callback function for the widget.
- unsigned int changed () const

  Checks if the widget value changed since the last callback.
- void clear_active ()

  Marks the widget as inactive without sending events or changing focus.
- void clear_changed ()

  Marks the value of the widget as unchanged.
- void clear_damage (uchar c=0)

  Clears or sets the damage flags.
- void clear_output ()

  Sets a widget to accept input.
- void clear_visible ()

  Hides the widget.
- void clear_visible_focus ()

  Disables keyboard focus navigation with this widget.
- Fl_Color color () const

  Gets the background color of the widget.
- void color (Fl_Color bg)

  Sets the background color of the widget.
- void color (Fl_Color bg, Fl_Color sel)

  Sets the background and selection color of the widget.
- int contains (const Fl_Widget *w) const

  Checks if w is a child of this widget.
- void copy_label (const char *new_label)

  Sets the current label.
- void copy_tooltip (const char *text)

  Sets the current tooltip text.
- uchar damage () const

  Returns non-zero if draw() needs to be called.
- void damage (uchar c)

  Sets the damage bits for the widget.
- void damage (uchar c, int x, int y, int w, int h)

  Sets the damage bits for an area inside the widget.
- int **damage_resize** (int, int, int, int)

  Internal use only.
- void deactivate ()

  Deactivates the widget.
- Fl_Image * deimage ()

  Gets the image that is used as part of the widget label when in the inactive state.
- const Fl_Image * deimage () const

  Gets the image that is used as part of the widget label when in the inactive state.
- void deimage (Fl_Image &img)

  Sets the image to use as part of the widget label when in the inactive state.
- void deimage (Fl_Image *img)

  Sets the image to use as part of the widget label when in the inactive state.
- int deimage_bound () const

  Returns whether the inactive image is managed by the widget.
- void do_callback (Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with default arguments.
- void do_callback (Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

Calls the widget callback function with arbitrary arguments.

- void do_callback (Fl_Widget *widget, void *arg=0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN)

  Calls the widget callback function with arbitrary arguments.
- void draw_label (int, int, int, int, Fl_Align) const

  Draws the label in an arbitrary bounding box with an arbitrary alignment.
- int h () const

  Gets the widget height.
- virtual void hide ()

  Makes a widget invisible.
- int horizontal_label_margin ()

  Get the spacing between the label and the horizontal edge of the widget.
- void horizontal_label_margin (int px)

  Set the spacing between the label and the horizontal edge of the widget.
- Fl_Image * image ()

  Gets the image that is used as part of the widget label when in the active state.
- const Fl_Image * image () const

  Gets the image that is used as part of the widget label when in the active state.
- void image (Fl_Image &img)

  Sets the image to use as part of the widget label when in the active state.
- void image (Fl_Image *img)

  Sets the image to use as part of the widget label when in the active state.
- int image_bound () const

  Returns whether the image is managed by the widget.
- int inside (const Fl_Widget *wgt) const

  Checks if this widget is a child of `wgt`.
- int is_label_copied () const

  Returns whether the current label was assigned with copy_label().
- const char * label () const

  Gets the current label text.
- void label (const char *text)

  Sets the current label pointer.
- void label (Fl_Labeltype a, const char *b)

  Shortcut to set the label text and type in one call.
- int label_image_spacing ()

  Return the gap size between the label and the image.
- void label_image_spacing (int gap)

  Set the gap between the label and the image in pixels.
- Fl_Color labelcolor () const

  Gets the label color.
- void labelcolor (Fl_Color c)

  Sets the label color.
- Fl_Font labelfont () const

  Gets the font to use.
- void labelfont (Fl_Font f)

  Sets the font to use.
- Fl_Fontsize labelsize () const

  Gets the font size in pixels.
- void labelsize (Fl_Fontsize pix)

  Sets the font size in pixels.
- Fl_Labeltype labeltype () const

  Gets the label type.

- void labeltype (Fl_Labeltype a)

  Sets the label type.
- void measure_label (int &ww, int &hh) const

  Sets width ww and height hh accordingly with the label size.
- bool needs_keyboard () const

  Returns whether this widget needs a keyboard.
- void needs_keyboard (bool needs)

  Sets whether this widget needs a keyboard.
- unsigned int output () const

  Returns if a widget is used for output only.
- Fl_Group * parent () const

  Returns a pointer to the parent widget.
- void parent (Fl_Group *p)

  Internal use only - "for hacks only".
- void position (int X, int Y)

  Repositions the window or widget.
- void redraw ()

  Schedules the drawing of the widget.
- void redraw_label ()

  Schedules the drawing of the label.
- Fl_Color selection_color () const

  Gets the selection color.
- void selection_color (Fl_Color a)

  Sets the selection color.
- int send (int event)

  Sends the specified event to the widget.
- void set_active ()

  Marks the widget as active without sending events or changing focus.
- void set_changed ()

  Marks the value of the widget as changed.
- void set_output ()

  Sets a widget to output only.
- void set_visible ()

  Makes the widget visible.
- void set_visible_focus ()

  Enables keyboard focus navigation with this widget.
- int shortcut_label () const

  Returns whether the widget's label uses '&' to indicate shortcuts.
- void shortcut_label (int value)

  Sets whether the widget's label uses '&' to indicate shortcuts.
- virtual void show ()

  Makes a widget visible.
- void size (int W, int H)

  Changes the size of the widget.
- int take_focus ()

  Gives the widget the keyboard focus.
- unsigned int takesevents () const

  Returns if the widget is able to take events.
- int test_shortcut ()

  Returns true if the widget's label contains the entered '&x' shortcut.
- const char * tooltip () const

Gets the current tooltip text.

- void tooltip (const char *text)

  Sets the Tooltip Text for a Widget.
- Fl_Window * top_window () const

  Returns a pointer to the top-level window for the widget.
- Fl_Window * top_window_offset (int &xoff, int &yoff) const

  Finds the x/y offset of the current widget relative to the top-level window.
- uchar type () const

  Gets the widget type.
- void type (uchar t)

  Sets the widget type.
- int **use_accents_menu** ()

  Returns non zero if MAC_USE_ACCENTS_MENU flag is set, 0 otherwise.
- void * user_data () const

  Gets the user data for this widget.
- void **user_data** (Fl_Callback_User_Data *v, bool auto_free)

  Sets the user data for this widget.
- void **user_data** (void *v)

  Sets the user data for this widget.
- int vertical_label_margin ()

  Get the spacing between the label and the vertical edge of the widget.
- void vertical_label_margin (int px)

  Set the spacing between the label and the vertical edge of the widget.
- unsigned int visible () const

  Returns whether a widget is visible.
- unsigned int visible_focus () const

  Checks whether this widget has a visible focus.
- void visible_focus (int v)

  Modifies keyboard focus navigation.
- int visible_r () const

  Returns whether a widget and all its parents are visible.
- int w () const

  Gets the widget width.
- Fl_When when () const

  Returns the conditions under which the callback is called.
- void when (uchar i)

  Sets the flags used to decide when a callback is called.
- Fl_Window * window () const

  Returns a pointer to the nearest parent window up the widget hierarchy.
- int x () const

  Gets the widget position in its window.
- int y () const

  Gets the widget position in its window.
- virtual ~Fl_Widget ()

  Destroys the widget.

Protected Member Functions

- void draw () override

  Draws the wizard border and visible child.

Protected Member Functions inherited from Fl_Group

- Fl_Rect ∗ bounds ()

  Returns the internal array of widget sizes and positions.
- void draw_child (Fl_Widget &widget) const

  Forces a child to redraw.
- void draw_children ()

  Draws all children of the group.
- void **draw_outside_label** (const Fl_Widget &widget) const

  Parents normally call this to draw outside labels of child widgets.
- virtual int on_insert (Fl_Widget ∗, int)

  Allow derived groups to act when a widget is added as a child.
- virtual int on_move (int, int)

  Allow derived groups to act when a widget is moved within the group.
- virtual void on_remove (int)

  Allow derived groups to act when a child widget is removed from the group.
- int ∗ sizes ()

  Returns the internal array of widget sizes and positions.
- void update_child (Fl_Widget &widget) const

  Draws a child only if it needs it.

Protected Member Functions inherited from Fl_Widget

- void **clear_flag** (unsigned int c)

  Clears a flag in the flags mask.
- void **draw_backdrop** () const

  If FL_ALIGN_IMAGE_BACKDROP is set, the image or deimage will be drawn.
- void **draw_box** () const

  Draws the widget box according to its box style.
- void **draw_box** (Fl_Boxtype t, Fl_Color c) const

  Draws a box of type t, of color c at the widget's position and size.
- void **draw_box** (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c) const

  Draws a box of type t, of color c at the position X,Y and size W,H.
- void draw_focus () const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int X, int Y, int W, int H) const

  Draws a focus rectangle around the widget.
- void draw_focus (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const

  Draws a focus box for the widget at the given position and size.
- void draw_label () const

  Draws the widget's label at the defined label position.
- void draw_label (int, int, int, int) const

  Draws the label in an arbitrary bounding box.
- Fl_Widget (int x, int y, int w, int h, const char ∗label=0L)

  Creates a widget at the given position and size.
- unsigned int **flags** () const

  Gets the widget flags mask.
- void h (int v)

  Internal use only.
- void **set_flag** (unsigned int c)

  Sets a flag in the flags mask.
- void w (int v)

Internal use only.

- void x (int v)

  Internal use only.

- void y (int v)

  Internal use only.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Group

- static Fl_Group * current ()

  Returns the currently active group.

- static void current (Fl_Group *g)

  Sets the current group.

Static Public Member Functions inherited from Fl_Widget

- static void default_callback (Fl_Widget *widget, void *data)

  The default callback for all widgets that don't set a callback.

- static unsigned int label_shortcut (const char *t)

  Returns the Unicode value of the '&x' shortcut in a given text.

- static int test_shortcut (const char *, const bool require_alt=false)

  Returns true if the given text `t` contains the entered '&x' shortcut.

Protected Types inherited from Fl_Widget

- enum {
  INACTIVE = 1<<0 , INVISIBLE = 1<<1 , OUTPUT = 1<<2 , NOBORDER = 1<<3 ,
  FORCE_POSITION = 1<<4 , NON_MODAL = 1<<5 , SHORTCUT_LABEL = 1<<6 , CHANGED = 1<<7
  ,
  OVERRIDE = 1<<8 , VISIBLE_FOCUS = 1<<9 , COPIED_LABEL = 1<<10 , CLIP_CHILDREN = 1<<11
  ,
  MENU_WINDOW = 1<<12 , TOOLTIP_WINDOW = 1<<13 , MODAL = 1<<14 , NO_OVERLAY = 1<<15
  ,
  GROUP_RELATIVE = 1<<16 , COPIED_TOOLTIP = 1<<17 , FULLSCREEN = 1<<18 , MAC_USE_ACCENTS_MENU
  = 1<<19 ,
  NEEDS_KEYBOARD = 1<<20 , IMAGE_BOUND = 1<<21 , DEIMAGE_BOUND = 1<<22 ,
  AUTO_DELETE_USER_DATA = 1<<23 ,
  MAXIMIZED = 1<<24 , POPUP = 1<<25 , USERFLAG3 = 1<<29 , USERFLAG2 = 1<<30 ,
  USERFLAG1 = 1<<31 }

  flags possible values enumeration.

13.157.1 Detailed Description

This widget is based off the Fl_Tabs widget, but instead of displaying tabs it only changes "tabs" under program control.
Its primary purpose is to support "wizards" that step a user through configuration or troubleshooting tasks.
As with Fl_Tabs, wizard panes are composed of child (usually Fl_Group) widgets. Navigation buttons must be added separately.

13.157.2 Constructor & Destructor Documentation

13.157.2.1 Fl_Wizard()

```
Fl_Wizard::Fl_Wizard (
            int xx,
            int yy,
```

```
            int ww,
            int hh,
            const char * l = 0 )
```

The constructor creates the Fl_Wizard widget at the specified position and size.
The inherited destructor destroys the widget and its children.

13.157.3 Member Function Documentation

13.157.3.1 draw()

```
void Fl_Wizard::draw (
            void ) [override], [protected], [virtual]
```

Draws the wizard border and visible child.
Reimplemented from Fl_Group.

13.157.3.2 next()

```
void Fl_Wizard::next ( )
```

This method shows the next child of the wizard.
If the last child is already visible, this function does nothing.

13.157.3.3 value()

```
void Fl_Wizard::value (
            int ix )
```

Sets the child widget that is visible.

Parameters

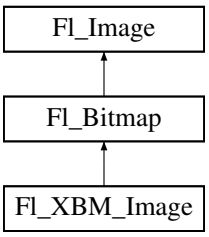
in	*ix*	index of the child widget to show. If the index is out of range, this function does nothing.

The documentation for this class was generated from the following files:

- Fl_Wizard.H
- Fl_Wizard.cxx

13.158 Fl_XBM_Image Class Reference

The Fl_XBM_Image class supports loading, caching, and drawing of X Bitmap (XBM) bitmap files.
`#include <Fl_XBM_Image.H>`
Inheritance diagram for Fl_XBM_Image:



Public Member Functions

- Fl_XBM_Image (const char *filename)
  The constructor loads the named XBM file from the given name filename.

Public Member Functions inherited from Fl_Bitmap

- int **cache_h** () const

- int **cache_w** () const
- Fl_Image ∗ **copy** () const
- Fl_Image ∗ copy (int W, int H) const override

  Creates a resized copy of the image.
- void **draw** (int X, int Y)
- void draw (int X, int Y, int W, int H, int cx=0, int cy=0) override

  Draws the image to the current drawing surface with a bounding box.
- Fl_Bitmap (const char ∗bits, int bits_length, int W, int H)

  The constructors create a new bitmap from the specified bitmap data.
- Fl_Bitmap (const char ∗bits, int W, int H)

  The constructors create a new bitmap from the specified bitmap data.
- Fl_Bitmap (const uchar ∗bits, int bits_length, int W, int H)

  The constructors create a new bitmap from the specified bitmap data.
- Fl_Bitmap (const uchar ∗bits, int W, int H)

  The constructors create a new bitmap from the specified bitmap data.
- void label (Fl_Menu_Item ∗m) override

  This method is an obsolete way to set the image attribute of a menu item.
- void label (Fl_Widget ∗w) override

  This method is an obsolete way to set the image attribute of a widget or menu item.
- void uncache () override

  If the image has been cached for display, delete the cache data.
- virtual ∼**Fl_Bitmap** ()

  The destructor frees all memory and server resources that are used by the bitmap.

Public Member Functions inherited from Fl_Image

- virtual class Fl_Shared_Image ∗ as_shared_image ()

  Returns whether an image is an Fl_Shared_Image or not.
- virtual void color_average (Fl_Color c, float i)

  The color_average() method averages the colors in the image with the provided FLTK color value.
- Fl_Image ∗ copy () const

  Creates a copy of the image in the same size.
- int count () const

  Returns the number of data values associated with the image.
- int d () const

  Returns the image depth.
- const char ∗const ∗ data () const

  Returns a pointer to the current image data array.
- int **data_h** () const

  Returns the height of the image data.
- int **data_w** () const

  Returns the width of the image data.
- virtual void desaturate ()

  The desaturate() method converts an image to grayscale.
- void draw (int X, int Y)

  Draws the image to the current drawing surface.
- int fail () const

  Returns a value that is not 0 if there is currently no image available.
- Fl_Image (int W, int H, int D)

  The constructor creates an empty image with the specified width, height, and depth.
- int h () const

Returns the current image drawing height in FLTK units.

- void inactive ()

  The inactive() method calls color_average(FL_BACKGROUND_COLOR, 0.33f) to produce an image that appears grayed out.

- int ld () const

  Returns the current line data size in bytes.

- virtual void release ()

  Releases an Fl_Image - the same as '`delete this`'.

- virtual void scale (int width, int height, int proportional=1, int can_expand=0)

  Sets the drawing size of the image.

- int w () const

  Returns the current image drawing width in FLTK units.

- virtual ~**Fl_Image** ()

  The destructor is a virtual method that frees all memory used by the image.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Image

- static Fl_Labeltype **define_FL_IMAGE_LABEL** ()
- static Fl_RGB_Scaling **RGB_scaling** ()

  Returns the currently used RGB image scaling method.

- static void RGB_scaling (Fl_RGB_Scaling)

  Sets the RGB image scaling method used for copy(int, int).

- static Fl_RGB_Scaling **scaling_algorithm** ()

  Gets what algorithm is used when resizing a source image to draw it.

- static void scaling_algorithm (Fl_RGB_Scaling algorithm)

  Sets what algorithm is used when resizing a source image to draw it.

Public Attributes inherited from Fl_Bitmap

- int **alloc_array**

  Non-zero if array points to bitmap data allocated internally.

- const uchar * **array**

  pointer to raw bitmap data

Static Public Attributes inherited from Fl_Image

- static const int **ERR_FILE_ACCESS** = -2
- static const int **ERR_FORMAT** = -3
- static const int **ERR_MEMORY_ACCESS** = -4
- static const int **ERR_NO_IMAGE** = -1
- static bool **register_images_done** = false

  True after fl_register_images() was called, false before.

Protected Member Functions inherited from Fl_Image

- void **d** (int D)

  Sets the current image depth.

- void data (const char *const *p, int c)

  Sets the current data pointer and count of pointers in the array.

- void draw_empty (int X, int Y)

  The protected method draw_empty() draws a box with an X in it.

- int draw_scaled (int X, int Y, int W, int H)

Draw the image to the current drawing surface rescaled to a given width and height.

- void h (int H)

  Sets the height of the image data.
- void ld (int LD)

  Sets the current line data size in bytes.
- void w (int W)

  Sets the width of the image data.

Static Protected Member Functions inherited from Fl_Image

- static void **labeltype** (const Fl_Label *lo, int lx, int ly, int lw, int lh, Fl_Align la)
- static void **measure** (const Fl_Label *lo, int &lw, int &lh)

13.158.1 Detailed Description

The Fl_XBM_Image class supports loading, caching, and drawing of X Bitmap (XBM) bitmap files.

13.158.2 Constructor & Destructor Documentation

13.158.2.1 Fl_XBM_Image()

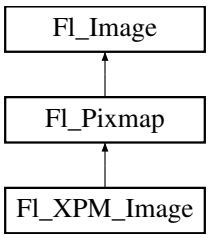
```
Fl_XBM_Image::Fl_XBM_Image (
            const char * name )
```

The constructor loads the named XBM file from the given name filename.
The destructor frees all memory and server resources that are used by the image.
The documentation for this class was generated from the following files:

- Fl_XBM_Image.H
- Fl_XBM_Image.cxx

13.159 Fl_XPM_Image Class Reference

The Fl_XPM_Image class supports loading, caching, and drawing of X Pixmap (XPM) images, including transparency.
`#include <Fl_XPM_Image.H>`
Inheritance diagram for Fl_XPM_Image:



Public Member Functions

- Fl_XPM_Image (const char *filename)

  The constructor loads the XPM image from the name filename.

Public Member Functions inherited from Fl_Pixmap

- int **cache_h** () const
- int **cache_w** () const
- void color_average (Fl_Color c, float i) override

  The color_average() method averages the colors in the image with the provided FLTK color value.

- Fl_Image * **copy** () const
- Fl_Image * copy (int W, int H) const override

  Creates a resized copy of the image.
- void desaturate () override

  The desaturate() method converts an image to grayscale.
- void **draw** (int X, int Y)
- void draw (int X, int Y, int W, int H, int cx=0, int cy=0) override

  Draws the image to the current drawing surface with a bounding box.
- Fl_Pixmap (char *const *D)

  The constructors create a new pixmap from the specified XPM data.
- Fl_Pixmap (const char *const *D)

  The constructors create a new pixmap from the specified XPM data.
- Fl_Pixmap (const uchar *const *D)

  The constructors create a new pixmap from the specified XPM data.
- Fl_Pixmap (uchar *const *D)

  The constructors create a new pixmap from the specified XPM data.
- void label (Fl_Menu_Item *m) override

  This method is an obsolete way to set the image attribute of a menu item.
- void label (Fl_Widget *w) override

  This method is an obsolete way to set the image attribute of a widget or menu item.
- void uncache () override

  If the image has been cached for display, delete the cache data.
- virtual ~**Fl_Pixmap** ()

  The destructor frees all memory and server resources that are used by the pixmap.

Public Member Functions inherited from Fl_Image

- virtual class Fl_Shared_Image * as_shared_image ()

  Returns whether an image is an Fl_Shared_Image or not.
- Fl_Image * copy () const

  Creates a copy of the image in the same size.
- int count () const

  Returns the number of data values associated with the image.
- int d () const

  Returns the image depth.
- const char *const * data () const

  Returns a pointer to the current image data array.
- int **data_h** () const

  Returns the height of the image data.
- int **data_w** () const

  Returns the width of the image data.
- void draw (int X, int Y)

  Draws the image to the current drawing surface.
- int fail () const

  Returns a value that is not 0 if there is currently no image available.
- Fl_Image (int W, int H, int D)

  The constructor creates an empty image with the specified width, height, and depth.
- int h () const

  Returns the current image drawing height in FLTK units.
- void inactive ()

The inactive() method calls color_average(FL_BACKGROUND_COLOR, 0.33f) to produce an image that appears grayed out.

- int ld () const

  Returns the current line data size in bytes.
- virtual void release ()

  Releases an Fl_Image - the same as '`delete this`'.
- virtual void scale (int width, int height, int proportional=1, int can_expand=0)

  Sets the drawing size of the image.
- int w () const

  Returns the current image drawing width in FLTK units.
- virtual ~**Fl_Image** ()

  The destructor is a virtual method that frees all memory used by the image.

Additional Inherited Members

Static Public Member Functions inherited from Fl_Image

- static Fl_Labeltype **define_FL_IMAGE_LABEL** ()
- static Fl_RGB_Scaling **RGB_scaling** ()

  Returns the currently used RGB image scaling method.
- static void RGB_scaling (Fl_RGB_Scaling)

  Sets the RGB image scaling method used for copy(int, int).
- static Fl_RGB_Scaling **scaling_algorithm** ()

  Gets what algorithm is used when resizing a source image to draw it.
- static void scaling_algorithm (Fl_RGB_Scaling algorithm)

  Sets what algorithm is used when resizing a source image to draw it.

Public Attributes inherited from Fl_Pixmap

- int **alloc_data**

Static Public Attributes inherited from Fl_Image

- static const int **ERR_FILE_ACCESS** = -2
- static const int **ERR_FORMAT** = -3
- static const int **ERR_MEMORY_ACCESS** = -4
- static const int **ERR_NO_IMAGE** = -1
- static bool **register_images_done** = false

  True after fl_register_images() was called, false before.

Protected Member Functions inherited from Fl_Pixmap

- void **measure** ()

Protected Member Functions inherited from Fl_Image

- void **d** (int D)

  Sets the current image depth.
- void data (const char *const *p, int c)

  Sets the current data pointer and count of pointers in the array.
- void draw_empty (int X, int Y)

  The protected method draw_empty() draws a box with an X in it.
- int draw_scaled (int X, int Y, int W, int H)

  Draw the image to the current drawing surface rescaled to a given width and height.

- void h (int H)

  Sets the height of the image data.
- void ld (int LD)

  Sets the current line data size in bytes.
- void w (int W)

  Sets the width of the image data.

Static Protected Member Functions inherited from Fl_Image

- static void **labeltype** (const Fl_Label *lo, int lx, int ly, int lw, int lh, Fl_Align la)
- static void **measure** (const Fl_Label *lo, int &lw, int &lh)

13.159.1 Detailed Description

The Fl_XPM_Image class supports loading, caching, and drawing of X Pixmap (XPM) images, including transparency.

13.159.2 Constructor & Destructor Documentation

13.159.2.1 Fl_XPM_Image()

```
Fl_XPM_Image::Fl_XPM_Image (
            const char * name )
```

The constructor loads the XPM image from the name filename.
The destructor frees all memory and server resources that are used by the image.
The documentation for this class was generated from the following files:

- Fl_XPM_Image.H
- Fl_XPM_Image.cxx

13.160 Fl_GIF_Image::GIF_FRAME Struct Reference

Classes

- struct CPAL

Public Member Functions

- void **colors** (int nclrs, int bg, int tp)
- void **disposal** (int mode, int time)
- **GIF_FRAME** (int frame, const uchar *data)
- **GIF_FRAME** (int frame, int W, int H, int fx, int fy, int fw, int fh, const uchar *data)

Public Attributes

- int **bkgd**
- const uchar * **bptr**
- int **clrs**
- const struct Fl_GIF_Image::GIF_FRAME::CPAL * **cpal**
- int **delay**
- int **dispose**
- int **h**
- int **height**
- int **ifrm**
- int **trans**
- int **w**
- int **width**

- int **x**
- int **y**

The documentation for this struct was generated from the following file:

- Fl_GIF_Image.H

13.161 Fl_ICO_Image::IconDirEntry Struct Reference

Windows ICONDIRENTRY structure

```
#include <Fl_ICO_Image.H>
```

Public Attributes

- int **bColorCount**

  Number of colors (0 if 8bpp)
- int **bHeight**

  Image height.
- int **bReserved**

  Reserved.
- int **bWidth**

  Image width.
- int **dwBytesInRes**

  Resource size in bytes.
- int **dwImageOffset**

  Offset to the image.
- int **wBitCount**

  Bits per pixel.
- int **wPlanes**

  Color Planes.

13.161.1 Detailed Description

Windows ICONDIRENTRY structure

The documentation for this struct was generated from the following file:

- Fl_ICO_Image.H

13.162 Fl_Text_Editor::Key_Binding Struct Reference

Simple linked list item associating a key/state to a function.

```
#include <Fl_Text_Editor.H>
```

Public Attributes

- Key_Func **function**

  associated function
- int **key**

  the key pressed
- Key_Binding * **next**

  next key binding in the list
- int **state**

  the state of key modifiers

13.162.1 Detailed Description

Simple linked list item associating a key/state to a function.
The documentation for this struct was generated from the following file:

- Fl_Text_Editor.H

13.163 Fl_Terminal::Margin Class Reference

Public Member Functions

- void **bottom** (int val)
- int **bottom** (void) const
- void **left** (int val)
- int **left** (void) const
- void **right** (int val)
- int **right** (void) const
- void **top** (int val)
- int **top** (void) const

The documentation for this class was generated from the following file:

- Fl_Terminal.H

13.164 Fl_Preferences::Name Class Reference

'Name' provides a simple method to create numerical or more complex procedural names for entries and groups on the fly.
`#include <Fl_Preferences.H>`

Public Member Functions

- Name (const char ∗format,...)

  Creates a group name or entry name on the fly.
- Name (unsigned int n)

  Creates a group name or entry name on the fly.
- **operator const char** ∗ () const

  Return the Name as a "C" string.

13.164.1 Detailed Description

'Name' provides a simple method to create numerical or more complex procedural names for entries and groups on the fly.
Example: prefs.set(Fl_Preferences::Name("File%d",i),file[i]);.
See test/preferences.cxx as a sample for writing arrays into preferences.
'Name' is actually implemented as a class inside Fl_Preferences. It casts into const char∗ and gets automatically destroyed after the enclosing call ends.

13.164.2 Constructor & Destructor Documentation

13.164.2.1 Name() [1/2]

```
Fl_Preferences::Name::Name (
            unsigned int n )
```

Creates a group name or entry name on the fly.
This version creates a simple unsigned integer as an entry name.

```
int n, i;
Fl_Preferences prev( appPrefs, "PreviousFiles" );
prev.get( "n", 0 );
for ( i=0; i<n; i++ )
  prev.get( Fl_Preferences::Name(i), prevFile[i], "" );
```

13.164.2.2 Name() [2/2]

```
Fl_Preferences::Name::Name (
            const char * format,
             ...  )
```

Creates a group name or entry name on the fly.
This version creates entry names as in 'printf'.

```
int n, i;
Fl_Preferences prefs( USER, "matthiasm.com", "test" );
prev.get( "nFiles", 0 );
for ( i=0; i<n; i++ )
  prev.get( Fl_Preferences::Name( "File%d", i ), prevFile[i], "" );
```

The documentation for this class was generated from the following files:

- Fl_Preferences.H
- Fl_Preferences.cxx

13.165 Fl_Preferences::Node Class Reference

Public Member Functions

- void **add** (const char *line)
- Node * **addChild** (const char *path)
- const char * **child** (int ix)
- Node * **childNode** (int ix)
- void **clearDirtyFlags** ()
- void **deleteAllChildren** ()
- void **deleteAllEntries** ()
- char **deleteEntry** (const char *name)
- char **dirty** () const
- Entry & **entry** (int i) const
- Node * **find** (const char *path)
- RootNode * **findRoot** () const
- const char * **get** (const char *name) const
- int **getEntry** (const char *name) const
- const char * **name** () const
- int **nChildren** () const
- int **nEntry** () const
- **Node** (const char *path)
- Node * **parent** () const
- const char * **path** () const
- char **remove** ()
- Node * **search** (const char *path, int offset=0)
- void **set** (const char *line)
- void **set** (const char *name, const char *value)
- void **setParent** (Node *parent)
- void **setRoot** (RootNode *r)
- int **write** (FILE *f)

Static Public Attributes

- static int **lastEntrySet** = -1

The documentation for this class was generated from the following files:

- Fl_Preferences.H
- Fl_Preferences.cxx

13.166 Fl_Paged_Device::page_format Struct Reference

width, height and name of a page format
`#include <Fl_Paged_Device.H>`

Public Attributes

- int **height**

  height in points
- const char * **name**

  format name
- int **width**

  width in points

13.166.1 Detailed Description

width, height and name of a page format
The documentation for this struct was generated from the following file:

- Fl_Paged_Device.H

13.167 Fl_Terminal::PartialUtf8Buf Class Reference

Public Member Functions

- bool **append** (const char *p, int len)
- const char * **buf** (void) const
- int **buflen** (void) const
- void **clear** (void)
- bool **is_complete** (void) const
- bool **is_continuation** (char c)

The documentation for this class was generated from the following file:

- Fl_Terminal.H

13.168 Fl_Terminal::RingBuffer Class Reference

Public Member Functions

- void **change_disp_cols** (int dcols, const CharStyle &style)
- void **change_disp_rows** (int drows, const CharStyle &style)
- void **clear** (void)
- void **clear_disp_rows** (int sdrow, int edrow, const CharStyle &style)
- void **clear_hist** (void)
- void **create** (int drows, int dcols, int hrows)
- int **disp_cols** (void) const
- int **disp_erow** (void) const
- void **disp_rows** (int val)
- int **disp_rows** (void) const
- int **disp_srow** (void) const
- int **hist_cols** (void) const
- int **hist_erow** (void) const
- void **hist_rows** (int val)
- int **hist_rows** (void) const
- int **hist_srow** (void) const

- void **hist_use** (int val)
- int **hist_use** (void) const
- int **hist_use_srow** (void) const
- bool **is_disp_ring_row** (int grow) const
- bool **is_hist_ring_row** (int grow) const
- void **move_disp_row** (int src_row, int dst_row)
- int **offset** (void) const
- void **offset_adjust** (int rows)
- void **resize** (int drows, int dcols, int hrows, const CharStyle &style)
- Utf8Char ∗ **ring_chars** (void)
- Utf8Char ∗ **ring_chars** (void) const
- int **ring_cols** (void) const
- int **ring_erow** (void) const
- int **ring_rows** (void) const
- int **ring_srow** (void) const
- **RingBuffer** (int drows, int dcols, int hrows)
- void **scroll** (int rows, const CharStyle &style)
- Utf8Char ∗ **u8c_disp_row** (int drow)
- const Utf8Char ∗ **u8c_disp_row** (int drow) const
- Utf8Char ∗ **u8c_hist_row** (int hrow)
- const Utf8Char ∗ **u8c_hist_row** (int hrow) const
- Utf8Char ∗ **u8c_hist_use_row** (int hurow)
- const Utf8Char ∗ **u8c_hist_use_row** (int hurow) const
- Utf8Char ∗ **u8c_ring_row** (int row)
- const Utf8Char ∗ **u8c_ring_row** (int row) const

The documentation for this class was generated from the following files:

- Fl_Terminal.H
- Fl_Terminal.cxx

13.169 Fl_Preferences::RootNode Class Reference

Public Member Functions

- char ∗ **filename** () const
- char **getPath** (char ∗path, int pathlen)
- int **read** ()
- Root **root** () const
- **RootNode** (Fl_Preferences ∗)
- **RootNode** (Fl_Preferences ∗, const char ∗path, const char ∗vendor, const char ∗application, Root flags)
- **RootNode** (Fl_Preferences ∗, Root root, const char ∗vendor, const char ∗application)
- int **write** ()

The documentation for this class was generated from the following files:

- Fl_Preferences.H
- Fl_Preferences.cxx

13.170 Fl_Scroll::Scrollbar_Data Struct Reference

A local struct to manage a scrollbar's xywh region and tab values.

```
#include <Fl_Scroll.H>
```

Public Attributes

- int **first**

  scrollbar tab's "number of first line"
- int **h**

  scrollbar's position and size
- int **pos**

  scrollbar tab's "position of first line displayed"
- int **size**

  scrollbar tab's "size of window in lines"
- int **total**

  scrollbar tab's "total number of lines"
- int **w**
- int **x**
- int **y**

13.170.1 Detailed Description

A local struct to manage a scrollbar's xywh region and tab values.
The documentation for this struct was generated from the following file:

- Fl_Scroll.H

13.171 Fl_Scroll::ScrollInfo Struct Reference

Structure to manage scrollbar and widget interior sizes.
`#include <Fl_Scroll.H>`

Public Attributes

- Fl_Rect **child**

  child bounding box or (x(), y(), 0, 0) if empty
- int **hneeded**

  horizontal scrollbar visibility
- Scrollbar_Data **hscroll**

  horizontal scrollbar region + values
- Fl_Rect **innerbox**

  widget's inner box, excluding scrollbars
- Fl_Rect **innerchild**

  widget's inner box, including scrollbars
- int **scrollsize**

  the effective scrollbar thickness (local or global)
- int **vneeded**

  vertical scrollbar visibility
- Scrollbar_Data **vscroll**

  vertical scrollbar region + values

13.171.1 Detailed Description

Structure to manage scrollbar and widget interior sizes.
This is filled out by recalc_scrollbars() for use in calculations that need to know the visible scroll area size, etc.

Version

1.3.3

The documentation for this struct was generated from the following file:

- Fl_Scroll.H

13.172 Fl_Terminal::Selection Class Reference

Public Member Functions

- bool **clear** (void)
- bool **dragged_off** (int row, int col, bool char_right) const
- int **ecol** (void) const
- void **end** (void)
- int **erow** (void) const
- bool **extend** (int row, int col, bool char_right)
- bool get_selection (int &srow, int &scol, int &erow, int &ecol) const

  Return selection start/end.
- bool **is_selection** (void) const
- void **push_clear** ()
- void **push_rowcol** (int row, int col, bool char_right)
- int **scol** (void) const
- void **scroll** (int nrows)
- void **select** (int srow, int scol, int erow, int ecol)
- **Selection** (Fl_Terminal *terminal)
- void **selectionbgcolor** (Fl_Color val)
- Fl_Color **selectionbgcolor** (void) const
- void **selectionfgcolor** (Fl_Color val)
- Fl_Color **selectionfgcolor** (void) const
- int **srow** (void) const
- bool **start** (int row, int col, bool char_right)
- void **start_push** ()
- int **state** (void) const

13.172.1 Member Function Documentation

13.172.1.1 get_selection()

```
bool Fl_Terminal::Selection::get_selection (
            int & srow,
            int & scol,
            int & erow,
            int & ecol ) const
```

Return selection start/end.
Ensures (start <= end) to allow walking 'forward' thru selection, left-to-right, top-to-bottom.
Returns:

- true – valid selection values returned
- false – no selection was made, returned values undefined

The documentation for this class was generated from the following files:

- Fl_Terminal.H
- Fl_Terminal.cxx

13.173 Fl_Tile::Size_Range Struct Reference

Public Attributes

- int **maxh**
- int **maxw**
- int **minh**
- int **minw**

The documentation for this struct was generated from the following file:

- Fl_Tile.H

13.174 Fl_Text_Display::Style_Table_Entry Struct Reference

This structure associates the color, font, and font size of a string to draw with an attribute mask matching attr.
`#include <Fl_Text_Display.H>`

Public Attributes

- unsigned **attr**

  further attributes for the text style (see `ATTR_BGCOLOR`, etc.)
- Fl_Color **bgcolor**

  text background color if `ATTR_BGCOLOR` or `ATTR_BGCOLOR_EXT` is set
- Fl_Color **color**

  text color
- Fl_Font **font**

  text font
- Fl_Fontsize **size**

  text font size

13.174.1 Detailed Description

This structure associates the color, font, and font size of a string to draw with an attribute mask matching attr. There must be one entry for each style that can be used in an Fl_Text_Display for displaying text. The style table is an array of struct Style_Table_Entry.
The style table is associated with an Fl_Text_Display by using Fl_Text_Display::highlight_data().

See also

Fl_Text_Display::highlight_data()

The documentation for this struct was generated from the following file:

- Fl_Text_Display.H

13.175 Fl::TimeoutData Struct Reference

Public Attributes

- Fl_Timeout_Handler **cb**
- void * **data**
- double **t**

The documentation for this struct was generated from the following file:

- Fl.H

13.176 Fl_Terminal::Utf8Char Class Reference

Public Member Functions

- Fl_Color **attr_bg_color** (const Fl_Widget *grp) const
- Fl_Color **attr_fg_color** (const Fl_Widget *grp) const
- uchar **attrib** (void) const
- Fl_Color **bgcolor** (void) const
- uchar **charflags** (void) const
- void **clear** (const CharStyle &style)
- Fl_Color **fgcolor** (void) const
- void **fl_font_set** (const CharStyle &style) const
- bool **is_char** (char c) const

- int **length** (void) const
- int **max_utf8** () const
- Utf8Char & **operator=** (const Utf8Char &o)
- double **pwidth** (void) const
- int **pwidth_int** (void) const
- void **show_char** (void) const
- void **show_char_info** (void) const
- void **text_ascii** (char c, const CharStyle &style)
- void **text_utf8** (const char *text, int len, const CharStyle &style)
- const char * **text_utf8** (void) const
- **Utf8Char** (const Utf8Char &o)

The documentation for this class was generated from the following files:

- Fl_Terminal.H
- Fl_Terminal.cxx

Chapter 14

File Documentation

14.1 fl_config.h

```
/* FL/fl_config.h.  Generated from fl_config.h.in by CMake.  */
/*
 * Build configuration file for the Fast Light Tool Kit (FLTK). This
 * configuration file is publicly accessible by user programs (installed).
 *
 * Copyright 1998-2025 by Bill Spitzak and others.
 *
 * This library is free software. Distribution and use rights are outlined in
 * the file "COPYING" which should have been included with this file.  If this
 * file is missing or damaged, see the license at:
 *
 *     https://www.fltk.org/COPYING.php
 *
 * Please see the following page on how to report bugs and issues:
 *
 *     https://www.fltk.org/bugs.php
 */

#ifndef _FL_fl_config_h_
#define _FL_fl_config_h_

#define FL_MAJOR_VERSION 1

#define FL_MINOR_VERSION 5

#define FL_PATCH_VERSION 0

#define FL_ABI_VERSION 10500

// End of doxygen group 'version_numbers'. More is added to the group in
// in file FL/Enumerations.H

/*
 * FLTK_HAVE_CAIRO
 *
 * Do we have Fl_Cairo_Window support?
*/

#define FLTK_HAVE_CAIRO 1


/*
 * FLTK_HAVE_CAIROEXT
 *
 * Do we have the Cairo library available and want extended Cairo use in FLTK ?
 * This implies to link cairo.lib in all FLTK based apps.
*/

/* #undef FLTK_HAVE_CAIROEXT */


/*
 * FLTK_HAVE_FORMS
 *
 * Do we have the Forms compatibility library available?
*/

/* #undef FLTK_HAVE_FORMS */


/*
 * FLTK_USE_X11
```

```
 *
 * Do we use X11 for the current platform?
 *
 */

#define FLTK_USE_X11 1


/*
 * FLTK_USE_CAIRO
 *
 * Do we use Cairo to draw to the display?
 *
 */

#define FLTK_USE_CAIRO 1


/*
 * FLTK_USE_WAYLAND
 *
 * Do we use Wayland for the current platform?
 *
 */

#define FLTK_USE_WAYLAND 1


/*
 * FLTK_USE_SVG
 *
 * Do we want FLTK to read and write SVG-formatted files ?
 *
 */

#define FLTK_USE_SVG 1


/*
 * FLTK_HAVE_PEN_SUPPORT
 *
 * Do we have pen/tablet support for the current platform?
 * See CMake option FLTK_OPTION_PEN_SUPPORT, but note that a build test
 * (try_compile()) is performed to test if the option is available
 *
 * We use cmakedefine01 instead of cmakedefine as we may want to
 * override it from the command-line.
 * For example, to avoid fluid to be compiled with pen support.
 *
 */

#define FLTK_HAVE_PEN_SUPPORT 1

#endif /* _FL_fl_config_h_ */
```

14.2 events.H File Reference

Global event handling variables and functions.

```
#include <FL/fl_config.h>
#include <FL/Fl_Export.H>
#include <FL/core/function_types.H>
```

Namespaces

- namespace Fl

  Fl is the FLTK global namespace containing state information and global methods for the current application.

Functions

- void Fl::add_handler (Fl_Event_Handler ha)

  Install a function to parse unrecognized events.
- void Fl::add_handler (Fl_Event_Handler ha, Fl_Event_Handler before)

  Install a function to parse unrecognized events with less priority than another function.
- void Fl::add_system_handler (Fl_System_Handler h, void *data)

Install a function to intercept system events.

- Fl_Widget * Fl::belowmouse ()

  Gets the widget that is below the mouse.
- void Fl::belowmouse (Fl_Widget *)

  Sets the widget that is below the mouse.
- Fl_Callback_Reason Fl::callback_reason ()

  Give the reason for calling a callback.
- int Fl::compose (int &del)

  Any text editing widget should call this for each FL_KEYBOARD event.
- void Fl::compose_reset ()

  If the user moves the cursor, be sure to call Fl::compose_reset().
- void Fl::disable_im ()

  Disables the system input methods facilities.
- void Fl::enable_im ()

  Enables the system input methods facilities.
- int Fl::event ()

  Returns the last event that was processed.
- int **Fl::event_alt** ()

  Returns non-zero if the Alt key is pressed.
- int Fl::event_button ()

  Gets which particular mouse button caused the current event.
- int Fl::event_button1 ()

  Returns non-zero if mouse button 1 is currently held down.
- int Fl::event_button2 ()

  Returns non-zero if mouse button 2 is currently held down.
- int Fl::event_button3 ()

  Returns non-zero if mouse button 3 is currently held down.
- int Fl::event_button4 ()

  Returns non-zero if mouse button 4 is currently held down.
- int Fl::event_button5 ()

  Returns non-zero if mouse button 5 is currently held down.
- int Fl::event_buttons ()

  Returns the mouse buttons state bits; if non-zero, then at least one button is pressed now.
- int Fl::event_clicks ()

  Returns non zero if we had a double click event.
- void Fl::event_clicks (int i)

  Manually sets the number returned by Fl::event_clicks().
- void * Fl::event_clipboard ()

  During an FL_PASTE event of non-textual data, returns a pointer to the pasted data.
- const char * Fl::event_clipboard_type ()

  Returns the type of the pasted data during an FL_PASTE event.
- int **Fl::event_command** ()

  Returns non-zero if the FL_COMMAND key is pressed, either FL_CTRL or on OSX FL_META.
- int **Fl::event_ctrl** ()

  Returns non-zero if the Control key is pressed.
- Fl_Event_Dispatch **Fl::event_dispatch** ()

  Return the current event dispatch function.
- void Fl::event_dispatch (Fl_Event_Dispatch d)

  Set a new event dispatch function.
- int Fl::event_dx ()

  Returns the current horizontal mouse scrolling associated with the FL_MOUSEWHEEL event.

- float Fl::event_dx_f ()

  Returns the current horizontal mouse scrolling associated with the FL_MOUSEWHEEL event.
- int Fl::event_dy ()

  Returns the current vertical mouse scrolling associated with the FL_MOUSEWHEEL event.
- float Fl::event_dy_f ()

  Returns the current vertical mouse scrolling associated with the FL_MOUSEWHEEL event.
- int Fl::event_inside (const Fl_Widget *)

  Returns whether or not the mouse event is inside a given child widget.
- int Fl::event_inside (int, int, int, int)

  Returns whether or not the mouse event is inside the given rectangle.
- int Fl::event_is_click ()

  Returns non-zero if the mouse has not moved far enough and not enough time has passed since the last FL_PUSH or FL_KEYBOARD event for it to be considered a "drag" rather than a "click".
- void Fl::event_is_click (int i)

  Clears the value returned by Fl::event_is_click().
- int Fl::event_key ()

  Gets which key on the keyboard was last pushed.
- int Fl::event_key (int key)

  Returns true if the given `key` was held down (or pressed) during the last event.
- int Fl::event_length ()

  Returns the length of the text in Fl::event_text().
- int Fl::event_original_key ()

  Returns the keycode of the last key event, regardless of the NumLock state.
- int **Fl::event_shift** ()

  Returns non-zero if the Shift key is pressed.
- int Fl::event_state ()

  Returns the keyboard and mouse button states of the last event.
- int Fl::event_state (int mask)

  Returns non-zero if any of the passed event state bits are turned on.
- const char * Fl::event_text ()

  Returns the text associated with the current event, including FL_PASTE or FL_DND_RELEASE events.
- int **Fl::event_x** ()

  Returns the mouse position of the event relative to the Fl_Window it was passed to.
- int Fl::event_x_root ()

  Returns the mouse position on the screen of the event.
- int **Fl::event_y** ()

  Returns the mouse position of the event relative to the Fl_Window it was passed to.
- int Fl::event_y_root ()

  Returns the mouse position on the screen of the event.
- Fl_Widget * Fl::focus ()

  Gets the current Fl::focus() widget.
- void Fl::focus (Fl_Widget *)

  Sets the widget that will receive FL_KEYBOARD events.
- int Fl::get_key (int key)

  Returns true if the given `key` is held down now.
- int Fl::get_mouse (int &X, int &Y)

  Return the number of the mouse-containing screen and computes the mouse screen coordinates.
- int Fl::handle (int, Fl_Window *)

  Handle events from the window system.
- int Fl::handle_ (int, Fl_Window *)

  Handle events from the window system.

- Fl_Event_Handler Fl::last_handler ()

  Returns the last function installed by a call to Fl::add_handler().
- Fl_Widget * Fl::pushed ()

  Gets the widget that is being pushed.
- void Fl::pushed (Fl_Widget *)

  Sets the widget that is being pushed.
- void Fl::remove_handler (Fl_Event_Handler h)

  Removes a previously added event handler.
- void Fl::remove_system_handler (Fl_System_Handler h)

  Removes a previously added system event handler.
- int Fl::test_shortcut (Fl_Shortcut)

  Tests the current event, which must be an FL_KEYBOARD or FL_SHORTCUT, against a shortcut value (described in Fl_Button).

Variables

- char const *const **Fl::clipboard_image** = "image"

  Denotes image data.
- char const *const **Fl::clipboard_plain_text** = "text/plain"

  Denotes plain textual data.

14.2.1 Detailed Description

Global event handling variables and functions.

14.3 events.H

Go to the documentation of this file.

```
00001 //
00002 // Global event header file for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 2025 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file.  If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 //     https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 //     https://www.fltk.org/bugs.php
00015 //
00016
00021 #ifndef Fl_core_events_H
00022 #define Fl_core_events_H
00023
00024 #include <FL/fl_config.h>             // build configuration
00025 #include <FL/Fl_Export.H>             // for FL_EXPORT
00026 #include <FL/core/function_types.H>   // widget callbacks and services
00027
00028 class Fl_Widget;
00029
00030 namespace Fl {
00031
00032 //
00033 // Event Variables (Internal)
00034 // Note: These variables should be private, but would harm back compatibility.
00035 //
00036
00037 #ifndef FL_DOXYGEN
00038
00039 // Core event information
00040 FL_EXPORT extern int                e_number;
00041 FL_EXPORT extern int                e_state;
00042
00043 // Mouse position and movement
00044 FL_EXPORT extern int                e_x;
00045 FL_EXPORT extern int                e_y;
```

```
FL_EXPORT extern int                e_x_root;
FL_EXPORT extern int                e_y_root;
FL_EXPORT extern int                e_dx;
FL_EXPORT extern int                e_dy;
FL_EXPORT extern float              e_dx_err;
FL_EXPORT extern float              e_dy_err;
FL_EXPORT extern float              e_dx_f;
FL_EXPORT extern float              e_dy_f;

// Mouse click handling
FL_EXPORT extern int                e_clicks;
FL_EXPORT extern int                e_is_click;

// Keyboard event data
FL_EXPORT extern int                e_keysym;
FL_EXPORT extern int                e_original_keysym;
FL_EXPORT extern char*              e_text;
FL_EXPORT extern int                e_length;

// Clipboard data (for paste events)
FL_EXPORT extern void*              e_clipboard_data;
FL_EXPORT extern const char*        e_clipboard_type;

// Event handling infrastructure
FL_EXPORT extern Fl_Event_Dispatch  e_dispatch;
FL_EXPORT extern Fl_Callback_Reason callback_reason_;

// Widget state tracking
FL_EXPORT extern Fl_Widget*         belowmouse_;
FL_EXPORT extern Fl_Widget*         pushed_;
FL_EXPORT extern Fl_Widget*         focus_;

// Event variables should be private, but would harm back compatibility.

#endif // FL_DOXYGEN


//
// Event Information Functions
//

FL_EXPORT inline int event()                    { return e_number; }

//
// Mouse Position Functions
//

FL_EXPORT inline int event_x()                  { return e_x; }

FL_EXPORT inline int event_y()                  { return e_y; }

FL_EXPORT inline int event_x_root()             { return e_x_root; }

FL_EXPORT inline int event_y_root()             { return e_y_root; }

//
// Mouse Wheel Functions
//

FL_EXPORT inline int event_dx()                 { return e_dx; }

FL_EXPORT inline int event_dy()                 { return e_dy; }

FL_EXPORT inline float event_dx_f()             { return e_dx_f; }

FL_EXPORT inline float event_dy_f()             { return e_dy_f; }

//
// Mouse Query Functions
//

FL_EXPORT extern int get_mouse(int &X,int &Y);

//
// Mouse Click Functions
//

FL_EXPORT inline int event_clicks()             { return e_clicks; }

FL_EXPORT inline void event_clicks(int i)       { e_clicks = i; }

FL_EXPORT inline int event_is_click()           { return e_is_click; }

FL_EXPORT inline void event_is_click(int i)     { e_is_click = i; }

//
// Mouse Button Functions
```

```
//

FL_EXPORT inline int event_button()             { return e_keysym - FL_Button; }

//
// Event State Functions
//

FL_EXPORT inline int event_state()              { return e_state; }

FL_EXPORT inline int event_state(int mask)      { return e_state & mask; }

//
// Keyboard Event Functions
//

FL_EXPORT inline int event_key()                { return e_keysym; }

FL_EXPORT inline int event_original_key()       { return e_original_keysym; }

FL_EXPORT extern int event_key(int key);

FL_EXPORT extern int get_key(int key); // platform dependent

//
// Text and Clipboard Functions
//

FL_EXPORT inline const char* event_text()       { return e_text; }

FL_EXPORT inline int event_length()             { return e_length; }

FL_EXPORT extern char const * const clipboard_plain_text;

FL_EXPORT extern char const * const clipboard_image;

FL_EXPORT inline void *event_clipboard()        { return e_clipboard_data; }

FL_EXPORT inline const char *event_clipboard_type() { return e_clipboard_type; }


//
// Advanced Event Functions
//

FL_EXPORT extern int compose(int &del);
FL_EXPORT extern void compose_reset();
FL_EXPORT extern int event_inside(int,int,int,int);
FL_EXPORT extern int event_inside(const Fl_Widget*);
FL_EXPORT extern int test_shortcut(Fl_Shortcut);

FL_EXPORT extern void enable_im();
FL_EXPORT extern void disable_im();

//
// Event Handling and Widget Management
//

FL_EXPORT extern int handle(int, Fl_Window*);
FL_EXPORT extern int handle_(int, Fl_Window*);

FL_EXPORT inline Fl_Widget* belowmouse()        { return belowmouse_; }
FL_EXPORT extern void belowmouse(Fl_Widget*);

FL_EXPORT inline Fl_Widget* pushed()            { return pushed_; }
FL_EXPORT extern void pushed(Fl_Widget*);

FL_EXPORT inline Fl_Widget* focus()             { return focus_; }
FL_EXPORT extern void focus(Fl_Widget*);

FL_EXPORT extern void add_handler(Fl_Event_Handler ha);
FL_EXPORT extern void add_handler(Fl_Event_Handler ha, Fl_Event_Handler before);
FL_EXPORT extern Fl_Event_Handler last_handler();
FL_EXPORT extern void remove_handler(Fl_Event_Handler h);
FL_EXPORT extern void add_system_handler(Fl_System_Handler h, void *data);
FL_EXPORT extern void remove_system_handler(Fl_System_Handler h);
FL_EXPORT extern void event_dispatch(Fl_Event_Dispatch d);
FL_EXPORT extern Fl_Event_Dispatch event_dispatch();
FL_EXPORT extern Fl_Callback_Reason callback_reason();

//
// Convenience State Check Functions
//

FL_EXPORT inline int event_shift()              { return e_state & FL_SHIFT; }

FL_EXPORT inline int event_ctrl()               { return e_state & FL_CTRL; }
```

```
FL_EXPORT inline int event_command()            { return e_state & FL_COMMAND; }

FL_EXPORT inline int event_alt()                { return e_state & FL_ALT; }

//
// Mouse Button State Functions
//

FL_EXPORT inline int event_buttons()            { return e_state & FL_BUTTONS; }

FL_EXPORT inline int event_button1()            { return e_state & FL_BUTTON1; }

FL_EXPORT inline int event_button2()            { return e_state & FL_BUTTON2; }

FL_EXPORT inline int event_button3()            { return e_state & FL_BUTTON3; }

FL_EXPORT inline int event_button4()            { return e_state & FL_BUTTON4; }

FL_EXPORT inline int event_button5()            { return e_state & FL_BUTTON5; }

} // namespace Fl

#endif // !Fl_core_events_H
```

14.4 function_types.H File Reference

FLTK typedefs for service and widget callbacks.

```
#include <FL/fl_config.h>
#include <FL/Fl_Export.H>
#include <FL/platform_types.h>
#include <FL/Enumerations.H>
```

Typedefs

- typedef void(∗ **Fl_Abort_Handler**) (const char ∗format,...)

  Signature of set_abort functions passed as parameters.
- typedef int(∗ **Fl_Args_Handler**) (int argc, char ∗∗argv, int &i)

  Signature of args functions passed as parameters.
- typedef void(∗ **Fl_Atclose_Handler**) (Fl_Window ∗window, void ∗data)

  Signature of set_atclose functions passed as parameters.
- typedef void(∗ **Fl_Awake_Handler**) (void ∗data)

  Signature of some wakeup callback functions passed as parameters.
- typedef void() **Fl_Box_Draw_F**(int x, int y, int w, int h, Fl_Color color)

  Signature of some box drawing functions passed as parameters.
- typedef void() **Fl_Box_Draw_Focus_F**(Fl_Boxtype bt, int x, int y, int w, int h, Fl_Color fg, Fl_Color bg)

  Signature of box focus frame drawing functions.
- typedef void(∗ **Fl_Clipboard_Notify_Handler**) (int source, void ∗data)

  Signature of add_clipboard_notify functions passed as parameters.
- typedef int(∗ Fl_Event_Dispatch) (int event, Fl_Window ∗w)

  Signature of event_dispatch functions passed as parameters.
- typedef int(∗ **Fl_Event_Handler**) (int event)

  Signature of add_handler functions passed as parameters.
- typedef void(∗ **Fl_FD_Handler**) (FL_SOCKET fd, void ∗data)

  Signature of add_fd functions passed as parameters.
- typedef void(∗ Fl_Idle_Handler) (void ∗data)

  Signature of add_idle callback function passed as parameter.
- typedef void() **Fl_Label_Draw_F**(const Fl_Label ∗label, int x, int y, int w, int h, Fl_Align align)

  Signature of some label drawing functions passed as parameters.
- typedef void() **Fl_Label_Measure_F**(const Fl_Label ∗label, int &width, int &height)

Signature of some label measurement functions passed as parameters.

- typedef void(∗ Fl_Old_Idle_Handler) ()

  Signature of add_idle callback function passed as parameter.

- typedef int(∗ **Fl_System_Handler**) (void ∗event, void ∗data)

  Signature of add_system_handler functions passed as parameters.

- typedef void(∗ Fl_Timeout_Handler) (void ∗data)

  Signature of timeout callback functions passed as parameters.

14.4.1 Detailed Description

FLTK typedefs for service and widget callbacks.

14.5 function_types.H

Go to the documentation of this file.

```
00001 //
00002 // Function types for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 2025 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file.  If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 //     https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 //     https://www.fltk.org/bugs.php
00015 //
00016
00021 #ifndef Fl_core_function_types_H
00022 #define Fl_core_function_types_H
00023
00024
00025 #include <FL/fl_config.h>             // build configuration
00026 #include <FL/Fl_Export.H>             // for FL_EXPORT
00027 #include <FL/platform_types.h>        // for FL_SOCKET
00028 #include <FL/Enumerations.H>          // for Fl_Align, Fl_Color, etc.
00029
00030 class Fl_Window;
00031 struct Fl_Label;
00032
00033
00052 //
00053 // Drawing and Rendering Function Types
00054 //
00055
00057 typedef void (Fl_Label_Draw_F)(const Fl_Label *label, int x, int y, int w, int h, Fl_Align align);
00058
00060 typedef void (Fl_Label_Measure_F)(const Fl_Label *label, int &width, int &height);
00061
00063 typedef void (Fl_Box_Draw_F)(int x, int y, int w, int h, Fl_Color color);
00064
00066 typedef void (Fl_Box_Draw_Focus_F)(Fl_Boxtype bt, int x, int y, int w, int h, Fl_Color fg, Fl_Color
      bg);
00067
00068 //
00069 // Timer and Scheduling Function Types
00070 //
00071
00075 typedef void (*Fl_Timeout_Handler)(void *data);
00076
00078 typedef void (*Fl_Awake_Handler)(void *data);
00079
00080 //
00081 // Idle Processing Function Types
00082 //
00083
00088 typedef void (*Fl_Idle_Handler)(void *data);
00089
00095 typedef void (*Fl_Old_Idle_Handler)();
00096
00097 //
00098 // File Descriptor and I/O Function Types
00099 //
00100
00102 typedef void (*Fl_FD_Handler)(FL_SOCKET fd, void *data);
```

```
//
// Event Handling Function Types
//

typedef int (*Fl_Event_Handler)(int event);

typedef int (*Fl_System_Handler)(void *event, void *data);

typedef int (*Fl_Event_Dispatch)(int event, Fl_Window *w);

//
// Application Lifecycle Function Types
//

typedef void (*Fl_Abort_Handler)(const char *format,...);

typedef void (*Fl_Atclose_Handler)(Fl_Window *window, void *data);

typedef int (*Fl_Args_Handler)(int argc, char **argv, int &i);

//
// Clipboard and Data Transfer Function Types
//

typedef void (*Fl_Clipboard_Notify_Handler)(int source, void *data);

/* group callback_functions */

#endif // !Fl_core_function_types_H
```

14.6 options.H File Reference

Application and system wide options management.

```
#include <FL/fl_config.h>
#include <FL/Fl_Export.H>
```

Namespaces

- namespace Fl

  Fl is the FLTK global namespace containing state information and global methods for the current application.

Enumerations

- enum Fl::Fl_Option { Fl::OPTION_ARROW_FOCUS = 0 , Fl::OPTION_VISIBLE_FOCUS , Fl::OPTION_DND_TEXT , Fl::OPTION_SHOW_TOOLTIPS , Fl::OPTION_FNFC_USES_GTK , Fl::OPTION_FNFC_USES_ZENITY , Fl::OPTION_FNFC_USES_KDIALOG , Fl::OPTION_PRINTER_USES_GTK , Fl::OPTION_SHOW_SCALING , Fl::OPTION_SIMPLE_ZOOM_SHORTCUT , Fl::OPTION_LAST }

  Enumerator for global FLTK options.

Functions

- int Fl::dnd_text_ops ()

  Gets whether drag and drop text operations are supported.
- void Fl::dnd_text_ops (int v)

  Sets whether drag and drop text operations are supported.
- bool Fl::option (Fl_Option opt)

  FLTK library options management.
- void Fl::option (Fl_Option opt, bool val)

  Override an option while the application is running.
- int Fl::visible_focus ()

  Gets or sets the visible keyboard focus on buttons and other non-text widgets.
- void Fl::visible_focus (int v)

Gets or sets the visible keyboard focus on buttons and other non-text widgets.

14.6.1 Detailed Description

Application and system wide options management.

14.7 options.H

Go to the documentation of this file.

```
00001 //
00002 // Core options header file for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 2025 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file.  If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 //     https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 //     https://www.fltk.org/bugs.php
00015 //
00016
00021 #ifndef Fl_core_options_H
00022 #define Fl_core_options_H
00023
00024 #include <FL/fl_config.h>             // build configuration
00025 #include <FL/Fl_Export.H>             // for FL_EXPORT
00026
00027
00028 namespace Fl {
00029
00036 typedef enum {
00037
00047   OPTION_ARROW_FOCUS = 0,
00048
00049
00050   // When switched on, FLTK will use the file chooser dialog that comes
00051   // with your operating system whenever possible. When switched off, FLTK
00052   // will present its own file chooser.
00053   // \todo implement me
00054   // OPTION_NATIVE_FILECHOOSER,
00055
00056   // When Filechooser Preview is enabled, the FLTK or native file chooser
00057   // will show a preview of a selected file (if possible) before the user
00058   // decides to choose the file.
00059   // \todo implement me
00060   // OPTION_FILECHOOSER_PREVIEW,
00061
00066   OPTION_VISIBLE_FOCUS,
00067
00071   OPTION_DND_TEXT,
00072
00076   OPTION_SHOW_TOOLTIPS,
00077
00081   OPTION_FNFC_USES_GTK,
00082
00085   OPTION_FNFC_USES_ZENITY,
00086
00090   OPTION_FNFC_USES_KDIALOG,
00091
00095   OPTION_PRINTER_USES_GTK,
00096
00100   OPTION_SHOW_SCALING,
00101
00107   OPTION_SIMPLE_ZOOM_SHORTCUT,
00108
00109   // don't change this, leave it always as the last element
00111   OPTION_LAST
00112 } Fl_Option;
00113
00114 /*
00115  Return a global setting for all FLTK applications, possibly overridden
00116  by a setting specifically for this application.
00117  */
00118 FL_EXPORT extern bool option(Fl_Option opt);
00119
00120 /*
00121  Override an option while the application is running.
00122  */
```

```
FL_EXPORT extern void option(Fl_Option opt, bool val);

// Visible focus methods...

FL_EXPORT inline void visible_focus(int v) { option(OPTION_VISIBLE_FOCUS, (v!=0)); }

FL_EXPORT inline int  visible_focus() { return option(OPTION_VISIBLE_FOCUS); }

// Drag-n-drop text operation methods...

FL_EXPORT inline void dnd_text_ops(int v) { option(OPTION_DND_TEXT, (v!=0)); }

FL_EXPORT inline int  dnd_text_ops() { return option(OPTION_DND_TEXT); }

} // namespace Fl


#endif // !Fl_core_options_H
```

14.8 pen_events.H File Reference

Pen event handling variables and functions.

```
#include <FL/fl_config.h>
#include <FL/Fl_Export.H>
#include <FL/core/function_types.H>
#include <cstdint>
```

Namespaces

- namespace Fl

  Fl is the FLTK global namespace containing state information and global methods for the current application.
- namespace Fl::Pen

  FLTK Pen/Stylus/Tablet input driver API.

Enumerations

- enum Fl::Pen::Event {
  Fl::Pen::DETECTED = 0x1000 , Fl::Pen::CHANGED , Fl::Pen::IN_RANGE , Fl::Pen::OUT_OF_RANGE , Fl::Pen::ENTER , Fl::Pen::LEAVE , Fl::Pen::TOUCH , Fl::Pen::LIFT , Fl::Pen::HOVER , Fl::Pen::DRAW , Fl::Pen::BUTTON_PUSH , Fl::Pen::BUTTON_RELEASE }

  List of pen events.
- enum class Fl::Pen::State : uint32_t {
  Fl::Pen::State::NONE = 0x0000 , Fl::Pen::State::TIP_HOVERS = 0x0001 , Fl::Pen::State::TIP_DOWN = 0x0002 , Fl::Pen::State::ERASER_HOVERS = 0x0004 , Fl::Pen::State::ERASER_DOWN = 0x0008 , Fl::Pen::State::BUTTON0 = 0x0100 , Fl::Pen::State::BUTTON1 = 0x0200 , Fl::Pen::State::BUTTON2 = 0x0400 , Fl::Pen::State::BUTTON3 = 0x0800 , Fl::Pen::State::ANY_DOWN = BUTTON0 | BUTTON1 | BUTTON2 | BUTTON3 | TIP_DOWN | ERASER_DOWN }

  Bitfield of pen state flags.
- enum class Fl::Pen::Trait : uint32_t {
  Fl::Pen::Trait::NONE = 0x0000 , Fl::Pen::Trait::DRIVER_AVAILABLE = 0x0001 , Fl::Pen::Trait::DETECTED = 0x0002 , Fl::Pen::Trait::DISPLAY = 0x0004 , Fl::Pen::Trait::PEN_ID = 0x0008 , Fl::Pen::Trait::ERASER = 0x0010 , Fl::Pen::Trait::PRESSURE = 0x0020 , Fl::Pen::Trait::BARREL_PRESSURE = 0x0040 , Fl::Pen::Trait::TILT_X = 0x0080 , Fl::Pen::Trait::TILT_Y = 0x0100 , Fl::Pen::Trait::TWIST = 0x0200 , Fl::Pen::Trait::PROXIMITY = 0x0400 }

  Bitfield of traits.

Functions

- Trait Fl::Pen::driver_traits ()

Query the traits supported by the pen/tablet driver.

- bool Fl::Pen::driver_traits (Trait bits)

  Return true if the corresponding bit is set in the driver traits.
- constexpr State Fl::Pen::operator& (State lhs, State rhs)

  Bitwise AND operator for State enum.
- constexpr Trait Fl::Pen::operator& (Trait lhs, Trait rhs)

  Bitwise AND operator for Trait enum.
- constexpr State Fl::Pen::operator| (State lhs, State rhs)

  Bitwise OR operator for State enum.
- constexpr Trait Fl::Pen::operator| (Trait lhs, Trait rhs)

  Bitwise OR operator for Trait enum.
- State & Fl::Pen::operator|= (State &lhs, State rhs)

  Bitwise OR assignment operator for State enum.
- Trait & Fl::Pen::operator|= (Trait &lhs, Trait rhs)

  Bitwise OR assignment operator for Trait enum.
- Trait Fl::Pen::pen_traits (int pen_id=0)

  Query traits of the current pen or stylus.
- bool Fl::Pen::pen_traits (Trait bits, int pen_id=0)

  Return true if the corresponding bit is set in the pen traits.
- void Fl::Pen::release ()

  Clear the "pushed" state and forward pen events as mouse events.
- void Fl::Pen::subscribe (Fl_Widget *widget)

  Receives a Pen::ENTER event when the pen enters this widget.
- void Fl::Pen::unsubscribe (Fl_Widget *widget)

  Stop receiving Pen::ENTER for this widget.

Query values during event handling

- double Fl::Pen::event_barrel_pressure ()

  Returns barrel pressure or tangential pressure.
- int Fl::Pen::event_pen_id ()

  Returns the ID of the pen used in the last event.
- double Fl::Pen::event_pressure ()

  Returns the pressure between the tip or eraser and the surface.
- double Fl::Pen::event_proximity ()

  Returns the proximity of the pen to the surface between 0 and 1.
- State Fl::Pen::event_state ()

  Returns the state of the various buttons and tips.
- bool Fl::Pen::event_state (State bits)

  Return true if any of the bits are set in the event state.
- double Fl::Pen::event_tilt_x ()

  Returns the tilt of the pen in the x and y directions between -1 and 1.
- double **Fl::Pen::event_tilt_y** ()

  Returns pen Y-axis tilt, see event_tilt_x()
- State Fl::Pen::event_trigger ()

  Returns the state change that triggered the event.
- double Fl::Pen::event_twist ()

  Returns the pen's axial rotation in degrees.
- double Fl::Pen::event_x ()

  Returns the pen x and y position inside the handling widget as doubles.
- double Fl::Pen::event_x_root ()

  Returns the pen x and y position in global coordinates as doubles.
- double **Fl::Pen::event_y** ()

  Returns pen Y coordinate in widget space, see event_x().
- double **Fl::Pen::event_y_root** ()

  Returns pen Y coordinate in screen space, see event_x_root().

14.8.1 Detailed Description

Pen event handling variables and functions.

14.9 pen_events.H

Go to the documentation of this file.

```
00001 //
00002 // Pen event header file for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 2025 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file.  If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 //     https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 //     https://www.fltk.org/bugs.php
00015 //
00016
00022 #ifndef FL_CORE_PEN_DRIVER_H
00023 #define FL_CORE_PEN_DRIVER_H
00024
00025 #include <FL/fl_config.h>             // build configuration
00026 #include <FL/Fl_Export.H>             // for FL_EXPORT
00027 #include <FL/core/function_types.H>   // widget callbacks and services
00028
00029 #include <cstdint>
00030
00031 class Fl_Widget;
00032
00033 namespace Fl {
00034
00036 namespace Pen {
00037
00065 enum class Trait : uint32_t {
00067   NONE              = 0x0000,
00069   DRIVER_AVAILABLE  = 0x0001,
00072   DETECTED          = 0x0002,
00074   DISPLAY           = 0x0004,
00076   PEN_ID            = 0x0008,
00078   ERASER            = 0x0010,
00080   PRESSURE          = 0x0020,
00082   BARREL_PRESSURE   = 0x0040,
00084   TILT_X            = 0x0080,
00086   TILT_Y            = 0x0100,
00088   TWIST             = 0x0200,
00090   PROXIMITY         = 0x0400,
00091 };
00092
00099 inline constexpr Trait operator|(Trait lhs, Trait rhs) {
00100   return static_cast<Trait>(static_cast<uint32_t>(lhs) | static_cast<uint32_t>(rhs));
00101 }
00102
00109 inline constexpr Trait operator&(Trait lhs, Trait rhs) {
00110   return static_cast<Trait>(static_cast<uint32_t>(lhs) & static_cast<uint32_t>(rhs));
00111 }
00112
00119 inline Trait& operator|=(Trait& lhs, Trait rhs) {
00120   lhs = lhs | rhs;
00121   return lhs;
00122 }
00123
00124
00129 enum class State : uint32_t {
00131   NONE              = 0x0000,
00133   TIP_HOVERS        = 0x0001,
00135   TIP_DOWN          = 0x0002,
00137   ERASER_HOVERS     = 0x0004,
00139   ERASER_DOWN       = 0x0008,
00141   BUTTON0           = 0x0100,
00143   BUTTON1           = 0x0200,
00145   BUTTON2           = 0x0400,
00147   BUTTON3           = 0x0800,
00149   ANY_DOWN          = BUTTON0 | BUTTON1 | BUTTON2 | BUTTON3 | TIP_DOWN | ERASER_DOWN,
00150 };
00151
00158 inline constexpr State operator|(State lhs, State rhs) {
00159   return static_cast<State>(static_cast<uint32_t>(lhs) | static_cast<uint32_t>(rhs));
00160 }
00161
```

```
inline constexpr State operator&(State lhs, State rhs) {
  return static_cast<State>(static_cast<uint32_t>(lhs) & static_cast<uint32_t>(rhs));
}

inline State& operator|=(State& lhs, State rhs) {
  lhs = lhs | rhs;
  return lhs;
}


enum Event {
  DETECTED = 0x1000,

  CHANGED,

  IN_RANGE,

  OUT_OF_RANGE,

  ENTER,

  LEAVE,

  TOUCH,

  LIFT,

  HOVER,

  DRAW,

  BUTTON_PUSH,

  BUTTON_RELEASE

};

FL_EXPORT extern Trait driver_traits();

inline bool driver_traits(Trait bits) {
  return ((driver_traits() & bits) != Trait::NONE);
}

FL_EXPORT extern Trait pen_traits(int pen_id = 0);

inline bool pen_traits(Trait bits, int pen_id = 0) {
  return ((pen_traits(pen_id) & bits) != Trait::NONE);
}

FL_EXPORT extern void subscribe(Fl_Widget* widget);

FL_EXPORT extern void unsubscribe(Fl_Widget* widget);

FL_EXPORT extern void release();


FL_EXPORT extern double event_x();
FL_EXPORT extern double event_y();

FL_EXPORT extern double event_x_root();
FL_EXPORT extern double event_y_root();

FL_EXPORT extern int event_pen_id();

FL_EXPORT extern double event_pressure();

FL_EXPORT extern double event_barrel_pressure();

FL_EXPORT extern double event_tilt_x();
FL_EXPORT extern double event_tilt_y();

FL_EXPORT extern double event_twist();

FL_EXPORT extern double event_proximity();

FL_EXPORT extern State event_state();

inline bool event_state(State bits) {
  return ((event_state() & bits) != State::NONE);
}

FL_EXPORT extern State event_trigger();

// group fl_pen_events  

} // namespace Pen

```

```
} // namespace Fl

#endif // !FL_CORE_PEN_DRIVER_H
```

14.10 Enumerations.H File Reference

This file contains type definitions and general enumerations.

```
#include <FL/fl_config.h>
#include "Fl_Export.H"
#include "fl_types.h"
#include <FL/platform_types.h>
```

Macros

- #define FL_API_VERSION (FL_MAJOR_VERSION∗10000 + FL_MINOR_VERSION∗100 + FL_PATCH_VERSION)

  The FLTK API version number as an int.
- #define **FL_IMAGE_WITH_ALPHA** 0x40000000
- #define FL_VERSION

  The FLTK version number as a double.

Names of Non-ASCII keys and mouse buttons

The following constants define the names of non-ASCII keys on the keyboard and of mouse buttons for FL_KEYBOARD and FL_SHORTCUT events.

See also

Fl::event_key() and Fl::get_key(int) (use ASCII letters for all other keys):

- #define **FL_Alt_Gr** 0xfe03

  The AltGr key on some international keyboards.
- #define **FL_Alt_L** 0xffe9

  The left alt key.
- #define **FL_Alt_R** 0xffea

  The right alt key.
- #define **FL_Back** 0xEF26

  Like back on a browser.
- #define **FL_BackSpace** 0xff08

  The backspace key.
- #define **FL_Button** 0xfee8

  A mouse button; use `FL_Button + n` for mouse button `n`.
- #define **FL_Caps_Lock** 0xffe5

  The caps lock key.
- #define **FL_Control_L** 0xffe3

  The lefthand control key.
- #define **FL_Control_R** 0xffe4

  The righthand control key.
- #define **FL_Delete** 0xffff

  The delete key.
- #define **FL_Down** 0xff54

  The down arrow key.
- #define **FL_Eisu** 0xff2f

  The Eisu key of JIS keyboards.
- #define **FL_End** 0xff57

  The end key.
- #define **FL_Enter** 0xff0d

  The enter key.
- #define **FL_Escape** 0xff1b

  The escape key.

- #define **FL_F** 0xffbd

  One of the function keys; use FL_F + n for function key n.
- #define **FL_F_Last** 0xffe0

  The last function key; use to range-check function keys.
- #define **FL_Favorites** 0xEF30

  Show favorite locations.
- #define **FL_Forward** 0xEF27

  Like forward on a browser.
- #define **FL_Help** 0xff68

  The 'help' key on Mac keyboards.
- #define **FL_Home** 0xff50

  The home key.
- #define **FL_Home_Page** 0xEF18

  Display user's home page.
- #define **FL_Insert** 0xff63

  The insert key.
- #define **FL_Iso_Key** 0xff0c

  The additional key of ISO keyboards.
- #define **FL_JIS_Underscore** 0xff31

  The underscore key of JIS keyboards.
- #define **FL_Kana** 0xff2e

  The Kana key of JIS keyboards.
- #define **FL_KP** 0xff80

  One of the keypad numbers; use FL_KP + 'n' for digit n.
- #define **FL_KP_Enter** 0xff8d

  The enter key on the keypad, same as Fl_KP+'\r'.
- #define **FL_KP_Last** 0xffbd

  The last keypad key; use to range-check keypad.
- #define **FL_Left** 0xff51

  The left arrow key.
- #define **FL_Mail** 0xEF19

  Invoke user's mail program.
- #define **FL_Media_Next** 0xEF17

  Next track.
- #define **FL_Media_Play** 0xEF14

  Start playing of audio.
- #define **FL_Media_Prev** 0xEF16

  Previous track.
- #define **FL_Media_Stop** 0xEF15

  Stop playing audio.
- #define **FL_Menu** 0xff67

  The menu key.
- #define **FL_Meta_L** 0xffe7

  The left meta/Windows key.
- #define **FL_Meta_R** 0xffe8

  The right meta/Windows key.
- #define **FL_Num_Lock** 0xff7f

  The num lock key.
- #define **FL_Page_Down** 0xff56

  The page-down key.
- #define **FL_Page_Up** 0xff55

  The page-up key.
- #define **FL_Pause** 0xff13

  The pause key.
- #define **FL_Print** 0xff61

  The print (or print-screen) key.
- #define **FL_Refresh** 0xEF29

  Refresh the page.
- #define **FL_Right** 0xff53

The right arrow key.

- #define **FL_Scroll_Lock** 0xff14

  The scroll lock key.
- #define **FL_Search** 0xEF1B

  Search.
- #define **FL_Shift_L** 0xffe1

  The lefthand shift key.
- #define **FL_Shift_R** 0xffe2

  The righthand shift key.
- #define **FL_Sleep** 0xEF2F

  Put system to sleep.
- #define **FL_Stop** 0xEF28

  Stop current operation.
- #define **FL_Tab** 0xff09

  The tab key.
- #define **FL_Up** 0xff52

  The up arrow key.
- #define **FL_Volume_Down** 0xEF11

  Volume control down.
- #define **FL_Volume_Mute** 0xEF12

  Mute sound from the system.
- #define **FL_Volume_Up** 0xEF13

  Volume control up.
- #define **FL_Yen** 0xff30

  The Yen key of JIS keyboards.

Mouse Buttons

These constants define the button numbers for FL_PUSH and FL_RELEASE events.

See also

Fl::event_button()

- #define **FL_BACK_MOUSE** 4

  The back mouse button (side button 1)
- #define **FL_FORWARD_MOUSE** 5

  The forward mouse button (side button 2)
- #define **FL_LEFT_MOUSE** 1

  The left mouse button.
- #define **FL_MIDDLE_MOUSE** 2

  The middle mouse button.
- #define **FL_RIGHT_MOUSE** 3

  The right mouse button.

Typedefs

- typedef int Fl_Fontsize

  Size of a font in pixels.

Enumerations

- enum { FL_READ = 1 , FL_WRITE = 4 , FL_EXCEPT = 8 }

  FD "when" conditions.
- enum Fl_Arrow_Type { FL_ARROW_SINGLE = 0x01 , FL_ARROW_DOUBLE = 0x02 , FL_ARROW_CHOICE = 0x03 , FL_ARROW_RETURN = 0x04 }

  Arrow types define the type of arrow drawing function.
- enum Fl_Damage { FL_DAMAGE_CHILD = 0x01 , FL_DAMAGE_EXPOSE = 0x02 , FL_DAMAGE_SCROLL = 0x04 , FL_DAMAGE_OVERLAY = 0x08 , FL_DAMAGE_USER1 = 0x10 , FL_DAMAGE_USER2 = 0x20 , FL_DAMAGE_ALL = 0x80 }

Damage masks.

- enum Fl_Event {
  FL_NO_EVENT = 0 , FL_PUSH = 1 , FL_RELEASE = 2 , FL_ENTER = 3 ,
  FL_LEAVE = 4 , FL_DRAG = 5 , FL_FOCUS = 6 , FL_UNFOCUS = 7 ,
  FL_KEYDOWN = 8 , FL_KEYBOARD = 8 , FL_KEYUP = 9 , FL_CLOSE = 10 ,
  FL_MOVE = 11 , FL_SHORTCUT = 12 , FL_DEACTIVATE = 13 , FL_ACTIVATE = 14 ,
  FL_HIDE = 15 , FL_SHOW = 16 , FL_PASTE = 17 , FL_SELECTIONCLEAR = 18 ,
  FL_MOUSEWHEEL = 19 , FL_DND_ENTER = 20 , FL_DND_DRAG = 21 , FL_DND_LEAVE = 22 ,
  FL_DND_RELEASE = 23 , FL_SCREEN_CONFIGURATION_CHANGED = 24 , FL_FULLSCREEN = 25 ,
  FL_ZOOM_GESTURE = 26 ,
  FL_ZOOM_EVENT = 27 , FL_BEFORE_TOOLTIP = 28 , FL_BEFORE_MENU = 29 , FL_APP_ACTIVATE = 30 ,
  FL_APP_DEACTIVATE = 31 }

  Every time a user moves the mouse pointer, clicks a button, or presses a key, an event is generated and sent to your application.

- enum Fl_Labeltype {
  FL_NORMAL_LABEL = 0 , FL_NO_LABEL , _FL_SHADOW_LABEL , _FL_ENGRAVED_LABEL ,
  _FL_EMBOSSED_LABEL , _FL_MULTI_LABEL , _FL_ICON_LABEL , _FL_IMAGE_LABEL ,
  FL_FREE_LABELTYPE }

  The labeltype() method sets the type of the label.

- enum Fl_Mode {
  FL_RGB = 0 , **FL_INDEX** = 1 , **FL_SINGLE** = 0 , **FL_DOUBLE** = 2 ,
  FL_ACCUM = 4 , **FL_ALPHA** = 8 , **FL_DEPTH** = 16 , **FL_STENCIL** = 32 ,
  FL_RGB8 = 64 , **FL_MULTISAMPLE** = 128 , **FL_STEREO** = 256 , **FL_FAKE_SINGLE** = 512 ,
  FL_OPENGL3 = 1024 , **FL_DEPTH32** = 2048 }

  visual types and Fl_Gl_Window::mode() (values match Glut)

- enum Fl_Orientation {
  FL_ORIENT_NONE = 0x00 , FL_ORIENT_RIGHT = 0x00 , FL_ORIENT_NE = 0x01 , FL_ORIENT_UP = 0x02 ,
  FL_ORIENT_NW = 0x03 , FL_ORIENT_LEFT = 0x04 , FL_ORIENT_SW = 0x05 , FL_ORIENT_DOWN = 0x06 ,
  FL_ORIENT_SE = 0x07 }

  Fl_Orientation describes the orientation of a GUI element.

When Conditions

- enum Fl_When {
  FL_WHEN_NEVER = 0 , FL_WHEN_CHANGED = 1 , FL_WHEN_NOT_CHANGED = 2 , FL_WHEN_RELEASE = 4 ,
  FL_WHEN_RELEASE_ALWAYS = 6 , FL_WHEN_ENTER_KEY = 8 , FL_WHEN_ENTER_KEY_ALWAYS = 10 , FL_WHEN_ENTER_KEY_CHANGED = 11 ,
  FL_WHEN_CLOSED = 16 }

  These constants determine when a callback is performed.

Callback Reasons

- enum Fl_Callback_Reason {
  FL_REASON_UNKNOWN =0 , FL_REASON_SELECTED , FL_REASON_DESELECTED , FL_REASON_RESELECTED ,
  FL_REASON_OPENED , FL_REASON_CLOSED , FL_REASON_DRAGGED , FL_REASON_CANCELLED ,
  FL_REASON_CHANGED , FL_REASON_GOT_FOCUS , FL_REASON_LOST_FOCUS , FL_REASON_RELEASED ,
  FL_REASON_ENTER_KEY , FL_REASON_USER = 32 }

  These constants describe why a callback is performed.

Cursors

- enum Fl_Cursor {
  FL_CURSOR_DEFAULT = 0 , FL_CURSOR_ARROW = 35 , FL_CURSOR_CROSS = 66 , FL_CURSOR_WAIT = 76 ,
  FL_CURSOR_INSERT = 77 , FL_CURSOR_HAND = 31 , FL_CURSOR_HELP = 47 , FL_CURSOR_MOVE = 27 ,
  FL_CURSOR_NS = 78 , FL_CURSOR_WE = 79 , FL_CURSOR_NWSE = 80 , FL_CURSOR_NESW = 81 ,
  FL_CURSOR_N = 70 , FL_CURSOR_NE = 69 , FL_CURSOR_E = 49 , FL_CURSOR_SE = 8 ,
  FL_CURSOR_S = 9 , FL_CURSOR_SW = 7 , FL_CURSOR_W = 36 , FL_CURSOR_NW = 68 ,
  FL_CURSOR_NONE =255 }

  The following constants define the mouse cursors that are available in FLTK.

Variables

- Fl_Fontsize FL_NORMAL_SIZE

  normal font size

Event States

`The following constants define bits in the Fl::event_state() value.`

- #define **FL_ALT** 0x00080000

  One of the alt keys is down.
- #define **FL_BUTTON**(n) (0x00800000<<(n))

  Mouse button n (n = 1..5) is pushed, undefined if n outside 1..5.
- #define **FL_BUTTON1** 0x01000000

  Mouse button 1 is pushed (L)
- #define **FL_BUTTON2** 0x02000000

  Mouse button 2 is pushed (M)
- #define **FL_BUTTON3** 0x04000000

  Mouse button 3 is pushed (R)
- #define **FL_BUTTON4** 0x08000000

  Mouse button 4 is pushed (BACK)
- #define **FL_BUTTON5** 0x10000000

  Mouse button 5 is pushed (FORWARD)
- #define **FL_BUTTONS** 0x1f000000

  Bitmask: any mouse button (1-5) is pushed.
- #define **FL_CAPS_LOCK** 0x00020000

  The caps lock is on.
- #define **FL_COMMAND** FL_CTRL
- constexpr int **fl_command_modifier** ()
- #define **FL_CONTROL** FL_META
- constexpr int **fl_control_modifier** ()
- #define **FL_CTRL** 0x00040000

  One of the ctrl keys is down.
- #define **FL_KEY_MASK** 0x0000ffff

  All keys are 16 bit for now.
- #define **FL_META** 0x00400000

  One of the meta/Windows keys is down.
- #define **FL_NUM_LOCK** 0x00100000

  The num lock is on.
- #define **FL_SCROLL_LOCK** 0x00800000

  The scroll lock is on.
- #define **FL_SHIFT** 0x00010000

  One of the shift keys is down.

Box Types

FLTK standard box types
This enum defines the standard box types included with FLTK.

Note

For backward compatibility, the Fl_Boxtype enum includes some additional symbols that start with underscores. These underscore-prefixed symbols have identical values to their regular counterparts.

FL_NO_BOX means nothing is drawn at all, so whatever is already on the screen remains. The FL_..._FRAME types only draw their edges, leaving the interior unchanged. The blue color in the image below is the area that is not drawn by the frame types.

Figure 14.1 FLTK Standard Box Types

Note

Not all box types are depicted in the figure above. See enum Fl_Boxtype below for the complete list of box types.

See also

Fl::get_system_colors()

- Fl_Boxtype fl_box (Fl_Boxtype b)

  Get the filled version of a frame.
- enum Fl_Boxtype {
  FL_NO_BOX = 0 , FL_FLAT_BOX , FL_UP_BOX , FL_DOWN_BOX ,
  FL_UP_FRAME , FL_DOWN_FRAME , FL_THIN_UP_BOX , FL_THIN_DOWN_BOX ,
  FL_THIN_UP_FRAME , FL_THIN_DOWN_FRAME , FL_ENGRAVED_BOX , FL_EMBOSSED_BOX ,
  FL_ENGRAVED_FRAME , FL_EMBOSSED_FRAME , FL_BORDER_BOX , FL_SHADOW_BOX ,
  FL_BORDER_FRAME , FL_SHADOW_FRAME , FL_ROUNDED_BOX , FL_RSHADOW_BOX ,
  FL_ROUNDED_FRAME , FL_RFLAT_BOX , FL_ROUND_UP_BOX , FL_ROUND_DOWN_BOX ,

FL_DIAMOND_UP_BOX , FL_DIAMOND_DOWN_BOX , FL_OVAL_BOX , FL_OSHADOW_BOX , FL_OVAL_FRAME , FL_OFLAT_BOX , FL_PLASTIC_UP_BOX , FL_PLASTIC_DOWN_BOX , FL_PLASTIC_UP_FRAME , FL_PLASTIC_DOWN_FRAME , FL_PLASTIC_THIN_UP_BOX , FL_PLASTIC_THIN_DOWN_BOX , FL_PLASTIC_ROUND_UP_BOX , FL_PLASTIC_ROUND_DOWN_BOX , FL_GTK_UP_BOX , FL_GTK_DOWN_BOX , FL_GTK_UP_FRAME , FL_GTK_DOWN_FRAME , FL_GTK_THIN_UP_BOX , FL_GTK_THIN_DOWN_BOX , FL_GTK_THIN_UP_FRAME , FL_GTK_THIN_DOWN_FRAME , FL_GTK_ROUND_UP_BOX , FL_GTK_ROUND_DOWN_BOX , FL_GLEAM_UP_BOX , FL_GLEAM_DOWN_BOX , FL_GLEAM_UP_FRAME , FL_GLEAM_DOWN_FRAME , FL_GLEAM_THIN_UP_BOX , FL_GLEAM_THIN_DOWN_BOX , FL_GLEAM_ROUND_UP_BOX , FL_GLEAM_ROUND_DOWN_BOX , FL_OXY_UP_BOX , FL_OXY_DOWN_BOX , FL_OXY_UP_FRAME , FL_OXY_DOWN_FRAME , FL_OXY_THIN_UP_BOX , FL_OXY_THIN_DOWN_BOX , FL_OXY_THIN_UP_FRAME , FL_OXY_THIN_DOWN_FRAME , FL_OXY_ROUND_UP_BOX , FL_OXY_ROUND_DOWN_BOX , FL_OXY_BUTTON_UP_BOX , FL_OXY_BUTTON_DOWN_BOX , FL_FREE_BOXTYPE , FL_MAX_BOXTYPE = 255 }

FLTK standard box types.

- #define **FL_CIRCLE_BOX** FL_ROUND_DOWN_BOX
- #define **FL_DIAMOND_BOX** FL_DIAMOND_DOWN_BOX
- Fl_Boxtype fl_down (Fl_Boxtype b)

  Get the "pressed" or "down" version of a box.
- Fl_Boxtype fl_frame (Fl_Boxtype b)

  Get the unfilled, frame only version of a box.
- #define **FL_FRAME** FL_ENGRAVED_FRAME
- #define **FL_FRAME_BOX** FL_ENGRAVED_BOX
- Fl_Labeltype fl_define_FL_EMBOSSED_LABEL ()

  Initializes the internal table entry for FL_EMBOSSED_LABEL and returns its internal value.
- Fl_Labeltype fl_define_FL_ENGRAVED_LABEL ()

  Initializes the internal table entry for FL_ENGRAVED_LABEL and returns its internal value.
- Fl_Labeltype fl_define_FL_ICON_LABEL ()

  Initializes the internal table entry for FL_ICON_LABEL and returns its internal value.
- Fl_Labeltype fl_define_FL_IMAGE_LABEL ()

  Initializes the internal table entry for FL_IMAGE_LABEL and returns its internal value.
- Fl_Labeltype fl_define_FL_MULTI_LABEL ()

  Initializes the internal table entry for FL_MULTI_LABEL and returns its internal value.
- Fl_Labeltype fl_define_FL_SHADOW_LABEL ()

  Initializes the internal table entry for FL_SHADOW_LABEL and returns its internal value.
- #define **FL_EMBOSSED_LABEL** fl_define_FL_EMBOSSED_LABEL()

  Draws a label with embossed text.
- #define **FL_ENGRAVED_LABEL** fl_define_FL_ENGRAVED_LABEL()

  Draws a label with engraved text.
- #define **FL_ICON_LABEL** fl_define_FL_ICON_LABEL()

  Draws an icon as the label.
- #define FL_IMAGE_LABEL fl_define_FL_IMAGE_LABEL()

  Draws an image (Fl_Image) as the label.
- #define FL_MULTI_LABEL fl_define_FL_MULTI_LABEL()

  Draws a label that can comprise several parts like text and images.
- #define **FL_SHADOW_LABEL** fl_define_FL_SHADOW_LABEL()

  Draws a label with shadows behind the text.
- #define FL_SYMBOL_LABEL FL_NORMAL_LABEL

  Sets the current label type and returns its corresponding Fl_Labeltype value.

Colors

The Fl_Color type holds an FLTK color value.
Colors are either 8-bit indexes into a `virtual colormap` or 24-bit RGB color values. (See Colors for the default FLTK colormap)
Color indices occupy the lower 8 bits of the value, while RGB colors occupy the upper 24 bits, for a byte organization of RGBI.

```
Fl_Color => 0xrrggbbii
                | | | |
                | | | +--- index between 0 and 255
                | | +----- blue color component (8 bit)
                | +------- green component (8 bit)
                +--------- red component (8 bit)
```

A color can have either an index or an rgb value. Colors with rgb set and an index >0 are reserved for special use.

- const Fl_Color **FL_BACKGROUND2_COLOR** = 7

  the default background color for text, list, and valuator widgets
- const Fl_Color **FL_BACKGROUND_COLOR** = 49

  Default background color.
- const Fl_Color **FL_BLACK** = 56
- const Fl_Color **FL_BLUE** = 216
- typedef unsigned int **Fl_Color**

  An FLTK color value; see also Colors

- Fl_Color fl_color_average (Fl_Color c1, Fl_Color c2, float weight)

  Returns the weighted average color between the two given colors.
- #define **FL_COLOR_CUBE** (Fl_Color)56
- Fl_Color fl_color_cube (int r, int g, int b)

  Returns a color out of the color cube.
- Fl_Color fl_contrast (Fl_Color fg, Fl_Color bg, int context=0, int size=0)

  Returns a color that contrasts with the background color.
- typedef Fl_Color() Fl_Contrast_Function(Fl_Color, Fl_Color, int, int)

  Type of a custom fl_contrast() function.
- void fl_contrast_function (Fl_Contrast_Function *f)

  Register a custom contrast function.
- int fl_contrast_level ()

  Get the contrast level (sensitivity) of the fl_contrast() method.
- void fl_contrast_level (int level)

  Set the contrast level (sensitivity) of the fl_contrast() method.
- enum Fl_Contrast_Mode {
  FL_CONTRAST_NONE = 0 , FL_CONTRAST_LEGACY , FL_CONTRAST_CIELAB , FL_CONTRAST_CUSTOM ,
  FL_CONTRAST_LAST }

  Define the possible modes to calculate fl_contrast().
- int fl_contrast_mode ()

  Return the current contrast algorithm (mode).
- void fl_contrast_mode (int mode)

  Set the contrast algorithm (mode).
- const Fl_Color **FL_CYAN** = 223
- const Fl_Color **FL_DARK1** = 47
- const Fl_Color **FL_DARK2** = 45
- const Fl_Color **FL_DARK3** = 39
- const Fl_Color **FL_DARK_BLUE** = 136

- const Fl_Color **FL_DARK_CYAN** = 140
- const Fl_Color **FL_DARK_GREEN** = 60
- const Fl_Color **FL_DARK_MAGENTA** = 152
- const Fl_Color **FL_DARK_RED** = 72
- const Fl_Color **FL_DARK_YELLOW** = 76
- Fl_Color **fl_darker** (Fl_Color c)

  Returns a darker version of the specified color.
- const Fl_Color **FL_FOREGROUND_COLOR** = 0

  the default foreground color (0) used for labels and text
- #define **FL_FREE_COLOR** (Fl_Color)16

  Colors numbered between FL_FREE_COLOR and FL_FREE_COLOR + FL_NUM_FREE_COLOR - 1 are free for the user to be given any value using Fl::set_color().
- #define **FL_GRAY** FL_BACKGROUND_COLOR
- const Fl_Color **FL_GRAY0** = 32
- #define **FL_GRAY_RAMP** (Fl_Color)32
- Fl_Color fl_gray_ramp (int i)

  Returns a gray color value from black (i == 0) to white (i == FL_NUM_GRAY - 1).
- const Fl_Color **FL_GREEN** = 63
- Fl_Color **fl_inactive** (Fl_Color c)

  Returns the inactive, dimmed version of the given color.
- const Fl_Color **FL_INACTIVE_COLOR** = 8

  the inactive foreground color
- const Fl_Color **FL_LIGHT1** = 50
- const Fl_Color **FL_LIGHT2** = 52
- const Fl_Color **FL_LIGHT3** = 54
- Fl_Color **fl_lighter** (Fl_Color c)

  Returns a lighter version of the specified color.
- double fl_lightness (Fl_Color color)

  Return the perceived lightness of a color.
- double fl_luminance (Fl_Color color)

  Return the raw / physical luminance of a color.
- const Fl_Color **FL_MAGENTA** = 248
- #define **FL_NUM_BLUE** 5
- #define **FL_NUM_FREE_COLOR** 16
- #define **FL_NUM_GRAY** 24
- #define **FL_NUM_GREEN** 8
- #define **FL_NUM_RED** 5
- const Fl_Color **FL_RED** = 88
- Fl_Color **fl_rgb_color** (uchar g)

  Returns the 24-bit color value closest to `g` (grayscale).
- Fl_Color **fl_rgb_color** (uchar r, uchar g, uchar b)

  Returns the 24-bit color value closest to `r`, `g`, `b`.
- const Fl_Color **FL_SELECTION_COLOR** = 15

  the default selection/highlight color
- const Fl_Color **FL_WHITE** = 255
- const Fl_Color **FL_YELLOW** = 95

Alignment Flags

Flags to control the label alignment.
This controls how the label is displayed next to or inside the widget. The default value is FL_ALIGN_CENTER (0) for most widgets, which centers the label inside the widget.
All alignment flags use the common prefix "FL_ALIGN_". In the following descriptions this prefix is sometimes omitted for brevity.
Flags can be or'd to achieve a combination of alignments, but there are some *"magic values"* (e.g. combinations of TOP and BOTTOM and of LEFT and RIGHT) that have special meanings (see below). For instance:
FL_ALIGN_TOP_LEFT == (FL_ALIGN_TOP | FL_ALIGN_LEFT) != FL_ALIGN_LEFT_TOP.

```
Outside alignments (FL_ALIGN_INSIDE is not set):

           TOP_LEFT        TOP       TOP_RIGHT
           +---------------------------------+
   LEFT_TOP|                                 |RIGHT_TOP
           |                                 |
       LEFT|              CENTER             |RIGHT
           |                                 |
LEFT_BOTTOM|                                 |RIGHT_BOTTOM
           +---------------------------------+
           BOTTOM_LEFT   BOTTOM   BOTTOM_RIGHT

Inside alignments (FL_ALIGN_INSIDE is set):

           +---------------------------------+
           |TOP_LEFT        TOP      TOP_RIGHT|
           |                                 |
           |LEFT          CENTER         RIGHT|
           |                                 |
           |BOTTOM_LEFT  BOTTOM  BOTTOM_RIGHT|
           +---------------------------------+
```

See also

Fl_Align, FL_ALIGN_CENTER, etc.

Note

1. Bit positions not defined in the following constants of type `Fl_Align` are reserved for future extensions. Do not use.
2. The *"magic values"* (FL_ALIGN_)LEFT_TOP, RIGHT_TOP, LEFT_BOTTOM, and RIGHT_BOTTOM must not be used together with FL_ALIGN_INSIDE. Use TOP_LEFT, TOP_RIGHT, BOTTOM_LEFT, or BOTTOM_RIGHT instead.
3. Although bits can be or'd together there are some unused/illegal combinations, for instance:
   - setting both FL_ALIGN_TOP and FL_ALIGN_BOTTOM in combinations other than those given in the `Fl_Align` constants below (magic values)
   - setting both FL_ALIGN_LEFT and FL_ALIGN_RIGHT in combinations other than those given in the `Fl_Align` constants below (magic values)
   - using one of the "magic values" (2) together with FL_ALIGN_INSIDE

Using illegal bit combinations or undefined bits may yield unexpected behavior, and this behavior may be changed without notice in future FLTK versions.

- typedef unsigned **Fl_Align**

  FLTK type for alignment control.
- const Fl_Align **FL_ALIGN_BOTTOM** = 0x0002

  Align the label at the bottom of the widget.
- const Fl_Align **FL_ALIGN_BOTTOM_LEFT** = FL_ALIGN_BOTTOM | FL_ALIGN_LEFT
- const Fl_Align **FL_ALIGN_BOTTOM_RIGHT** = FL_ALIGN_BOTTOM | FL_ALIGN_RIGHT
- const Fl_Align **FL_ALIGN_CENTER** = 0x0000

  Align the label horizontally in the middle.
- const Fl_Align **FL_ALIGN_CLIP** = 0x0040

  All parts of the label that are larger than the widget will not be drawn.
- const Fl_Align **FL_ALIGN_IMAGE_BACKDROP** = 0x0200

  If the label contains an image, draw the image or deimage in the background.

- const Fl_Align **FL_ALIGN_IMAGE_MASK** = 0x0320

  Mask value to test for image alignment flags.
- const Fl_Align **FL_ALIGN_IMAGE_NEXT_TO_TEXT** = 0x0100

  If the label contains an image, draw the text to the right of the image.
- const Fl_Align **FL_ALIGN_IMAGE_OVER_TEXT** = 0x0000

  If the label contains an image, draw the text below the image.
- const Fl_Align **FL_ALIGN_INSIDE** = 0x0010

  Draw the label inside of the widget.
- const Fl_Align FL_ALIGN_LEFT = 0x0004

  Align the label at the left of the widget.
- const Fl_Align **FL_ALIGN_LEFT_BOTTOM** = 0x000d

  Outside only, left of widget, bottom position, magic value: TOP | LEFT | RIGHT.
- const Fl_Align **FL_ALIGN_LEFT_TOP** = 0x0007

  Outside only, left of widget, top position, magic value: TOP | BOTTOM | LEFT.
- const Fl_Align **FL_ALIGN_NOWRAP** = 0x0000

  Nothing, same as FL_ALIGN_CENTER, for back compatibility.
- const Fl_Align **FL_ALIGN_POSITION_MASK** = 0x000f

  Mask value to test for TOP, BOTTOM, LEFT, and RIGHT flags.
- const Fl_Align **FL_ALIGN_RIGHT** = 0x0008

  Align the label to the right of the widget.
- const Fl_Align **FL_ALIGN_RIGHT_BOTTOM** = 0x000e

  Outside only, right of widget, bottom position, magic value: BOTTOM | LEFT | RIGHT.
- const Fl_Align **FL_ALIGN_RIGHT_TOP** = 0x000b

  Outside only, right of widget, top position, magic value: TOP | BOTTOM | RIGHT.
- const Fl_Align **FL_ALIGN_TEXT_NEXT_TO_IMAGE** = 0x0120

  If the label contains an image, draw the text to the left of the image.
- const Fl_Align **FL_ALIGN_TEXT_OVER_IMAGE** = 0x0020

  If the label contains an image, draw the text on top of the image.
- const Fl_Align FL_ALIGN_TOP = 0x0001

  Align the label at the top of the widget.
- const Fl_Align **FL_ALIGN_TOP_LEFT** = FL_ALIGN_TOP | FL_ALIGN_LEFT
- const Fl_Align **FL_ALIGN_TOP_RIGHT** = FL_ALIGN_TOP | FL_ALIGN_RIGHT
- const Fl_Align **FL_ALIGN_WRAP** = 0x0080

  Wrap text that does not fit the width of the widget.

Font Numbers

The following constants define the standard FLTK fonts:

- const Fl_Font **FL_BOLD** = 1

  add this to Helvetica, Courier, or Times
- const Fl_Font **FL_BOLD_ITALIC** = 3

  add this to Helvetica, Courier, or Times
- const Fl_Font **FL_COURIER** = 4

  Courier normal.
- const Fl_Font **FL_COURIER_BOLD** = 5

  Courier bold.
- const Fl_Font **FL_COURIER_BOLD_ITALIC** = 7

  Courier bold-italic.
- const Fl_Font **FL_COURIER_ITALIC** = 6

  Courier italic.
- typedef int **Fl_Font**

A font number is an index into the internal font table.

- const Fl_Font **FL_FREE_FONT** = 16

  first one to allocate
- const Fl_Font **FL_HELVETICA** = 0

  Helvetica (or Arial) normal (0)
- const Fl_Font **FL_HELVETICA_BOLD** = 1

  Helvetica (or Arial) bold.
- const Fl_Font **FL_HELVETICA_BOLD_ITALIC** = 3

  Helvetica (or Arial) bold-oblique.
- const Fl_Font **FL_HELVETICA_ITALIC** = 2

  Helvetica (or Arial) oblique.
- const Fl_Font **FL_ITALIC** = 2

  add this to Helvetica, Courier, or Times
- const Fl_Font **FL_SCREEN** = 13

  Default monospaced screen font.
- const Fl_Font **FL_SCREEN_BOLD** = 14

  Default monospaced bold screen font.
- const Fl_Font **FL_SYMBOL** = 12

  Standard symbol font.
- const Fl_Font **FL_TIMES** = 8

  Times roman.
- const Fl_Font **FL_TIMES_BOLD** = 9

  Times roman bold.
- const Fl_Font **FL_TIMES_BOLD_ITALIC** = 11

  Times roman bold-italic.
- const Fl_Font **FL_TIMES_ITALIC** = 10

  Times roman italic.
- const Fl_Font **FL_ZAPF_DINGBATS** = 15

  Zapf-dingbats font.

14.10.1 Detailed Description

This file contains type definitions and general enumerations.

14.10.2 Macro Definition Documentation

14.10.2.1 FL_IMAGE_LABEL

```
#define FL_IMAGE_LABEL fl_define_FL_IMAGE_LABEL()
```

Draws an image (Fl_Image) as the label.

This is useful for one particular part of an Fl_Multi_Label. Use Fl_Widget::image() and/or Fl_Widget::deimage() for normal widgets with images as labels.

14.10.2.2 FL_MULTI_LABEL

```
#define FL_MULTI_LABEL fl_define_FL_MULTI_LABEL()
```

Draws a label that can comprise several parts like text and images.

See also

Fl_Multi_Label

14.10.2.3 FL_SYMBOL_LABEL

```
#define FL_SYMBOL_LABEL FL_NORMAL_LABEL
```

Sets the current label type and returns its corresponding Fl_Labeltype value.
FL_SYMBOL_LABEL is an alias for FL_NORMAL_LABEL.
'@' symbols can be drawn with normal labels as well.
This definition is for historical reasons only (forms compatibility). You should use FL_NORMAL_LABEL instead.

14.10.3 Typedef Documentation

14.10.3.1 Fl_Contrast_Function

```
typedef Fl_Color() Fl_Contrast_Function(Fl_Color, Fl_Color, int, int)
```

Type of a custom fl_contrast() function.
Use this signature to define your own custom fl_contrast() function together with fl_contrast_mode(FL_↩ CONTRAST_CUSTOM). Example:

```
Fl_Color my_contrast(Fl_Color fg, Fl_Color bg, int context, int size) {
  // calculate contrast and ...
  return color;
}
// call this early in your main() program:
fl_contrast_function(my_contrast);
fl_contrast_mode(FL_CONTRAST_CUSTOM);
fl_contrast_level(50); // optional, must be called after fl_contrast_mode()
```

For parameters and types see fl_contrast(Fl_Color, Fl_Color, int, int).

See also

fl_contrast(Fl_Color, Fl_Color, int, int)

fl_contrast_mode(int)

14.10.3.2 Fl_Fontsize

```
typedef int Fl_Fontsize
```

Size of a font in pixels.
This is the approximate height of a font in pixels.

14.10.4 Enumeration Type Documentation

14.10.4.1 anonymous enum

```
anonymous enum
```

FD "when" conditions.

Enumerator

FL_READ	Call the callback when there is data to be read.
FL_WRITE	Call the callback when data can be written without blocking.
FL_EXCEPT	Call the callback if an exception occurs on the file.

14.10.4.2 Fl_Arrow_Type

```
enum Fl_Arrow_Type
```

Arrow types define the type of arrow drawing function.
FLTK schemes can draw several graphical elements in their particular way. One of these elements is an arrow type that can be in different GUI elements like scrollbars, choice buttons, and FLTK's Fl_Return_Button.

Note

This enum is not yet stable (as of FLTK 1.4.0) and may be changed without notice as necessary.

Since

1.4.0

Enumerator

FL_ARROW_SINGLE	Single arrow, e.g. in Fl_Scrollbar.
FL_ARROW_DOUBLE	Double arrow, e.g. in Fl_Counter.
FL_ARROW_CHOICE	Dropdown box, e.g. in Fl_Choice.
FL_ARROW_RETURN	Return arrow, e.g. in Fl_Return_Button.

14.10.4.3 Fl_Boxtype

`enum Fl_Boxtype`

FLTK standard box types.

This enum defines the standard box types included with FLTK.

Enumerator

FL_NO_BOX	nothing is drawn at all, this box is invisible
FL_FLAT_BOX	a flat box
FL_UP_BOX	see figure Standard Box Types
FL_DOWN_BOX	see figure Standard Box Types
FL_UP_FRAME	see figure Standard Box Types
FL_DOWN_FRAME	see figure Standard Box Types
FL_THIN_UP_BOX	see figure Standard Box Types
FL_THIN_DOWN_BOX	see figure Standard Box Types
FL_THIN_UP_FRAME	see figure Standard Box Types
FL_THIN_DOWN_FRAME	see figure Standard Box Types
FL_ENGRAVED_BOX	see figure Standard Box Types
FL_EMBOSSED_BOX	see figure Standard Box Types
FL_ENGRAVED_FRAME	see figure Standard Box Types
FL_EMBOSSED_FRAME	see figure Standard Box Types
FL_BORDER_BOX	see figure Standard Box Types
FL_SHADOW_BOX	see figure Standard Box Types
FL_BORDER_FRAME	see figure Standard Box Types
FL_SHADOW_FRAME	see figure Standard Box Types
FL_ROUNDED_BOX	see figure Standard Box Types
FL_RSHADOW_BOX	see figure Standard Box Types
FL_ROUNDED_FRAME	see figure Standard Box Types
FL_RFLAT_BOX	see figure Standard Box Types
FL_ROUND_UP_BOX	see figure Standard Box Types
FL_ROUND_DOWN_BOX	see figure Standard Box Types
FL_DIAMOND_UP_BOX	see figure Standard Box Types
FL_DIAMOND_DOWN_BOX	see figure Standard Box Types
FL_OVAL_BOX	see figure Standard Box Types
FL_OSHADOW_BOX	see figure Standard Box Types
FL_OVAL_FRAME	see figure Standard Box Types
FL_OFLAT_BOX	see figure Standard Box Types
FL_PLASTIC_UP_BOX	plastic version of FL_UP_BOX

Enumerator

FL_PLASTIC_DOWN_BOX	plastic version of FL_DOWN_BOX
FL_PLASTIC_UP_FRAME	plastic version of FL_UP_FRAME
FL_PLASTIC_DOWN_FRAME	plastic version of FL_DOWN_FRAME
FL_PLASTIC_THIN_UP_BOX	plastic version of FL_THIN_UP_BOX
FL_PLASTIC_THIN_DOWN_BOX	plastic version of FL_THIN_DOWN_BOX
FL_PLASTIC_ROUND_UP_BOX	plastic version of FL_ROUND_UP_BOX
FL_PLASTIC_ROUND_DOWN_BOX	plastic version of FL_ROUND_DOWN_BOX
FL_GTK_UP_BOX	gtk+ version of FL_UP_BOX
FL_GTK_DOWN_BOX	gtk+ version of FL_DOWN_BOX
FL_GTK_UP_FRAME	gtk+ version of FL_UP_FRAME
FL_GTK_DOWN_FRAME	gtk+ version of FL_DOWN_FRAME
FL_GTK_THIN_UP_BOX	gtk+ version of FL_THIN_UP_BOX
FL_GTK_THIN_DOWN_BOX	gtk+ version of FL_THIN_DOWN_BOX
FL_GTK_THIN_UP_FRAME	gtk+ version of FL_THIN_UP_FRAME
FL_GTK_THIN_DOWN_FRAME	gtk+ version of FL_THIN_DOWN_FRAME
FL_GTK_ROUND_UP_BOX	gtk+ version of FL_ROUND_UP_BOX
FL_GTK_ROUND_DOWN_BOX	gtk+ version of FL_ROUND_DOWN_BOX
FL_GLEAM_UP_BOX	gleam version of FL_UP_BOX
FL_GLEAM_DOWN_BOX	gleam version of FL_DOWN_BOX
FL_GLEAM_UP_FRAME	gleam version of FL_UP_FRAME
FL_GLEAM_DOWN_FRAME	gleam version of FL_DOWN_FRAME
FL_GLEAM_THIN_UP_BOX	gleam version of FL_THIN_UP_BOX
FL_GLEAM_THIN_DOWN_BOX	gleam version of FL_THIN_DOWN_BOX
FL_GLEAM_ROUND_UP_BOX	gleam version of FL_ROUND_UP_BOX
FL_GLEAM_ROUND_DOWN_BOX	gleam version of FL_ROUND_DOWN_BOX
FL_OXY_UP_BOX	oxy version of FL_UP_BOX
FL_OXY_DOWN_BOX	oxy version of FL_DOWN_BOX
FL_OXY_UP_FRAME	oxy version of FL_UP_FRAME
FL_OXY_DOWN_FRAME	oxy version of FL_DOWN_FRAME
FL_OXY_THIN_UP_BOX	oxy version of FL_THIN_UP_BOX
FL_OXY_THIN_DOWN_BOX	oxy version of FL_THIN_DOWN_BOX
FL_OXY_THIN_UP_FRAME	oxy version of FL_THIN_UP_FRAME
FL_OXY_THIN_DOWN_FRAME	oxy version of FL_THIN_DOWN_FRAME
FL_OXY_ROUND_UP_BOX	oxy version of FL_ROUND_UP_BOX
FL_OXY_ROUND_DOWN_BOX	oxy version of FL_ROUND_DOWN_BOX
FL_OXY_BUTTON_UP_BOX	FL_OXY_BUTTON_UP_BOX.
FL_OXY_BUTTON_DOWN_BOX	FL_OXY_BUTTON_DOWN_BOX.
FL_FREE_BOXTYPE	the first free box type for creation of new box types
FL_MAX_BOXTYPE	highest legal index for a box type

14.10.4.4 Fl_Callback_Reason

`enum Fl_Callback_Reason`

These constants describe why a callback is performed.

See also

Fl::callback_reason(), Fl_Widget::when(), Fl_When

Enumerator

FL_REASON_UNKNOWN	unknown or unset reason
FL_REASON_SELECTED	an item was selected
FL_REASON_DESELECTED	an item was de-selected
FL_REASON_RESELECTED	an item was re-selected (double-clicked).
FL_REASON_OPENED	an item was opened
FL_REASON_CLOSED	an item was closed
FL_REASON_DRAGGED	an item was dragged into a new place
FL_REASON_CANCELLED	a dialog was cancelled
FL_REASON_CHANGED	the value of the widget was modified
FL_REASON_GOT_FOCUS	a widget received focus
FL_REASON_LOST_FOCUS	a widget lost focus
FL_REASON_RELEASED	the mouse button was released
FL_REASON_ENTER_KEY	user finished input pressing Enter
FL_REASON_USER	user defined callback reasons

14.10.4.5 Fl_Contrast_Mode

`enum Fl_Contrast_Mode`

Define the possible modes to calculate fl_contrast().

Enumerator

FL_CONTRAST_NONE	always return foreground color
FL_CONTRAST_LEGACY	legacy (FLTK 1.3.x) contrast function
FL_CONTRAST_CIELAB	new (FLTK 1.4.0) default function
FL_CONTRAST_CUSTOM	optional custom contrast function
FL_CONTRAST_LAST	internal use only (invalid contrast mode)

14.10.4.6 Fl_Cursor

`enum Fl_Cursor`

The following constants define the mouse cursors that are available in FLTK.
Cursors are provided by the system when available, or bitmaps built into FLTK as a fallback.

Enumerator

FL_CURSOR_DEFAULT	the default cursor, usually an arrow:
FL_CURSOR_ARROW	an arrow pointer:
FL_CURSOR_CROSS	crosshair:
FL_CURSOR_WAIT	busy indicator (for instance hourglass): ,
FL_CURSOR_INSERT	I-beam:
FL_CURSOR_HAND	pointing hand:
FL_CURSOR_HELP	question mark pointer: ?
FL_CURSOR_MOVE	4-pointed arrow or hand: ,
FL_CURSOR_NS	up/down resize:

Enumerator

FL_CURSOR_WE	left/right resize:
FL_CURSOR_NWSE	diagonal resize:
FL_CURSOR_NESW	diagonal resize:
FL_CURSOR_N	upwards resize:
FL_CURSOR_NE	upwards, right resize:
FL_CURSOR_E	rightwards resize:
FL_CURSOR_SE	downwards, right resize:
FL_CURSOR_S	downwards resize:
FL_CURSOR_SW	downwards, left resize:
FL_CURSOR_W	leftwards resize:
FL_CURSOR_NW	upwards, left resize:
FL_CURSOR_NONE	invisible.

14.10.4.7 Fl_Damage

`enum Fl_Damage`

Damage masks.

Enumerator

FL_DAMAGE_CHILD	A child needs to be redrawn.
FL_DAMAGE_EXPOSE	The window was exposed.
FL_DAMAGE_SCROLL	The Fl_Scroll widget was scrolled. Used by other widgets for other widget specific damages.
FL_DAMAGE_OVERLAY	The overlay planes need to be redrawn.
FL_DAMAGE_USER1	First user-defined damage bit.
FL_DAMAGE_USER2	Second user-defined damage bit.
FL_DAMAGE_ALL	Everything needs to be redrawn.

14.10.4.8 Fl_Event

`enum Fl_Event`

Every time a user moves the mouse pointer, clicks a button, or presses a key, an event is generated and sent to your application.

Events can also come from other programs like the window manager.

Events are identified by the integer argument passed to the Fl_Widget::handle() virtual method. Other information about the most recent event is stored in static locations and acquired by calling the Fl::event_∗() methods. This static information remains valid until the next event is read from the window system, so it is ok to look at it outside of the handle() method.

Event numbers can be converted to their actual names using the fl_eventnames[] array (std::map since FLTK 1.5) defined in #include <FL/names.h>

See also

Fl::event_text(), Fl::event_key(), class Fl::

Enumerator

FL_NO_EVENT	No event.

Enumerator

FL_PUSH	A mouse button has gone down with the mouse pointing at this widget. You can find out what button by calling Fl::event_button(). You find out the mouse position by calling Fl::event_x() and Fl::event_y(). A widget indicates that it "wants" the mouse click by returning non-zero from its Fl_Widget::handle() method. It will then become the Fl::pushed() widget and will get FL_DRAG and the matching FL_RELEASE events. If Fl_Widget::handle() returns zero then FLTK will try sending the FL_PUSH to another widget.
FL_RELEASE	A mouse button has been released. You can find out what button by calling Fl::event_button(). In order to receive the FL_RELEASE event, the widget must return non-zero when handling FL_PUSH.
FL_ENTER	The mouse has been moved to point at this widget. This can be used for highlighting feedback. If a widget wants to highlight or otherwise track the mouse, it indicates this by returning non-zero from its handle() method. It then becomes the Fl::belowmouse() widget and will receive FL_MOVE and FL_LEAVE events.
FL_LEAVE	The mouse has moved out of the widget. In order to receive the FL_LEAVE event, the widget must return non-zero when handling FL_ENTER.
FL_DRAG	The mouse has moved with a button held down. The current button state is in Fl::event_state(). The mouse position is in Fl::event_x() and Fl::event_y(). In order to receive FL_DRAG events, the widget must return non-zero when handling FL_PUSH.
FL_FOCUS	This indicates an *attempt* to give a widget the keyboard focus. If a widget wants the focus, it should change itself to display the fact that it has the focus, and return non-zero from its handle() method. It then becomes the Fl::focus() widget and gets FL_KEYDOWN, FL_KEYUP, and FL_UNFOCUS events. The focus will change either because the window manager changed which window gets the focus, or because the user tried to navigate using tab, arrows, or other keys. You can check Fl::event_key() to figure out why it moved. For navigation it will be the key pressed and for interaction with the window manager it will be zero.
FL_UNFOCUS	This event is sent to the previous Fl::focus() widget when another widget gets the focus or the window loses focus.

Enumerator

FL_KEYDOWN	A key was pressed (FL_KEYDOWN) or released (FL_KEYUP). Fl_KEYBOARD is a synonym for FL_KEYDOWN. The key can be found in Fl::event_key(). The text that the key should insert can be found with Fl::event_text() and its length is in Fl::event_length(). If you use the key handle() should return 1. If you return zero then FLTK assumes you ignored the key and will then attempt to send it to a parent widget. If none of them want it, it will change the event into a FL_SHORTCUT event. To receive FL_KEYBOARD events you must also respond to the FL_FOCUS and FL_UNFOCUS events. If you are writing a text-editing widget you may also want to call the Fl::compose() function to translate individual keystrokes into non-ASCII characters. FL_KEYUP events are sent to the widget that currently has focus. This is not necessarily the same widget that received the corresponding FL_KEYDOWN event because focus may have changed between events.
FL_KEYBOARD	Equivalent to FL_KEYDOWN. **See also** FL_KEYDOWN
FL_KEYUP	Key release event. **See also** FL_KEYDOWN
FL_CLOSE	The user clicked the close button of a window. This event is used internally only to trigger the callback of Fl_Window derived classed. The default callback closes the window calling Fl_Window::hide().
FL_MOVE	The mouse has moved without any mouse buttons held down. This event is sent to the Fl::belowmouse() widget. In order to receive FL_MOVE events, the widget must return non-zero when handling FL_ENTER.
FL_SHORTCUT	If the Fl::focus() widget is zero or ignores an FL_KEYBOARD event then FLTK tries sending this event to every widget it can, until one of them returns non-zero. FL_SHORTCUT is first sent to the Fl::belowmouse() widget, then its parents and siblings, and eventually to every widget in the window, trying to find an object that returns non-zero. FLTK tries really hard to not to ignore any keystrokes! You can also make "global" shortcuts by using Fl::add_handler(). A global shortcut will work no matter what windows are displayed or which one has the focus.
FL_DEACTIVATE	This widget is no longer active, due to Fl_Widget::deactivate() being called on it or one of its parents. Fl_Widget::active() may still be true after this, the widget is only active if Fl_Widget::active() is true on it and all its parents (use Fl_Widget::active_r() to check this).
FL_ACTIVATE	This widget is now active, due to Fl_Widget::activate() being called on it or one of its parents.

Enumerator

FL_HIDE	This widget is no longer visible, due to Fl_Widget::hide() being called on it or one of its parents, or due to a parent window being minimized (old Wayland versions may not send the FL_HIDE event to minimized toplevel windows). Fl_Widget::visible() may still be true after this, but the widget is visible only if visible() is true for it and all its parents (use Fl_Widget::visible_r() to check this).
FL_SHOW	This widget is visible again, due to Fl_Widget::show() being called on it or one of its parents, or due to a parent window being restored. Child Fl_Windows respond to this by actually creating the window if not done already, so if you subclass a window, be sure to pass FL_SHOW to the base class Fl_Widget::handle() method!
FL_PASTE	You should get this event some time after you call Fl::paste(). The contents of Fl::event_text() is the text to insert and the number of characters is in Fl::event_length().
FL_SELECTIONCLEAR	The Fl::selection_owner() will get this event before the selection is moved to another widget. This indicates that some other widget or program has claimed the selection. Motif programs used this to clear the selection indication. Most modern programs ignore this.
FL_MOUSEWHEEL	The user has moved the mouse wheel. The Fl::event_dx() and Fl::event_dy() methods can be used to find the amount to scroll horizontally and vertically.
FL_DND_ENTER	The mouse has been moved to point at this widget. A widget that is interested in receiving drag'n'drop data must return 1 to receive FL_DND_DRAG, FL_DND_LEAVE and FL_DND_RELEASE events.
FL_DND_DRAG	The mouse has been moved inside a widget while dragging data. A widget that is interested in receiving drag'n'drop data should indicate the possible drop position.
FL_DND_LEAVE	The mouse has moved out of the widget.
FL_DND_RELEASE	The user has released the mouse button dropping data into the widget. If the widget returns 1, it will receive the data in the immediately following FL_PASTE event.
FL_SCREEN_CONFIGURATION_CHANGED	The screen configuration (number, positions) was changed. Use Fl::add_handler() to be notified of this event.
FL_FULLSCREEN	The fullscreen state of the window has changed. This event is sent to the window's handle method.
FL_ZOOM_GESTURE	The user has made a zoom/pinch/magnification gesture (Mac OS platform only). The Fl::event_dy() method can be used to find magnification amount, Fl::event_x() and Fl::event_y() are set as well. This event is sent to the window's handle method.
FL_ZOOM_EVENT	A zoom event (ctrl/+/-/0/ or cmd/+/-/0/) was processed. Use Fl::add_handler() to be notified of this event.
FL_BEFORE_TOOLTIP	A tooltip is about to pop up for this widget. The mouse coordinates are available in Fl::event_x() and Fl::event_y(). Change the widget tooltip as needed.
FL_BEFORE_MENU	Triggered just before a menu is displayed. Widgets derived from Fl_Menu_ receive this event right before the menu appears, providing an opportunity to update menu item states and activation.

Enumerator

FL_APP_ACTIVATE	Sent when the application is activated and becomes topmost and focused. X11 requires support of EWMH to generate this event. **See also** Fl::add_handler()
FL_APP_DEACTIVATE	Sent when the application is sent to the background and deactivated. X11 requires support of EWMH to generate this event. **See also** Fl::add_handler()

14.10.4.9 Fl_Labeltype

`enum Fl_Labeltype`

The labeltype() method sets the type of the label.

Note

The documented `enum Fl_Labeltype` contains some values (names) with leading underscores, e.g. ↩ **`_FL_IMAGE_LABEL`**. This is due to technical reasons - please use the same values (names) without the leading underscore in your code! Enum values with leading underscores are reserved for internal use and subject to change without notice!

The following standard label types are included:

Enumerator

FL_NORMAL_LABEL	draws the text (0)
FL_NO_LABEL	does nothing
_FL_SHADOW_LABEL	draws a drop shadow under the text
_FL_ENGRAVED_LABEL	draws edges as though the text is engraved
_FL_EMBOSSED_LABEL	draws edges as though the text is raised
_FL_MULTI_LABEL	draws a composite label **See also** Fl_Multi_Label
_FL_ICON_LABEL	draws the icon associated with the text
_FL_IMAGE_LABEL	the label displays an "icon" based on a Fl_Image
FL_FREE_LABELTYPE	first free labeltype to use for creating own labeltypes

14.10.4.10 Fl_Orientation

`enum Fl_Orientation`

Fl_Orientation describes the orientation of a GUI element.
FLTK schemes can draw several graphical elements, for instance arrows, pointing at different directions. This enum defines the direction to use for drawing a particular GUI element.
The definition of this enum was chosen such that the enum value can be multiplied by 45 to get a rotation angle in degrees starting at the horizontal axis (0 = right, 1 = NE, 2 = up, ...) that can be used with fl_rotate(). Note: angle is counter-clockwise in degrees.

The 'unspecified' value **FL_ORIENT_NONE** shall be used for elements that would usually not be rotated, like the return arrow of the Fl_Return_Button. It can still be used as an angle though since it is the same value as `FL_ORIENT_RIGHT` (0 degrees).

Note

This enum is not yet stable (as of FLTK 1.4.0) and may be changed without notice as necessary.

Since

1.4.0

Enumerator

FL_ORIENT_NONE	GUI element direction is unspecified.
FL_ORIENT_RIGHT	GUI element pointing right (0°)
FL_ORIENT_NE	GUI element pointing NE (45°)
FL_ORIENT_UP	GUI element pointing up (90°)
FL_ORIENT_NW	GUI element pointing NW (135°)
FL_ORIENT_LEFT	GUI element pointing left (180°)
FL_ORIENT_SW	GUI element pointing SW (225°)
FL_ORIENT_DOWN	GUI element pointing down (270°)
FL_ORIENT_SE	GUI element pointing SE (315°)

14.10.4.11 Fl_When

`enum Fl_When`

These constants determine when a callback is performed.

Fl_When is a bit field. Some values are merely shortcuts for common bit combinations. New flags may be added in the future, so it's important to mask the required bit when reading via `when()`.

Note

Some widgets may not fully support `FL_WHEN_`... flags.

See also

Fl_Widget::when(), Fl::callback_reason(), Fl_Callback_Reason, Fl_Widget::do_callback()

Enumerator

FL_WHEN_NEVER	Never call the callback.
FL_WHEN_CHANGED	Do the callback only when the widget value changes.
FL_WHEN_NOT_CHANGED	Do the callback whenever the user interacts with the widget.
FL_WHEN_RELEASE	Do the callback when the button or key is released and the value changes.
FL_WHEN_RELEASE_ALWAYS	Do the callback when the button or key is released, even if the value doesn't change.
FL_WHEN_ENTER_KEY	Do the callback when the user presses the ENTER key and the value changes.
FL_WHEN_ENTER_KEY_ALWAYS	Do the callback when the user presses the ENTER key, even if the value doesn't change.
FL_WHEN_ENTER_KEY_CHANGED	Do callbacks whether the value changed or not, and when the ENTER key is pressed.
FL_WHEN_CLOSED	Do the callback when a child of Fl_Tabs is closed.

14.10.5 Function Documentation

14.10.5.1 fl_box()

```
Fl_Boxtype fl_box (
            Fl_Boxtype b ) [inline]
```

Get the filled version of a frame.
If no filled version of a given frame exists, the behavior of this function is undefined and some random box or frame is returned.

14.10.5.2 fl_color_cube()

```
Fl_Color fl_color_cube (
            int r,
            int g,
            int b ) [inline]
```

Returns a color out of the color cube.
`r` must be in the range 0 to FL_NUM_RED (5) minus 1, `g` must be in the range 0 to FL_NUM_GREEN (8) minus 1, `b` must be in the range 0 to FL_NUM_BLUE (5) minus 1.
To get the closest color to a 8-bit set of R,G,B values use:

```
fl_color_cube(R * (FL_NUM_RED - 1) / 255,
   G * (FL_NUM_GREEN - 1) / 255,
   B * (FL_NUM_BLUE - 1) / 255);
```

14.10.5.3 fl_define_FL_EMBOSSED_LABEL()

```
Fl_Labeltype fl_define_FL_EMBOSSED_LABEL ( ) [extern]
```

Initializes the internal table entry for FL_EMBOSSED_LABEL and returns its internal value.
Internal use only.

14.10.5.4 fl_define_FL_ENGRAVED_LABEL()

```
Fl_Labeltype fl_define_FL_ENGRAVED_LABEL ( ) [extern]
```

Initializes the internal table entry for FL_ENGRAVED_LABEL and returns its internal value.
Internal use only.

14.10.5.5 fl_define_FL_ICON_LABEL()

```
Fl_Labeltype fl_define_FL_ICON_LABEL ( ) [extern]
```

Initializes the internal table entry for FL_ICON_LABEL and returns its internal value.
Internal use only.

14.10.5.6 fl_define_FL_IMAGE_LABEL()

```
Fl_Labeltype fl_define_FL_IMAGE_LABEL ( ) [extern]
```

Initializes the internal table entry for FL_IMAGE_LABEL and returns its internal value.
Internal use only.

14.10.5.7 fl_define_FL_MULTI_LABEL()

```
Fl_Labeltype fl_define_FL_MULTI_LABEL ( ) [extern]
```

Initializes the internal table entry for FL_MULTI_LABEL and returns its internal value.
Internal use only.

14.10.5.8 fl_define_FL_SHADOW_LABEL()

```
Fl_Labeltype fl_define_FL_SHADOW_LABEL ( ) [extern]
```

Initializes the internal table entry for FL_SHADOW_LABEL and returns its internal value.
Internal use only.

14.10.5.9 fl_down()

```
Fl_Boxtype fl_down (
            Fl_Boxtype b )  [inline]
```

Get the "pressed" or "down" version of a box.
If no "down" version of a given box exists, the behavior of this function is undefined and some random box or frame is returned.

14.10.5.10 fl_frame()

```
Fl_Boxtype fl_frame (
            Fl_Boxtype b )  [inline]
```

Get the unfilled, frame only version of a box.
If no frame version of a given box exists, the behavior of this function is undefined and some random box or frame is returned.

14.10.5.11 fl_gray_ramp()

```
Fl_Color fl_gray_ramp (
            int i )  [inline]
```

Returns a gray color value from black (i == 0) to white (i == FL_NUM_GRAY - 1).
FL_NUM_GRAY is defined to be 24 in the current FLTK release. To get the closest FLTK gray value to an 8-bit grayscale color 'I' use:

```
fl_gray_ramp(I * (FL_NUM_GRAY - 1) / 255)
```

14.10.6 Variable Documentation

14.10.6.1 FL_ALIGN_LEFT

```
const Fl_Align FL_ALIGN_LEFT = 0x0004
```

Align the label at the left of the widget.
Inside labels appear left-justified starting at the left side of the widget, outside labels are right-justified and drawn to the left of the widget.

14.10.6.2 FL_ALIGN_TOP

```
const Fl_Align FL_ALIGN_TOP = 0x0001
```

Align the label at the top of the widget.
Inside labels appear below the top, outside labels are drawn on top of the widget.

14.10.6.3 FL_NORMAL_SIZE

```
Fl_Fontsize FL_NORMAL_SIZE  [extern]
```

normal font size
normal font size

14.11 Enumerations.H

Go to the documentation of this file.

```
00001 //
00002 // Enumerations for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 1998-2026 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file.  If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 //     https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 //     https://www.fltk.org/bugs.php
00015 //
00016
```

```
#ifndef Fl_Enumerations_H
#define Fl_Enumerations_H

/*
 ******************************************************************************
 * FL_ABI_VERSION is defined by CMake since FLTK 1.3.4.
 * It is written to FL/fl_config.h and #included here.
 * Since FLTK 1.5.0 all version checks of FL_ABI_VERSION are done by CMake
 * in CMake/gen_config.cmake.
 ******************************************************************************
 * For more informations on FL_ABI_VERSION see README.abi-version.txt.
 ******************************************************************************
 */

#include <FL/fl_config.h>

#include "Fl_Export.H"
#include "fl_types.h"
#include <FL/platform_types.h> // for FL_COMMAND and FL_CONTROL

// Doxygen group 'version_numbers' is defined in file FL/fl_config.h
// which is generated by CMake from fl_config.h.in and included above.

#define FL_VERSION      ( (double)FL_MAJOR_VERSION + \
                          (double)FL_MINOR_VERSION * 0.01 + \
                          (double)FL_PATCH_VERSION * 0.0001 )

#define FL_API_VERSION (FL_MAJOR_VERSION*10000 + FL_MINOR_VERSION*100 + FL_PATCH_VERSION)


// end of group 'version_numbers'  00095
// DEV NOTE: Keep this list in sync with FL/names.h
enum Fl_Event { // events
  FL_NO_EVENT           = 0,

  FL_PUSH               = 1,

  FL_RELEASE            = 2,

  FL_ENTER              = 3,

  FL_LEAVE              = 4,

  FL_DRAG               = 5,

  FL_FOCUS              = 6,

  FL_UNFOCUS            = 7,

  FL_KEYDOWN            = 8,

  FL_KEYBOARD           = 8,

  FL_KEYUP              = 9,

  FL_CLOSE              = 10,

  FL_MOVE               = 11,

  FL_SHORTCUT           = 12,

  FL_DEACTIVATE         = 13,

  FL_ACTIVATE           = 14,

  FL_HIDE               = 15,

  FL_SHOW               = 16,

  FL_PASTE              = 17,

  FL_SELECTIONCLEAR     = 18,

  FL_MOUSEWHEEL         = 19,

  FL_DND_ENTER          = 20,

  FL_DND_DRAG           = 21,

  FL_DND_LEAVE          = 22,

  FL_DND_RELEASE        = 23,
  FL_SCREEN_CONFIGURATION_CHANGED = 24,
  FL_FULLSCREEN         = 25,
  FL_ZOOM_GESTURE       = 26,
  FL_ZOOM_EVENT         = 27,
  FL_BEFORE_TOOLTIP     = 28,
```

```
  FL_BEFORE_MENU        = 29,
  FL_APP_ACTIVATE       = 30,
  FL_APP_DEACTIVATE     = 31,
  // DEV NOTE: Keep this list in sync with FL/names.h
};

enum Fl_When { // Fl_Widget::when():
  FL_WHEN_NEVER             =  0,
  FL_WHEN_CHANGED           =  1,
  FL_WHEN_NOT_CHANGED       =  2,
  FL_WHEN_RELEASE           =  4,
  FL_WHEN_RELEASE_ALWAYS    =  6,
  FL_WHEN_ENTER_KEY         =  8,
  FL_WHEN_ENTER_KEY_ALWAYS  = 10,
  FL_WHEN_ENTER_KEY_CHANGED = 11,
  FL_WHEN_CLOSED            = 16
};
// group:  When Conditions  00376

enum Fl_Callback_Reason {
  FL_REASON_UNKNOWN=0,
  FL_REASON_SELECTED,
  FL_REASON_DESELECTED,
  FL_REASON_RESELECTED,
  FL_REASON_OPENED,
  FL_REASON_CLOSED,
  FL_REASON_DRAGGED,
  FL_REASON_CANCELLED,
  FL_REASON_CHANGED,
  FL_REASON_GOT_FOCUS,
  FL_REASON_LOST_FOCUS,
  FL_REASON_RELEASED,
  FL_REASON_ENTER_KEY,
  FL_REASON_USER = 32
};
// group:  Callback Reasons  00401

// FIXME: These codes collide with valid Unicode keys

#define FL_Button       0xfee8
#define FL_BackSpace    0xff08
#define FL_Tab          0xff09
#define FL_Iso_Key      0xff0c
#define FL_Enter        0xff0d
#define FL_Pause        0xff13
#define FL_Scroll_Lock  0xff14
#define FL_Escape       0xff1b
#define FL_Kana         0xff2e
#define FL_Eisu         0xff2f
#define FL_Yen          0xff30
#define FL_JIS_Underscore 0xff31
#define FL_Home         0xff50
#define FL_Left         0xff51
#define FL_Up           0xff52
#define FL_Right        0xff53
#define FL_Down         0xff54
#define FL_Page_Up      0xff55
#define FL_Page_Down    0xff56
#define FL_End          0xff57
#define FL_Print        0xff61
#define FL_Insert       0xff63
#define FL_Menu         0xff67
#define FL_Help         0xff68
#define FL_Num_Lock     0xff7f
#define FL_KP           0xff80
#define FL_KP_Enter     0xff8d
#define FL_KP_Last      0xffbd
#define FL_F            0xffbd
#define FL_F_Last       0xffe0
#define FL_Shift_L      0xffe1
#define FL_Shift_R      0xffe2
#define FL_Control_L    0xffe3
#define FL_Control_R    0xffe4
#define FL_Caps_Lock    0xffe5
#define FL_Meta_L       0xffe7
#define FL_Meta_R       0xffe8
#define FL_Alt_L        0xffe9
#define FL_Alt_R        0xffea
#define FL_Delete       0xffff
#define FL_Alt_Gr       0xfe03

// These use the Private Use Area (PUA) of the Basic Multilingual Plane
// of Unicode. Guaranteed not to conflict with a proper Unicode character.

// These primarily map to the XFree86 keysym range
#define FL_Volume_Down  0xEF11
#define FL_Volume_Mute  0xEF12
```

```
#define FL_Volume_Up    0xEF13
#define FL_Media_Play   0xEF14
#define FL_Media_Stop   0xEF15
#define FL_Media_Prev   0xEF16
#define FL_Media_Next   0xEF17
#define FL_Home_Page    0xEF18
#define FL_Mail         0xEF19
#define FL_Search       0xEF1B
#define FL_Back         0xEF26
#define FL_Forward      0xEF27
#define FL_Stop         0xEF28
#define FL_Refresh      0xEF29
#define FL_Sleep        0xEF2F
#define FL_Favorites    0xEF30

// group:  Non-ASCII key names  00479
#define FL_LEFT_MOUSE     1
#define FL_MIDDLE_MOUSE   2
#define FL_RIGHT_MOUSE    3
#define FL_BACK_MOUSE     4
#define FL_FORWARD_MOUSE  5

// group:  Mouse Buttons  00496

// group:  Event States  00504
// FIXME: it would be nice to have the modifiers in the upper 8 bits so that
//        a unicode key (21 bits) can be sent as an unsigned with the modifiers.

#define FL_SHIFT        0x00010000
#define FL_CAPS_LOCK    0x00020000
#define FL_CTRL         0x00040000
#define FL_ALT          0x00080000
#define FL_NUM_LOCK     0x00100000
                                        // most X servers do this?
#define FL_META         0x00400000
                                        // correct for XFree86
#define FL_SCROLL_LOCK  0x00800000
                                        // correct for XFree86

// To allow compile time initialisation of menus, this must not be moved into the System Driver.
#if defined(__APPLE__) &&!defined(FLTK_USE_X11)
#define FL_COMMAND FL_META
#define FL_CONTROL FL_CTRL
#else
#define FL_COMMAND FL_CTRL
#define FL_CONTROL FL_META
#endif
constexpr FL_EXPORT int fl_command_modifier() { return FL_COMMAND; }
constexpr FL_EXPORT int fl_control_modifier() { return FL_CONTROL; }

// Mouse buttons

#define FL_BUTTON1      0x01000000
#define FL_BUTTON2      0x02000000
#define FL_BUTTON3      0x04000000
#define FL_BUTTON4      0x08000000
#define FL_BUTTON5      0x10000000
#define FL_BUTTONS      0x1f000000

#define FL_BUTTON(n)    (0x00800000«(n))

#define FL_KEY_MASK     0x0000ffff
                                        //   FIXME: Unicode needs 21 bits!

// group:  Event States  00546
// group:  Box Types  00571
enum Fl_Boxtype { // boxtypes (if you change these you must also change fl_boxtype.cxx):

  FL_NO_BOX = 0,
  FL_FLAT_BOX,
  FL_UP_BOX,
  FL_DOWN_BOX,
  FL_UP_FRAME,
  FL_DOWN_FRAME,
  FL_THIN_UP_BOX,
  FL_THIN_DOWN_BOX,
  FL_THIN_UP_FRAME,
  FL_THIN_DOWN_FRAME,
  FL_ENGRAVED_BOX,
  FL_EMBOSSED_BOX,
  FL_ENGRAVED_FRAME,
  FL_EMBOSSED_FRAME,
  FL_BORDER_BOX,
  FL_SHADOW_BOX,
  FL_BORDER_FRAME,
  FL_SHADOW_FRAME,
  FL_ROUNDED_BOX,
```

```
  FL_RSHADOW_BOX,
  FL_ROUNDED_FRAME,
  FL_RFLAT_BOX,
  FL_ROUND_UP_BOX,
  FL_ROUND_DOWN_BOX,
  FL_DIAMOND_UP_BOX,
  FL_DIAMOND_DOWN_BOX,
  FL_OVAL_BOX,
  FL_OSHADOW_BOX,
  FL_OVAL_FRAME,
  FL_OFLAT_BOX,
  FL_PLASTIC_UP_BOX,
  FL_PLASTIC_DOWN_BOX,
  FL_PLASTIC_UP_FRAME,
  FL_PLASTIC_DOWN_FRAME,
  FL_PLASTIC_THIN_UP_BOX,
  FL_PLASTIC_THIN_DOWN_BOX,
  FL_PLASTIC_ROUND_UP_BOX,
  FL_PLASTIC_ROUND_DOWN_BOX,
  FL_GTK_UP_BOX,
  FL_GTK_DOWN_BOX,
  FL_GTK_UP_FRAME,
  FL_GTK_DOWN_FRAME,
  FL_GTK_THIN_UP_BOX,
  FL_GTK_THIN_DOWN_BOX,
  FL_GTK_THIN_UP_FRAME,
  FL_GTK_THIN_DOWN_FRAME,
  FL_GTK_ROUND_UP_BOX,
  FL_GTK_ROUND_DOWN_BOX,
  FL_GLEAM_UP_BOX,
  FL_GLEAM_DOWN_BOX,
  FL_GLEAM_UP_FRAME,
  FL_GLEAM_DOWN_FRAME,
  FL_GLEAM_THIN_UP_BOX,
  FL_GLEAM_THIN_DOWN_BOX,
  FL_GLEAM_ROUND_UP_BOX,
  FL_GLEAM_ROUND_DOWN_BOX,
  FL_OXY_UP_BOX,
  FL_OXY_DOWN_BOX,
  FL_OXY_UP_FRAME,
  FL_OXY_DOWN_FRAME,
  FL_OXY_THIN_UP_BOX,
  FL_OXY_THIN_DOWN_BOX,
  FL_OXY_THIN_UP_FRAME,
  FL_OXY_THIN_DOWN_FRAME,
  FL_OXY_ROUND_UP_BOX,
  FL_OXY_ROUND_DOWN_BOX,
  FL_OXY_BUTTON_UP_BOX,
  FL_OXY_BUTTON_DOWN_BOX,
  FL_FREE_BOXTYPE,
  FL_MAX_BOXTYPE = 255,

  // To reduce size at link time, FLTK used a trick to remove rarely used box
  // types from the main library. FLTK no offers Schems at runtime, so all
  // box types are now included in the main library. The following symbols
  // are provided for backward compatibility, but they are not documented and
  // should not be used in new code.
#ifndef FL_DOXYGEN
  _FL_SHADOW_BOX = FL_SHADOW_BOX,
  _FL_SHADOW_FRAME = FL_SHADOW_FRAME,
  _FL_ROUNDED_BOX = FL_ROUNDED_BOX,
  _FL_RSHADOW_BOX = FL_RSHADOW_BOX,
  _FL_ROUNDED_FRAME = FL_ROUNDED_FRAME,
  _FL_RFLAT_BOX = FL_RFLAT_BOX,
  _FL_ROUND_UP_BOX = FL_ROUND_UP_BOX,
  _FL_ROUND_DOWN_BOX = FL_ROUND_DOWN_BOX,
  _FL_DIAMOND_UP_BOX = FL_DIAMOND_UP_BOX,
  _FL_DIAMOND_DOWN_BOX = FL_DIAMOND_DOWN_BOX,
  _FL_OVAL_BOX = FL_OVAL_BOX,
  _FL_OSHADOW_BOX = FL_OSHADOW_BOX,
  _FL_OVAL_FRAME = FL_OVAL_FRAME,
  _FL_OFLAT_BOX = FL_OFLAT_BOX,
  _FL_PLASTIC_UP_BOX = FL_PLASTIC_UP_BOX,
  _FL_PLASTIC_DOWN_BOX = FL_PLASTIC_DOWN_BOX,
  _FL_PLASTIC_UP_FRAME = FL_PLASTIC_UP_FRAME,
  _FL_PLASTIC_DOWN_FRAME = FL_PLASTIC_DOWN_FRAME,
  _FL_PLASTIC_THIN_UP_BOX = FL_PLASTIC_THIN_UP_BOX,
  _FL_PLASTIC_THIN_DOWN_BOX = FL_PLASTIC_THIN_DOWN_BOX,
  _FL_PLASTIC_ROUND_UP_BOX = FL_PLASTIC_ROUND_UP_BOX,
  _FL_PLASTIC_ROUND_DOWN_BOX = FL_PLASTIC_ROUND_DOWN_BOX,
  _FL_GTK_UP_BOX = FL_GTK_UP_BOX,
  _FL_GTK_DOWN_BOX = FL_GTK_DOWN_BOX,
  _FL_GTK_UP_FRAME = FL_GTK_UP_FRAME,
  _FL_GTK_DOWN_FRAME = FL_GTK_DOWN_FRAME,
  _FL_GTK_THIN_UP_BOX = FL_GTK_THIN_UP_BOX,
  _FL_GTK_THIN_DOWN_BOX = FL_GTK_THIN_DOWN_BOX,
  _FL_GTK_THIN_UP_FRAME = FL_GTK_THIN_UP_FRAME,
```

```
  _FL_GTK_THIN_DOWN_FRAME = FL_GTK_THIN_DOWN_FRAME,
  _FL_GTK_ROUND_UP_BOX = FL_GTK_ROUND_UP_BOX,
  _FL_GTK_ROUND_DOWN_BOX = FL_GTK_ROUND_DOWN_BOX,
  _FL_GLEAM_UP_BOX = FL_GLEAM_UP_BOX,
  _FL_GLEAM_DOWN_BOX = FL_GLEAM_DOWN_BOX,
  _FL_GLEAM_UP_FRAME = FL_GLEAM_UP_FRAME,
  _FL_GLEAM_DOWN_FRAME = FL_GLEAM_DOWN_FRAME,
  _FL_GLEAM_THIN_UP_BOX = FL_GLEAM_THIN_UP_BOX,
  _FL_GLEAM_THIN_DOWN_BOX = FL_GLEAM_THIN_DOWN_BOX,
  _FL_GLEAM_ROUND_UP_BOX = FL_GLEAM_ROUND_UP_BOX,
  _FL_GLEAM_ROUND_DOWN_BOX = FL_GLEAM_ROUND_DOWN_BOX,
  _FL_OXY_UP_BOX = FL_OXY_UP_BOX,
  _FL_OXY_DOWN_BOX = FL_OXY_DOWN_BOX,
  _FL_OXY_UP_FRAME = FL_OXY_UP_FRAME,
  _FL_OXY_DOWN_FRAME = FL_OXY_DOWN_FRAME,
  _FL_OXY_THIN_UP_BOX = FL_OXY_THIN_UP_BOX,
  _FL_OXY_THIN_DOWN_BOX = FL_OXY_THIN_DOWN_BOX,
  _FL_OXY_THIN_UP_FRAME = FL_OXY_THIN_UP_FRAME,
  _FL_OXY_THIN_DOWN_FRAME = FL_OXY_THIN_DOWN_FRAME,
  _FL_OXY_ROUND_UP_BOX = FL_OXY_ROUND_UP_BOX,
  _FL_OXY_ROUND_DOWN_BOX = FL_OXY_ROUND_DOWN_BOX,
  _FL_OXY_BUTTON_UP_BOX = FL_OXY_BUTTON_UP_BOX,
  _FL_OXY_BUTTON_DOWN_BOX = FL_OXY_BUTTON_DOWN_BOX,
#endif // ! FL_DOXYGEN

};

// conversions of box types to other boxtypes:
inline Fl_Boxtype fl_box(Fl_Boxtype b) {
  return (Fl_Boxtype)((b<FL_UP_BOX||b%4>1)?b:(b-2));
}
inline Fl_Boxtype fl_down(Fl_Boxtype b) {
  return (Fl_Boxtype)((b<FL_UP_BOX)?b:(b|1));
}
inline Fl_Boxtype fl_frame(Fl_Boxtype b) {
  return (Fl_Boxtype)((b%4<2)?b:(b+2));
}

// back-compatibility box types:
#define FL_FRAME FL_ENGRAVED_FRAME
#define FL_FRAME_BOX FL_ENGRAVED_BOX
#define FL_CIRCLE_BOX FL_ROUND_DOWN_BOX
#define FL_DIAMOND_BOX FL_DIAMOND_DOWN_BOX

// group:  Box Types  00745
enum Fl_Labeltype {    // labeltypes:
  FL_NORMAL_LABEL = 0,
  FL_NO_LABEL,
  _FL_SHADOW_LABEL,
  _FL_ENGRAVED_LABEL,
  _FL_EMBOSSED_LABEL,
  _FL_MULTI_LABEL,
  _FL_ICON_LABEL,
  _FL_IMAGE_LABEL,

  FL_FREE_LABELTYPE
};

#define FL_SYMBOL_LABEL FL_NORMAL_LABEL
extern Fl_Labeltype FL_EXPORT fl_define_FL_SHADOW_LABEL();
#define FL_SHADOW_LABEL fl_define_FL_SHADOW_LABEL()

extern Fl_Labeltype FL_EXPORT fl_define_FL_ENGRAVED_LABEL();
#define FL_ENGRAVED_LABEL fl_define_FL_ENGRAVED_LABEL()

extern Fl_Labeltype FL_EXPORT fl_define_FL_EMBOSSED_LABEL();
#define FL_EMBOSSED_LABEL fl_define_FL_EMBOSSED_LABEL()

extern Fl_Labeltype FL_EXPORT fl_define_FL_MULTI_LABEL();
#define FL_MULTI_LABEL fl_define_FL_MULTI_LABEL()

extern Fl_Labeltype FL_EXPORT fl_define_FL_ICON_LABEL();
#define FL_ICON_LABEL fl_define_FL_ICON_LABEL()

extern Fl_Labeltype FL_EXPORT fl_define_FL_IMAGE_LABEL();
#define FL_IMAGE_LABEL fl_define_FL_IMAGE_LABEL()

typedef unsigned Fl_Align;

const Fl_Align FL_ALIGN_CENTER          = 0x0000;

const Fl_Align FL_ALIGN_TOP             = 0x0001;

const Fl_Align FL_ALIGN_BOTTOM          = 0x0002;

const Fl_Align FL_ALIGN_LEFT            = 0x0004;

```

```
const Fl_Align FL_ALIGN_RIGHT           = 0x0008;

const Fl_Align FL_ALIGN_INSIDE          = 0x0010;

const Fl_Align FL_ALIGN_TEXT_OVER_IMAGE = 0x0020;

const Fl_Align FL_ALIGN_IMAGE_OVER_TEXT = 0x0000;

const Fl_Align FL_ALIGN_CLIP            = 0x0040;

const Fl_Align FL_ALIGN_WRAP            = 0x0080;

const Fl_Align FL_ALIGN_IMAGE_NEXT_TO_TEXT = 0x0100;

const Fl_Align FL_ALIGN_TEXT_NEXT_TO_IMAGE = 0x0120;

const Fl_Align FL_ALIGN_IMAGE_BACKDROP  = 0x0200;

const Fl_Align FL_ALIGN_TOP_LEFT        = FL_ALIGN_TOP | FL_ALIGN_LEFT;
const Fl_Align FL_ALIGN_TOP_RIGHT       = FL_ALIGN_TOP | FL_ALIGN_RIGHT;
const Fl_Align FL_ALIGN_BOTTOM_LEFT     = FL_ALIGN_BOTTOM | FL_ALIGN_LEFT;
const Fl_Align FL_ALIGN_BOTTOM_RIGHT    = FL_ALIGN_BOTTOM | FL_ALIGN_RIGHT;

const Fl_Align FL_ALIGN_LEFT_TOP        = 0x0007;

const Fl_Align FL_ALIGN_RIGHT_TOP       = 0x000b;

const Fl_Align FL_ALIGN_LEFT_BOTTOM     = 0x000d;

const Fl_Align FL_ALIGN_RIGHT_BOTTOM    = 0x000e;

const Fl_Align FL_ALIGN_NOWRAP          = 0x0000;

const Fl_Align FL_ALIGN_POSITION_MASK   = 0x000f;

const Fl_Align FL_ALIGN_IMAGE_MASK      = 0x0320;
typedef int Fl_Font;

const Fl_Font FL_HELVETICA              = 0;
const Fl_Font FL_HELVETICA_BOLD         = 1;
const Fl_Font FL_HELVETICA_ITALIC       = 2;
const Fl_Font FL_HELVETICA_BOLD_ITALIC  = 3;
const Fl_Font FL_COURIER                = 4;
const Fl_Font FL_COURIER_BOLD           = 5;
const Fl_Font FL_COURIER_ITALIC         = 6;
const Fl_Font FL_COURIER_BOLD_ITALIC    = 7;
const Fl_Font FL_TIMES                  = 8;
const Fl_Font FL_TIMES_BOLD             = 9;
const Fl_Font FL_TIMES_ITALIC           = 10;
const Fl_Font FL_TIMES_BOLD_ITALIC      = 11;
const Fl_Font FL_SYMBOL                 = 12;
const Fl_Font FL_SCREEN                 = 13;
const Fl_Font FL_SCREEN_BOLD            = 14;
const Fl_Font FL_ZAPF_DINGBATS          = 15;

const Fl_Font FL_FREE_FONT              = 16;
const Fl_Font FL_BOLD                   = 1;
const Fl_Font FL_ITALIC                 = 2;
const Fl_Font FL_BOLD_ITALIC            = 3;

typedef int Fl_Fontsize;

extern FL_EXPORT Fl_Fontsize FL_NORMAL_SIZE;

typedef unsigned int Fl_Color;

// Standard colors. These are used as default colors in widgets and altered as necessary
const Fl_Color FL_FOREGROUND_COLOR  = 0;
const Fl_Color FL_BACKGROUND2_COLOR = 7;
const Fl_Color FL_INACTIVE_COLOR    = 8;
const Fl_Color FL_SELECTION_COLOR   = 15;

  // boxtypes generally limit themselves to these colors so
  // the whole ramp is not allocated:

const Fl_Color FL_GRAY0   = 32;                 // 'A'
const Fl_Color FL_DARK3   = 39;                 // 'H'
const Fl_Color FL_DARK2   = 45;                 // 'N'
const Fl_Color FL_DARK1   = 47;                 // 'P'
const Fl_Color FL_BACKGROUND_COLOR  = 49;       // 'R' default background color
const Fl_Color FL_LIGHT1  = 50;                 // 'S'
const Fl_Color FL_LIGHT2  = 52;                 // 'U'
const Fl_Color FL_LIGHT3  = 54;                 // 'W'

  // FLTK provides a 5x8x5 color cube that is used with colormap visuals

const Fl_Color FL_BLACK   = 56;
```

```
const Fl_Color FL_RED     = 88;
const Fl_Color FL_GREEN   = 63;
const Fl_Color FL_YELLOW  = 95;
const Fl_Color FL_BLUE    = 216;
const Fl_Color FL_MAGENTA = 248;
const Fl_Color FL_CYAN    = 223;
const Fl_Color FL_DARK_RED = 72;

const Fl_Color FL_DARK_GREEN    = 60;
const Fl_Color FL_DARK_YELLOW   = 76;
const Fl_Color FL_DARK_BLUE     = 136;
const Fl_Color FL_DARK_MAGENTA  = 152;
const Fl_Color FL_DARK_CYAN     = 140;

const Fl_Color FL_WHITE         = 255;

#define FL_FREE_COLOR     (Fl_Color)16
#define FL_NUM_FREE_COLOR 16
#define FL_GRAY_RAMP      (Fl_Color)32
#define FL_NUM_GRAY       24
#define FL_GRAY           FL_BACKGROUND_COLOR
#define FL_COLOR_CUBE     (Fl_Color)56
#define FL_NUM_RED        5
#define FL_NUM_GREEN      8
#define FL_NUM_BLUE       5

FL_EXPORT Fl_Color fl_inactive(Fl_Color c);

typedef Fl_Color (Fl_Contrast_Function)(Fl_Color, Fl_Color, int, int);

FL_EXPORT void fl_contrast_function(Fl_Contrast_Function *f);

enum Fl_Contrast_Mode {
  FL_CONTRAST_NONE = 0,
  FL_CONTRAST_LEGACY,
  FL_CONTRAST_CIELAB,
  FL_CONTRAST_CUSTOM,
  FL_CONTRAST_LAST
};

// The following functions are defined and documented in src/fl_contrast.cxx

FL_EXPORT void fl_contrast_level(int level);
FL_EXPORT int fl_contrast_level();
FL_EXPORT void fl_contrast_mode(int mode);
FL_EXPORT int fl_contrast_mode();

FL_EXPORT Fl_Color fl_contrast(Fl_Color fg, Fl_Color bg, int context = 0, int size = 0);

FL_EXPORT double fl_lightness(Fl_Color color);
FL_EXPORT double fl_luminance(Fl_Color color);

// Other color functions are defined and documented in src/fl_color.cxx

FL_EXPORT Fl_Color fl_color_average(Fl_Color c1, Fl_Color c2, float weight);

inline Fl_Color fl_lighter(Fl_Color c) { return fl_color_average(c, FL_WHITE, .67f); }

inline Fl_Color fl_darker(Fl_Color c) { return fl_color_average(c, FL_BLACK, .67f); }

inline Fl_Color fl_rgb_color(uchar r, uchar g, uchar b) {
  if (!r && !g && !b) return FL_BLACK;
  else return (Fl_Color)(((((r « 8) | g) « 8) | b) « 8);
}

inline Fl_Color fl_rgb_color(uchar g) {
  if (!g) return FL_BLACK;
  else return (Fl_Color)(((((g « 8) | g) « 8) | g) « 8);
}

inline Fl_Color fl_gray_ramp(int i) {return (Fl_Color)(i+FL_GRAY_RAMP);}

inline Fl_Color fl_color_cube(int r, int g, int b) {
  return (Fl_Color)((b*FL_NUM_RED + r) * FL_NUM_GREEN + g + FL_COLOR_CUBE);}

// group:  Colors  01195
/* FIXME: We should renumber these, but that will break the ABI */
enum Fl_Cursor {
  FL_CURSOR_DEFAULT    =  0,  // U+2196
  FL_CURSOR_ARROW      = 35,  // U+2196
  FL_CURSOR_CROSS      = 66,  // U+FF0B
  FL_CURSOR_WAIT       = 76,  // U+231A, U+231B
  FL_CURSOR_INSERT     = 77,  // U+2336
  FL_CURSOR_HAND       = 31,  // U+261C
  FL_CURSOR_HELP       = 47,
  FL_CURSOR_MOVE       = 27,  // U+2725, U+270B

```

```
  /* Resize indicators */
  FL_CURSOR_NS         = 78,  // U+21D5
  FL_CURSOR_WE         = 79,  // U+21D4
  FL_CURSOR_NWSE       = 80,  // U+2921
  FL_CURSOR_NESW       = 81,  // U+2922
  FL_CURSOR_N          = 70,  // U+2912
  FL_CURSOR_NE         = 69,  // U+2197
  FL_CURSOR_E          = 49,  // U+21E5
  FL_CURSOR_SE         =  8,  // U+21F2
  FL_CURSOR_S          =  9,  // U+2913
  FL_CURSOR_SW         =  7,  // U+2199
  FL_CURSOR_W          = 36,  // U+21E4
  FL_CURSOR_NW         = 68,  // U+21F1

  FL_CURSOR_NONE       =255
};
// group:  Cursors  01232
enum { // values for "when" passed to Fl::add_fd()
  FL_READ   = 1,
  FL_WRITE  = 4,
  FL_EXCEPT = 8
};

enum Fl_Mode {
  FL_RGB        = 0,
  FL_INDEX      = 1,
  FL_SINGLE     = 0,
  FL_DOUBLE     = 2,
  FL_ACCUM      = 4,
  FL_ALPHA      = 8,
  FL_DEPTH      = 16,
  FL_STENCIL    = 32,
  FL_RGB8       = 64,
  FL_MULTISAMPLE= 128,
  FL_STEREO     = 256,
  FL_FAKE_SINGLE = 512, // Fake single buffered windows using double-buffer
  FL_OPENGL3    = 1024,
  FL_DEPTH32    = 2048,
};

// image alpha blending

#define FL_IMAGE_WITH_ALPHA 0x40000000

enum Fl_Damage {
  FL_DAMAGE_CHILD    = 0x01,
  FL_DAMAGE_EXPOSE   = 0x02,
  FL_DAMAGE_SCROLL   = 0x04,
  FL_DAMAGE_OVERLAY  = 0x08,
  FL_DAMAGE_USER1    = 0x10,
  FL_DAMAGE_USER2    = 0x20,
  FL_DAMAGE_ALL      = 0x80
};

// FLTK 1.0.x compatibility definitions (FLTK_1_0_COMPAT) dropped in 1.4.0

enum Fl_Arrow_Type {
  FL_ARROW_SINGLE = 0x01,
  FL_ARROW_DOUBLE = 0x02,
  FL_ARROW_CHOICE = 0x03,
  FL_ARROW_RETURN = 0x04
};

enum Fl_Orientation {
  FL_ORIENT_NONE  = 0x00,
  FL_ORIENT_RIGHT = 0x00,
  FL_ORIENT_NE    = 0x01,
  FL_ORIENT_UP    = 0x02,
  FL_ORIENT_NW    = 0x03,
  FL_ORIENT_LEFT  = 0x04,
  FL_ORIENT_SW    = 0x05,
  FL_ORIENT_DOWN  = 0x06,
  FL_ORIENT_SE    = 0x07
};

#endif
```

14.12 filename.H File Reference

File names and URI utility functions.

```
#include "Fl_Export.H"
#include <FL/platform_types.h>
#include <string>
```

Macros

- #define **FL_PATH_MAX** 2048

  all path buffers should use this length

Typedefs

- typedef int() Fl_File_Sort_F(struct dirent **, struct dirent **)

  File sorting function.

Functions

- void fl_decode_uri (char *uri)

  Decodes a URL-encoded string.
- int fl_filename_absolute (char *to, int tolen, const char *from)

  Makes a filename absolute from a relative filename to the current working directory.
- int fl_filename_absolute (char *to, int tolen, const char *from, const char *cwd)

  Concatenate the absolute path `base` with `from` to form the new absolute path in `to`.
- std::string fl_filename_absolute_str (const std::string &from)

  Makes a filename absolute from a filename relative to the current working directory.
- std::string fl_filename_absolute_str (const std::string &from, const std::string &base)

  Append the relative filename `from` to the absolute filename `base` to form the new absolute path.
- int fl_filename_expand (char *to, int tolen, const char *from)

  Expands a filename containing shell variables and tilde (~).
- std::string fl_filename_expand_str (const std::string &from)

  Expands a filename containing shell variables and tilde (~).
- const char * fl_filename_ext (const char *buf)

  Gets the extension of a filename.
- std::string fl_filename_ext_str (const std::string &filename)

  Return a new string that contains the filename extension.
- void fl_filename_free_list (struct dirent ***l, int n)

  Free the list of filenames that is generated by fl_filename_list().
- int fl_filename_isdir (const char *name)

  Determines if a file exists and is a directory from its filename.
- int fl_filename_list (const char *d, struct dirent ***l, Fl_File_Sort_F *s=fl_numericsort)

  Portable and const-correct wrapper for the scandir() function.
- int fl_filename_match (const char *name, const char *pattern)

  Checks if a string `s` matches a pattern `p`.
- const char * fl_filename_name (const char *filename)

  Gets the file name from a path.
- std::string fl_filename_name_str (const std::string &filename)

  Return a new string that contains the name part of the filename.
- std::string fl_filename_path_str (const std::string &filename)

  Return a new string that contains the path part of the filename.
- int fl_filename_relative (char *to, int tolen, const char *from)

  Makes a filename relative to the current working directory.
- int fl_filename_relative (char *to, int tolen, const char *from, const char *cwd)

  Makes a filename relative to any other directory.
- std::string fl_filename_relative_str (const std::string &from)

  Makes a filename relative to the current working directory.

- std::string fl_filename_relative_str (const std::string &from, const std::string &base)

  Makes a filename relative to any directory.
- char * fl_filename_setext (char *to, int tolen, const char *ext)

  Replaces the extension in `buf` *of max.*
- std::string fl_filename_setext_str (const std::string &filename, const std::string &new_extension)

  Return a copy of the old filename with the new extension.
- std::string fl_getcwd_str ()

  Cross-platform function to get the current working directory as a UTF-8 encoded value in an std::string.
- int fl_open_uri (const char *uri, char *msg, int msglen)

  Opens the specified Uniform Resource Identifier (URI).

14.12.1 Detailed Description

File names and URI utility functions.

14.13 filename.H

Go to the documentation of this file.

```
00001 /*
00002  * Filename header file for the Fast Light Tool Kit (FLTK).
00003  *
00004  * Copyright 1998-2023 by Bill Spitzak and others.
00005  *
00006  * This library is free software. Distribution and use rights are outlined in
00007  * the file "COPYING" which should have been included with this file.  If this
00008  * file is missing or damaged, see the license at:
00009  *
00010  *     https://www.fltk.org/COPYING.php
00011  *
00012  * Please see the following page on how to report bugs and issues:
00013  *
00014  *     https://www.fltk.org/bugs.php
00015  */
00016
00017 /*
00018  * Note to devs:
00019  * Under Windows, we include filename.H from numericsort.c; this should probably change.
00020  * This implies that we need C-style comments and '#ifdef __cplusplus ... #endif'
00021  */
00022
00027 #ifndef FL_FILENAME_H
00028 #  define FL_FILENAME_H
00029
00030 #include "Fl_Export.H"
00031 #include <FL/platform_types.h>
00032
00033 #ifdef __cplusplus
00034
00035 // The following include is not (yet) used in FLTK 1.4
00036 // In FLTK 1.5 or 4.0 using std::string would be default.
00037 // #include <string>
00038
00039 #endif /* __cplusplus */
00040
00045 #  define FL_PATH_MAX 2048
00061 FL_EXPORT const char *fl_filename_name(const char * filename);
00062 FL_EXPORT const char *fl_filename_ext(const char *buf);
00063 FL_EXPORT char *fl_filename_setext(char *to, int tolen, const char *ext);
00064 FL_EXPORT int fl_filename_expand(char *to, int tolen, const char *from);
00065 FL_EXPORT int fl_filename_absolute(char *to, int tolen, const char *from);
00066 FL_EXPORT int fl_filename_relative(char *to, int tolen, const char *from);
00067 FL_EXPORT int fl_filename_match(const char *name, const char *pattern);
00068 FL_EXPORT int fl_filename_isdir(const char *name);
00069
00070 #  if defined(__cplusplus)
00071
00072 FL_EXPORT int fl_filename_absolute(char *to, int tolen, const char *from, const char *cwd);
00073 FL_EXPORT int fl_filename_relative(char *to, int tolen, const char *from, const char *cwd);
00074
00075 #include <string>
00076
00077 FL_EXPORT std::string fl_filename_name_str(const std::string &filename);
00078 FL_EXPORT std::string fl_filename_path_str(const std::string &filename);
00079 FL_EXPORT std::string fl_filename_ext_str(const std::string &filename);
00080 FL_EXPORT std::string fl_filename_setext_str(const std::string &filename, const std::string
      &new_extension);
```

```
FL_EXPORT std::string fl_filename_expand_str(const std::string &from);
FL_EXPORT std::string fl_filename_absolute_str(const std::string &from);
FL_EXPORT std::string fl_filename_absolute_str(const std::string &from, const std::string &base);
FL_EXPORT std::string fl_filename_relative_str(const std::string &from);
FL_EXPORT std::string fl_filename_relative_str(const std::string &from, const std::string &base);
FL_EXPORT std::string fl_getcwd_str();

#  endif /* defined(__cplusplus) */

#  if defined(__cplusplus) && !defined(FL_DOXYGEN)
/*
 * Under Windows, we include filename.H from numericsort.c; this should probably change...
 */

inline char *fl_filename_setext(char *to, const char *ext) { return fl_filename_setext(to, FL_PATH_MAX, ext); }
inline int fl_filename_expand(char *to, const char *from) { return fl_filename_expand(to, FL_PATH_MAX, from); }
inline int fl_filename_absolute(char *to, const char *from) { return fl_filename_absolute(to, FL_PATH_MAX, from); }
inline int fl_filename_relative(char *to, const char *from) { return fl_filename_relative(to, FL_PATH_MAX, from); }
#  endif /* __cplusplus */

#  if defined (__cplusplus)
extern "C" {
#  endif /* __cplusplus */

#  if !defined(FL_DOXYGEN)
FL_EXPORT int fl_alphasort(struct dirent **, struct dirent **);
FL_EXPORT int fl_casealphasort(struct dirent **, struct dirent **);
FL_EXPORT int fl_casenumericsort(struct dirent **, struct dirent **);
FL_EXPORT int fl_numericsort(struct dirent **, struct dirent **);
#  endif

typedef int (Fl_File_Sort_F)(struct dirent **, struct dirent **);
#  if defined(__cplusplus)
}

/*
 * Portable "scandir" function.  Ugly but necessary...
 */

FL_EXPORT int fl_filename_list(const char *d, struct dirent ***l,
                               Fl_File_Sort_F *s = fl_numericsort);
FL_EXPORT void fl_filename_free_list(struct dirent ***l, int n);

/*
 * Generic function to open a Uniform Resource Identifier (URI) using a
 * system-defined program (added in FLTK 1.1.8)
 */

FL_EXPORT int   fl_open_uri(const char *uri, char *msg = (char *)0,
                            int msglen = 0);

FL_EXPORT void fl_decode_uri(char *uri);

#  endif /* __cplusplus */

/*
 * Note: FLTK 1.0.x compatibility definitions (FLTK_1_0_COMPAT) dropped in 1.4.0
 */

#endif /* FL_FILENAME_H */
```

14.14 Fl.H File Reference

Fl namespace.

```
#include <FL/fl_config.h>
#include <FL/Fl_Export.H>
#include <FL/platform_types.h>
#include <FL/fl_casts.H>
#include <FL/core/function_types.H>
#include <FL/core/events.H>
#include <FL/core/options.H>
#include <FL/core/pen_events.H>
#include <FL/Fl_Widget_Tracker.H>
#include <FL/Fl_Cairo.H>
```

```
#include "fl_utf8.h"
#include "Enumerations.H"
#include <string.h>
#include <vector>
```

Classes

- struct Fl::TimeoutData

Namespaces

- namespace Fl

  Fl is the FLTK global namespace containing state information and global methods for the current application.

Macros

- #define **Fl_Object** Fl_Widget

  for back compatibility - use Fl_Widget!

Functions

- int Fl::abi_check (const int val=99999)

  Returns whether the runtime library ABI version is correct.
- int Fl::abi_version ()

  Returns the compiled-in value of the FL_ABI_VERSION constant.
- void Fl::add_check (Fl_Timeout_Handler, void *=0)

  FLTK will call this callback just before it flushes the display and waits for events.
- void Fl::add_clipboard_notify (Fl_Clipboard_Notify_Handler h, void *data=0)

  FLTK will call the registered callback whenever there is a change to the selection buffer or the clipboard.
- void Fl::add_fd (int fd, Fl_FD_Handler cb, void *=0)

  Adds file descriptor fd to listen to.
- void Fl::add_fd (int fd, int when, Fl_FD_Handler cb, void *=0)

  Adds file descriptor fd to listen to.
- void Fl::add_idle (Fl_Idle_Handler cb, void *data=0)

  Adds a callback function that is called every time by Fl::wait() and also makes it act as though the timeout is zero (this makes Fl::wait() return immediately, so if it is in a loop it is called repeatedly, and thus the idle function is called repeatedly).
- void **Fl::add_idle** (Fl_Old_Idle_Handler cb)
- void Fl::add_timeout (double t, Fl_Timeout_Handler cb, void *data=0)

  Adds a one-shot timeout callback.
- int Fl::api_version ()

  Returns the compiled-in value of the FL_API_VERSION constant.
- int Fl::arg (int argc, char **argv, int &i)

  Parse a single switch from `argv`, starting at word `i`.
- void Fl::args (int argc, char **argv)

  Parse all command line switches matching standard FLTK options only.
- int Fl::args (int argc, char **argv, int &i, Fl_Args_Handler cb=0)

  Parse command line switches using the `cb` argument handler.
- int Fl::args_to_utf8 (int argc, char **&argv)

  Convert Windows commandline arguments to UTF-8.
- void Fl::awake ()

  Notifies the main GUI thread from a worker thread.
- int Fl::awake (Fl_Awake_Handler handler, void *user_data=nullptr)

Schedules a callback to be executed by the main thread, then wakes up the main thread.

- void Fl::awake (void *message)

  Awake the main GUI thread and leave a message pointer.
- int Fl::awake_once (Fl_Awake_Handler handler, void *user_data=nullptr)

  Schedules a callback to be executed once by the main thread, then wakes up the main thread.
- void Fl::background (uchar, uchar, uchar)

  Changes fl_color(FL_BACKGROUND_COLOR) to the given color, and changes the gray ramp from 32 to 56 to black to white.
- void Fl::background2 (uchar, uchar, uchar)

  Changes the alternative background color.
- bool Fl::box_bg (Fl_Boxtype)

  Returns whether the given boxtype draws its background.
- int Fl::box_border_radius_max ()

  Get the maximum border radius of all "rounded" boxtypes in pixels.
- void Fl::box_border_radius_max (int R)

  Set the maximum border radius of all "rounded" boxtypes in pixels.
- Fl_Color Fl::box_color (Fl_Color)

  Gets the drawing color to be used for the background of a box.
- int Fl::box_dh (Fl_Boxtype)

  Returns the height offset for the given boxtype.
- int Fl::box_dw (Fl_Boxtype)

  Returns the width offset for the given boxtype.
- int Fl::box_dx (Fl_Boxtype)

  Returns the X offset for the given boxtype.
- int Fl::box_dy (Fl_Boxtype)

  Returns the Y offset for the given boxtype.
- int Fl::box_shadow_width ()

  Get the box shadow width of all "shadow" boxtypes in pixels.
- void Fl::box_shadow_width (int W)

  Set the box shadow width of all "shadow" boxtypes in pixels.
- bool Fl::cairo_autolink_context ()

  Gets the current autolink mode for Cairo support.
- void Fl::cairo_autolink_context (bool alink)

  When FLTK_HAVE_CAIRO is defined and cairo_autolink_context() is true, any current window dc is linked to a current Cairo context.
- cairo_t * **Fl::cairo_cc** ()

  Gets the current Cairo context linked with a fltk window.
- void Fl::cairo_cc (cairo_t *c, bool own=false)

  Sets the current Cairo context to `c`.
- void Fl::cairo_flush (cairo_t *c)

  Flush Cairo drawings on Cairo context `c`.
- cairo_t * Fl::cairo_make_current (Fl_Window *w)

  Provides a Cairo context for window wi.
- int Fl::check ()

  Same as Fl::wait(0).
- int Fl::clipboard_contains (const char *type)

  Returns non 0 if the clipboard contains data matching `type`.
- void Fl::copy (const char *stuff, int len, int destination=0, const char *type=Fl::clipboard_plain_text)

  Copies data to the selection buffer, the clipboard, or both.
- int **Fl::damage** ()

  If true then flush() will do something.

- void **Fl::damage** (int d)

  If true then flush() will do something.
- void Fl::default_atclose (Fl_Window ∗, void ∗)

  Default callback for window widgets.
- void Fl::display (const char ∗)

  Sets the X or Wayland display to use for all windows.
- int Fl::dnd ()

  Initiate a Drag And Drop operation.
- int Fl::draw_box_active ()

  Determines if the currently drawn box is active or inactive.
- int Fl::draw_GL_text_with_textures ()

  returns whether whether OpenGL uses textures to draw all text.
- void Fl::draw_GL_text_with_textures (int val)

  sets whether OpenGL uses textures to draw all text.
- Fl_Window ∗ Fl::first_window ()

  Returns the first top-level window in the list of shown() windows.
- void Fl::first_window (Fl_Window ∗)

  Sets the window that is returned by first_window().
- void Fl::flush ()

  Causes all the windows that need it to be redrawn and graphics forced out through the pipes.
- void **Fl::foreground** (uchar, uchar, uchar)

  Changes fl_color(FL_FOREGROUND_COLOR).
- void Fl::free_color (Fl_Color i, int overlay=0)

  Frees the specified color from the colormap, if applicable.
- Fl_Box_Draw_F ∗ **Fl::get_boxtype** (Fl_Boxtype)

  Gets the current box drawing function for the specified box type.
- unsigned Fl::get_color (Fl_Color i)

  Returns the RGB value(s) for the given FLTK color index.
- void Fl::get_color (Fl_Color i, uchar &red, uchar &green, uchar &blue)

  Returns the RGB value(s) for the given FLTK color index.
- void Fl::get_color (Fl_Color i, uchar &red, uchar &green, uchar &blue, uchar &alpha)

  Returns the RGBA value(s) for the given FLTK color index.
- const char ∗ Fl::get_font (Fl_Font)

  Gets the string for this face.
- const char ∗ Fl::get_font_name (Fl_Font, int ∗attributes=0)

  Get a human-readable string describing the family of this face.
- int Fl::get_font_sizes (Fl_Font, int ∗&sizep)

  Return an array of sizes in `sizep`.
- void Fl::get_system_colors ()

  Read the user preference colors from the system and use them to call Fl::foreground(), Fl::background(), and Fl::background2().
- int Fl::gl_visual (int, int ∗alist=0)

  This does the same thing as Fl::visual(int) but also requires OpenGL drawing to work.
- Fl_Window ∗ Fl::grab ()

  Returns the window that currently receives all events.
- void **Fl::grab** (Fl_Window &win)

  See grab(Fl_Window∗)
- void Fl::grab (Fl_Window ∗)

  Selects the window to grab.
- int **Fl::h** ()

  Returns the height in pixels of the main screen work area.

- int **Fl::has_check** (Fl_Timeout_Handler, void *=0)

  Returns 1 if the check exists and has not been called yet, 0 otherwise.
- int Fl::has_idle (Fl_Idle_Handler cb, void *data=0)

  Returns true if the specified idle callback is currently installed.
- int Fl::has_timeout (Fl_Timeout_Handler cb, void *data=0)

  Returns true if the timeout exists and has not been called yet.
- void Fl::hide_all_windows ()

  Hide all visible windows to make FLTK leave Fl::run().
- bool Fl::idle ()

  Returns whether at least one idle callback is currently set.
- int Fl::is_scheme (const char *name)

  Returns whether the current scheme is the given name.
- void Fl::keyboard_screen_scaling (int value)

  Controls the possibility to scale all windows by ctrl/+/-/0/ or cmd/+/-/0/.
- int Fl::lock ()

  Acquire the global UI lock for FLTK.
- int Fl::menu_linespacing ()

  Gets the default line spacing used by menus.
- void Fl::menu_linespacing (int H)

  Sets the default line spacing used by menus.
- Fl_Window * Fl::modal ()

  Returns the top-most modal() window currently shown.
- Fl_Window * Fl::next_window (const Fl_Window *)

  Returns the next top-level window in the list of shown() windows.
- float Fl::normalized_screen_scale (int n)

  Gets the normalized GUI scaling factor of screen number `n`.
- void Fl::normalized_screen_scale (int n, float factor)

  Sets the normalized GUI scaling factor of screen number `n`.
- Fl_Timestamp Fl::now (double offset=0)

  Set a time stamp at this point in time with optional signed offset in seconds.
- void Fl::own_colormap ()

  Makes FLTK use its `own colormap`.
- void Fl::paste (Fl_Widget &receiver)

  Backward compatibility only.
- void Fl::paste (Fl_Widget &receiver, int source, const char *type=Fl::clipboard_plain_text)

  Pastes the data from the selection buffer (`source` is 0) or the clipboard (`source` is 1) into `receiver`.
- int Fl::program_should_quit ()

  Returns non-zero when a request for program termination was received and accepted.
- void Fl::program_should_quit (int should_i)

  Indicate to the FLTK library whether a program termination request was received and accepted.
- Fl_Widget * Fl::readqueue ()

  Reads the default callback queue and returns the first widget.
- int Fl::ready ()

  This is similar to Fl::check() except this does not call Fl::flush() or any callbacks, which is useful if your program is in a state where such callbacks are illegal.
- void **Fl::redraw** ()

  Redraws all widgets.
- int Fl::reload_scheme ()

  Called internally when setting a new scheme according to scheme name.
- void Fl::remove_check (Fl_Timeout_Handler, void *=0)

  Removes a check callback.

- void **Fl::remove_clipboard_notify** (Fl_Clipboard_Notify_Handler h)

  Stop calling the specified callback when there are changes to the selection buffer or the clipboard.
- void **Fl::remove_fd** (int)

  Removes a file descriptor handler.
- void **Fl::remove_fd** (int, int when)

  Removes a file descriptor handler.
- void Fl::remove_idle (Fl_Idle_Handler cb, void ∗data=0)

  Removes the specified idle callback, if it is installed.
- int Fl::remove_next_timeout (Fl_Timeout_Handler cb, void ∗data=0, void ∗∗data_return=0)

  Remove the next matching timeout callback and return its `data` pointer.
- void Fl::remove_timeout (Fl_Timeout_Handler cb, void ∗data=0)

  Remove one or more matching timeout callbacks from the timer queue.
- void Fl::repeat_timeout (double t, Fl_Timeout_Handler cb, void ∗data=0)

  Repeats a timeout callback from the expiration of the previous timeout, allowing for more accurate timing.
- int Fl::run ()

  Calls Fl::wait()repeatedly as long as any windows are displayed.
- const char ∗ **Fl::scheme** ()

  See void scheme(const char ∗name)
- int Fl::scheme (const char ∗name)

  Sets the current widget scheme.
- int Fl::screen_count ()

  Gets the total count of available screens.
- void Fl::screen_dpi (float &h, float &v, int n=0)

  Gets the screen resolution in dots-per-inch for the given screen.
- Fl_Screen_Driver ∗ **Fl::screen_driver** ()

  Returns a pointer to the unique Fl_Screen_Driver object of the platform.
- int Fl::screen_num (int x, int y)

  Gets the screen number of a screen that contains the specified screen position `x`, `y`.
- int Fl::screen_num (int x, int y, int w, int h)

  Gets the screen number of the screen which intersects the most with the rectangle defined by `x`, `y`, `w`, `h`.
- float Fl::screen_scale (int n)

  Gets the GUI scaling factor of screen number `n`.
- void Fl::screen_scale (int n, float factor)

  Sets the GUI scaling factor of screen number `n`.
- int Fl::screen_scaling_supported ()

  Returns whether scaling factors are supported by this platform.
- void Fl::screen_work_area (int &X, int &Y, int &W, int &H)

  Gets the bounding box of the work area of the screen that contains the mouse pointer.
- void Fl::screen_work_area (int &X, int &Y, int &W, int &H, int mx, int my)

  Gets the bounding box of the work area of a screen that contains the specified screen position `mx`, `my`.
- void Fl::screen_work_area (int &X, int &Y, int &W, int &H, int n)

  Gets the bounding box of the work area of the given screen.
- void Fl::screen_xywh (int &X, int &Y, int &W, int &H)

  Gets the bounding box of a screen that contains the mouse pointer.
- void Fl::screen_xywh (int &X, int &Y, int &W, int &H, int mx, int my)

  Gets the bounding box of a screen that contains the specified screen position `mx`, `my`.
- void Fl::screen_xywh (int &X, int &Y, int &W, int &H, int mx, int my, int mw, int mh)

  Gets the screen bounding rect for the screen which intersects the most with the rectangle defined by `mx`, `my`, `mw`, `mh`.
- void Fl::screen_xywh (int &X, int &Y, int &W, int &H, int n)

  Gets the screen bounding rect for the given screen.
- int Fl::scrollbar_size ()

Gets the default scrollbar size used by Fl_Browser_, Fl_Help_View, Fl_Scroll, and Fl_Text_Display widgets.

- void Fl::scrollbar_size (int W)

  Sets the default scrollbar size that is used by the Fl_Browser_, Fl_Help_View, Fl_Scroll, and Fl_Text_Display widgets.
- double Fl::seconds_between (Fl_Timestamp &back, Fl_Timestamp &further_back)

  Return the time in seconds between two time stamps.
- double Fl::seconds_since (Fl_Timestamp &then)

  Return the time in seconds between now and a previously taken time stamp.
- void Fl::selection (Fl_Widget &owner, const char *, int len)

  Changes the current selection.
- Fl_Widget * Fl::selection_owner ()

  back-compatibility only: Gets the widget owning the current selection
- void Fl::selection_owner (Fl_Widget *)

  Back-compatibility only: The single-argument call can be used to move the selection to another widget or to set the owner to NULL, without changing the actual text of the selection.
- int Fl::selection_to_clipboard ()

  Returns the current selection_to_clipboard mode.
- void Fl::selection_to_clipboard (int mode)

  Copies selections on X11 directly to the clipboard if enabled.
- void **Fl::set_abort** (Fl_Abort_Handler f)

  For back compatibility, sets the void Fl::fatal handler callback.
- void Fl::set_atclose (Fl_Atclose_Handler f)

  For back compatibility, sets the Fl::atclose handler callback.
- void Fl::set_box_color (Fl_Color)

  Sets the drawing color for the box that is currently drawn.
- void Fl::set_boxtype (Fl_Boxtype, Fl_Box_Draw_F *, uchar, uchar, uchar, uchar, Fl_Box_Draw_Focus_F *=NULL)

  Sets the function to call to draw a specific box type.
- void **Fl::set_boxtype** (Fl_Boxtype, Fl_Boxtype from)

  Copies the from boxtype.
- void Fl::set_color (Fl_Color i, unsigned c)

  Sets an entry in the fl_color index table.
- void Fl::set_color (Fl_Color, uchar, uchar, uchar)

  Sets an entry in the fl_color index table.
- void Fl::set_color (Fl_Color, uchar, uchar, uchar, uchar)

  Sets an entry in the fl_color index table.
- void Fl::set_font (Fl_Font, const char *)

  Changes a face.
- void **Fl::set_font** (Fl_Font, Fl_Font)

  Copies one face to another.
- Fl_Font Fl::set_fonts (const char *=0)

  FLTK will open the display, and add every fonts on the server to the face table.
- void **Fl::set_labeltype** (Fl_Labeltype, Fl_Label_Draw_F *, Fl_Label_Measure_F *)

  Sets the functions to call to draw and measure a specific labeltype.
- void **Fl::set_labeltype** (Fl_Labeltype, Fl_Labeltype from)

  Sets the functions to call to draw and measure a specific labeltype.
- int **Fl::system** (const char *command)

  Run a command line on the computer.
- Fl_System_Driver * **Fl::system_driver** ()

  Returns a pointer to the unique Fl_System_Driver object of the platform.
- void * Fl::thread_message ()

  Returns the last message sent by a child thread.

- long Fl::ticks_between (Fl_Timestamp &back, Fl_Timestamp &further_back)

  Return the time in ticks (60Hz) between two time stamps.
- long Fl::ticks_since (Fl_Timestamp &then)

  Return the time in ticks (60Hz) between now and a previously taken time stamp.
- std::vector< TimeoutData > Fl::timeout_list ()

  Return a list of all currently running timeouts.
- void Fl::unlock ()

  Release the global UI lock set by Fl::lock().
- int Fl::use_high_res_GL ()

  Returns whether GL windows should be drawn at high resolution on Apple computers with retina displays.
- void Fl::use_high_res_GL (int val)

  Sets whether GL windows should be drawn at high resolution on Apple computers with retina displays.
- double Fl::version ()

  Returns the compiled-in value of the FL_VERSION constant.
- int Fl::visual (int)

  Selects a visual so that your graphics are drawn correctly.
- int **Fl::w** ()

  Returns the width in pixels of the main screen work area.
- int Fl::wait ()

  Waits until "something happens" and then returns.
- double Fl::wait (double time)

  Waits a maximum of `time_to_wait` seconds or until "something happens".
- int **Fl::x** ()

  Returns the leftmost x coordinate of the main screen work area.
- int **Fl::y** ()

  Returns the topmost y coordinate of the main screen work area.

Variables

- void(* Fl::atclose)(Fl_Window *, void *) = Fl::default_atclose

  Back compatibility: default window callback handler.
- void(* Fl::error)(const char *,...) = Fl_System_Driver::error

  FLTK calls Fl::error() to output a normal error message.
- void(* Fl::fatal)(const char *,...) = Fl_System_Driver::fatal

  FLTK calls Fl::fatal() to output a fatal error message.
- const char * **fl_local_alt**

  string pointer used in shortcuts, you can change it to another language
- const char * **fl_local_ctrl**

  string pointer used in shortcuts, you can change it to another language
- const char * **fl_local_meta**

  string pointer used in shortcuts, you can change it to another language
- const char * **fl_local_shift**

  string pointer used in shortcuts, you can change it to another language
- const char *const Fl::help = helpmsg+13

  Usage string displayed if Fl::args() detects an invalid argument.
- void(* Fl::warning)(const char *,...) = Fl_System_Driver::warning

  FLTK calls Fl::warning() to output a warning message.

14.14.1 Detailed Description

Fl namespace.

14.15 Fl.H

Go to the documentation of this file.

```
//
// Main header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2026 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef Fl_H
#define Fl_H

#include <FL/fl_config.h>             // build configuration
#include <FL/Fl_Export.H>             // for FL_EXPORT
#include <FL/platform_types.h>        // for FL_SOCKET
#include <FL/fl_casts.H>              // type casting
#include <FL/core/function_types.H>   // widget callbacks and services
#include <FL/core/events.H>           // global event handling
#include <FL/core/options.H>          // system and application setting
#include <FL/core/pen_events.H>       // pen and tablet events
#include <FL/Fl_Widget_Tracker.H>     // historically included here

#ifdef FLTK_HAVE_CAIRO
# include <FL/Fl_Cairo.H>
#endif

#include "fl_utf8.h"
#include "Enumerations.H"
#ifndef Fl_Object
#  define Fl_Object Fl_Widget
#endif

#ifdef check
#  undef check
#endif

#ifdef BSD
#  undef BSD
#endif

#include <string.h> // FIXME: Fl::is_scheme(): strcmp needs string.h
#include <vector>

class Fl_Widget;
class Fl_Window;
class Fl_Image;
struct Fl_Label;
class Fl_Screen_Driver;
class Fl_System_Driver;

// Pointers you can use to change FLTK to another language.
// Note: Similar pointers are defined in FL/fl_ask.H and src/fl_ask.cxx

extern FL_EXPORT const char* fl_local_alt;
extern FL_EXPORT const char* fl_local_ctrl;
extern FL_EXPORT const char* fl_local_meta;
extern FL_EXPORT const char* fl_local_shift;


namespace Fl {

FL_EXPORT extern Fl_Screen_Driver *screen_driver();
FL_EXPORT extern Fl_System_Driver *system_driver();

#ifdef __APPLE__ // deprecated in 1.4 - only for compatibility with 1.3
FL_EXPORT extern void reset_marked_text();
FL_EXPORT extern void insertion_point_location(int x, int y, int height);
#endif

FL_EXPORT extern int box_shadow_width();
FL_EXPORT extern void box_shadow_width(int W);

FL_EXPORT extern int box_border_radius_max();
FL_EXPORT extern void box_border_radius_max(int R);

// should be private:
```

```

#ifndef FL_DOXYGEN
FL_EXPORT extern int damage_;
FL_EXPORT extern Fl_Widget* selection_owner_;
FL_EXPORT extern Fl_Window* modal_;
FL_EXPORT extern Fl_Window* grab_;
FL_EXPORT extern int compose_state; // used for dead keys (Windows) or marked text (MacOS)
#endif // FL_DOXYGEN


FL_EXPORT inline void damage(int d) {damage_ = d;}

FL_EXPORT extern bool idle();

#ifndef FL_DOXYGEN
FL_EXPORT extern const char* scheme_;
FL_EXPORT extern Fl_Image* scheme_bg_;

FL_EXPORT extern int scrollbar_size_;
FL_EXPORT extern int menu_linespacing_; // STR #2927
#endif

// API version number
FL_EXPORT extern double version();
FL_EXPORT extern int api_version();

// ABI version number
FL_EXPORT extern int abi_version();

FL_EXPORT inline int abi_check(const int val = FL_ABI_VERSION) {
  return val == abi_version();
}

// argument parsers:
FL_EXPORT extern int arg(int argc, char **argv, int& i);
FL_EXPORT extern int args(int argc, char **argv, int& i, Fl_Args_Handler cb = 0);
FL_EXPORT extern void args(int argc, char **argv);

FL_EXPORT extern const char* const help;

// things called by initialization:
FL_EXPORT extern void display(const char*);
FL_EXPORT extern int visual(int);

FL_EXPORT extern int gl_visual(int, int *alist=0); // platform dependent
FL_EXPORT extern void own_colormap();
FL_EXPORT extern void get_system_colors();
FL_EXPORT extern void foreground(uchar, uchar, uchar);
FL_EXPORT extern void background(uchar, uchar, uchar);
FL_EXPORT extern void background2(uchar, uchar, uchar);

// schemes:
FL_EXPORT extern int scheme(const char *name);
FL_EXPORT inline const char* scheme() {return scheme_;}

FL_EXPORT inline int is_scheme(const char *name) {
  return (scheme_ && name && !strcmp(name,scheme_));
}

FL_EXPORT extern int reload_scheme(); // defined in 'src/Fl_get_system_colors.cxx'

FL_EXPORT extern int scrollbar_size();
FL_EXPORT extern void scrollbar_size(int W);
FL_EXPORT extern int menu_linespacing();
FL_EXPORT extern void menu_linespacing(int H);

// execution:
FL_EXPORT extern int wait();
FL_EXPORT extern double wait(double time);
FL_EXPORT extern int check();
FL_EXPORT extern int ready();
FL_EXPORT extern int run();
FL_EXPORT extern int program_should_quit();
FL_EXPORT extern void program_should_quit(int should_i);
FL_EXPORT extern void hide_all_windows();

FL_EXPORT extern Fl_Widget* readqueue();

//
// cross-platform timer support
//

FL_EXPORT extern void add_timeout(double t, Fl_Timeout_Handler cb, void *data = 0);
FL_EXPORT extern void repeat_timeout(double t, Fl_Timeout_Handler cb, void *data = 0);
FL_EXPORT extern int  has_timeout(Fl_Timeout_Handler cb, void *data = 0);
FL_EXPORT extern void remove_timeout(Fl_Timeout_Handler cb, void *data = 0);
FL_EXPORT extern int remove_next_timeout(Fl_Timeout_Handler cb, void *data = 0, void **data_return =
```

```
0);
typedef struct { double t; Fl_Timeout_Handler cb; void *data; } TimeoutData;
FL_EXPORT extern std::vector<TimeoutData> timeout_list();

FL_EXPORT extern void add_check(Fl_Timeout_Handler, void* = 0);
FL_EXPORT extern int  has_check(Fl_Timeout_Handler, void* = 0);
FL_EXPORT extern void remove_check(Fl_Timeout_Handler, void* = 0);

FL_EXPORT extern Fl_Timestamp now(double offset = 0);
FL_EXPORT extern double seconds_since(Fl_Timestamp& then);
FL_EXPORT extern double seconds_between(Fl_Timestamp& back, Fl_Timestamp& further_back);
FL_EXPORT extern long ticks_since(Fl_Timestamp& then);
FL_EXPORT extern long ticks_between(Fl_Timestamp& back, Fl_Timestamp& further_back);

FL_EXPORT extern void add_fd(int fd, int when, Fl_FD_Handler cb, void* = 0); // platform dependent
FL_EXPORT extern void add_fd(int fd, Fl_FD_Handler cb, void* = 0); // platform dependent
FL_EXPORT extern void remove_fd(int, int when); // platform dependent
FL_EXPORT extern void remove_fd(int); // platform dependent

FL_EXPORT extern void add_idle(Fl_Idle_Handler cb, void* data = 0);
FL_EXPORT extern int  has_idle(Fl_Idle_Handler cb, void* data = 0);
FL_EXPORT extern void remove_idle(Fl_Idle_Handler cb, void* data = 0);
FL_EXPORT extern void add_idle(Fl_Old_Idle_Handler cb);
FL_EXPORT inline int damage() {return damage_;}
FL_EXPORT extern void redraw();
FL_EXPORT extern void flush();

FL_EXPORT extern void (*warning)(const char*, ...);

FL_EXPORT extern void (*error)(const char*, ...);

FL_EXPORT extern void (*fatal)(const char*, ...);

FL_EXPORT extern Fl_Window* first_window();
FL_EXPORT extern void first_window(Fl_Window*);
FL_EXPORT extern Fl_Window* next_window(const Fl_Window*);

FL_EXPORT inline Fl_Window* modal() {return modal_;}

FL_EXPORT inline Fl_Window* grab() {return grab_;}

FL_EXPORT extern void grab(Fl_Window*); // platform dependent

FL_EXPORT extern void copy(const char *stuff, int len, int destination = 0,
                  const char *type = Fl::clipboard_plain_text);

FL_EXPORT extern void selection_to_clipboard(int mode);
FL_EXPORT extern int selection_to_clipboard();

FL_EXPORT extern void paste(Fl_Widget &receiver, int source, const char *type =
Fl::clipboard_plain_text);

FL_EXPORT extern void add_clipboard_notify(Fl_Clipboard_Notify_Handler h, void *data = 0);

FL_EXPORT extern void remove_clipboard_notify(Fl_Clipboard_Notify_Handler h);

FL_EXPORT extern int clipboard_contains(const char *type);

FL_EXPORT extern int dnd(); // platform dependent

// These are for back-compatibility only:
FL_EXPORT inline Fl_Widget* selection_owner() {return selection_owner_;}
FL_EXPORT extern void selection_owner(Fl_Widget*);
FL_EXPORT extern void selection(Fl_Widget &owner, const char*, int len);
FL_EXPORT extern void paste(Fl_Widget &receiver);

FL_EXPORT extern int x(); // via screen driver
FL_EXPORT extern int y(); // via screen driver
FL_EXPORT extern int w(); // via screen driver
FL_EXPORT extern int h(); // via screen driver

// multi-head support:
FL_EXPORT extern int screen_count(); // via screen driver
FL_EXPORT extern void screen_xywh(int &X, int &Y, int &W, int &H); // via screen driver
FL_EXPORT extern void screen_xywh(int &X, int &Y, int &W, int &H, int mx, int my); // via screen
driver
FL_EXPORT extern void screen_xywh(int &X, int &Y, int &W, int &H, int n); // via screen driver
FL_EXPORT extern void screen_xywh(int &X, int &Y, int &W, int &H, int mx, int my, int mw, int mh); //
via screen driver
FL_EXPORT extern int screen_num(int x, int y); // via screen driver
FL_EXPORT extern int screen_num(int x, int y, int w, int h); // via screen driver
FL_EXPORT extern void screen_dpi(float &h, float &v, int n=0); // via screen driver
FL_EXPORT extern void screen_work_area(int &X, int &Y, int &W, int &H, int mx, int my); // via screen
driver
FL_EXPORT extern void screen_work_area(int &X, int &Y, int &W, int &H, int n); // via screen driver
FL_EXPORT extern void screen_work_area(int &X, int &Y, int &W, int &H); // via screen driver
FL_EXPORT extern float screen_scale(int n); // via screen driver
```

```
FL_EXPORT extern void screen_scale(int n, float factor); // via screen driver
FL_EXPORT extern float normalized_screen_scale(int n);
FL_EXPORT extern void normalized_screen_scale(int n, float factor);
FL_EXPORT extern int screen_scaling_supported();
FL_EXPORT extern void keyboard_screen_scaling(int value);

// color map:
FL_EXPORT extern void   set_color(Fl_Color, uchar, uchar, uchar);
FL_EXPORT extern void   set_color(Fl_Color, uchar, uchar, uchar, uchar);

FL_EXPORT extern void   set_color(Fl_Color i, unsigned c); // platform dependent
FL_EXPORT extern unsigned get_color(Fl_Color i);
FL_EXPORT extern void   get_color(Fl_Color i, uchar &red, uchar &green, uchar &blue);
FL_EXPORT extern void   get_color(Fl_Color i, uchar &red, uchar &green, uchar &blue, uchar &alpha);

FL_EXPORT extern void   free_color(Fl_Color i, int overlay = 0); // platform dependent

// fonts:
FL_EXPORT extern const char* get_font(Fl_Font);

FL_EXPORT extern const char* get_font_name(Fl_Font, int* attributes = 0);

FL_EXPORT extern int get_font_sizes(Fl_Font, int*& sizep);
FL_EXPORT extern void set_font(Fl_Font, const char*);
FL_EXPORT extern void set_font(Fl_Font, Fl_Font);

FL_EXPORT extern Fl_Font set_fonts(const char* = 0); // platform dependent

// <Hack to re-order the 'Drawing functions' group>
// labeltypes:
FL_EXPORT extern void set_labeltype(Fl_Labeltype,Fl_Label_Draw_F*,Fl_Label_Measure_F*);
FL_EXPORT extern void set_labeltype(Fl_Labeltype, Fl_Labeltype from); // is it defined ?

// boxtypes:
FL_EXPORT extern Fl_Box_Draw_F *get_boxtype(Fl_Boxtype);
FL_EXPORT extern void set_boxtype(Fl_Boxtype, Fl_Box_Draw_F*,
                        uchar, uchar, uchar, uchar,
                        Fl_Box_Draw_Focus_F* =NULL);
FL_EXPORT extern void set_boxtype(Fl_Boxtype, Fl_Boxtype from);
FL_EXPORT extern int box_dx(Fl_Boxtype);
FL_EXPORT extern int box_dy(Fl_Boxtype);
FL_EXPORT extern int box_dw(Fl_Boxtype);
FL_EXPORT extern int box_dh(Fl_Boxtype);

FL_EXPORT extern bool box_bg(Fl_Boxtype);

FL_EXPORT extern int draw_box_active();
FL_EXPORT extern Fl_Color box_color(Fl_Color);
FL_EXPORT extern void set_box_color(Fl_Color);

// back compatibility:

FL_EXPORT inline void set_abort(Fl_Abort_Handler f) {fatal = f;}
FL_EXPORT extern void (*atclose)(Fl_Window*,void*);
FL_EXPORT extern void default_atclose(Fl_Window*,void*);

FL_EXPORT inline void set_atclose(Fl_Atclose_Handler f) {atclose = f;}

FL_EXPORT inline void grab(Fl_Window& win) {grab(&win);}

// --- FLTK Multithreading support functions ---

// Thread locking:
FL_EXPORT extern int lock();
FL_EXPORT extern void unlock();
// Thread wakup and defered calls:
FL_EXPORT extern void awake();
FL_DEPRECATED("since 1.5.0 - use Fl::awake() or Fl::awake(handler, user_data) instead",
FL_EXPORT extern void awake(void* message));
FL_EXPORT extern int awake(Fl_Awake_Handler handler, void* user_data=nullptr);
FL_EXPORT extern int awake_once(Fl_Awake_Handler handler, void* user_data=nullptr);
FL_DEPRECATED("since 1.5.0 - use Fl::awake() or Fl::awake(handler, user_data) instead",
FL_EXPORT extern void* thread_message()); // platform dependent

FL_EXPORT extern void use_high_res_GL(int val);
FL_EXPORT extern int use_high_res_GL();

FL_EXPORT extern void draw_GL_text_with_textures(int val);
FL_EXPORT extern int draw_GL_text_with_textures();

FL_EXPORT extern int system(const char *command);

// Convert Windows commandline arguments to UTF-8 (documented in src/Fl.cxx)
FL_EXPORT extern int args_to_utf8(int argc, char ** &argv);

#ifdef FLTK_HAVE_CAIRO
// Cairo support API
```

```

FL_EXPORT extern cairo_t *cairo_make_current(Fl_Window *w);
FL_EXPORT extern void cairo_autolink_context(bool alink);
FL_EXPORT extern bool cairo_autolink_context();
FL_EXPORT extern cairo_t *cairo_cc();
FL_EXPORT extern void cairo_cc(cairo_t *c, bool own=false);
FL_EXPORT extern void cairo_flush(cairo_t *c);

#endif // FLTK_HAVE_CAIRO

} // namespace Fl


#endif // !Fl_H
```

14.16 Fl_Adjuster.H

```
//
// Adjuster widget header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Adjuster widget . */

// 3-button "slider", made for Nuke

#ifndef Fl_Adjuster_H
#define Fl_Adjuster_H

#ifndef Fl_Valuator_H
#include "Fl_Valuator.H"
#endif

class FL_EXPORT Fl_Adjuster : public Fl_Valuator {
  int drag;
  int ix;
  int soft_;
protected:
  void draw() override;
  int handle(int) override;
  void value_damage() override;
public:
  Fl_Adjuster(int X,int Y,int W,int H,const char *l=0);
  void soft(int s) {soft_ = s;}
  int soft() const {return soft_;}
};

#endif
```

14.17 Fl_Anim_GIF_Image.H

```
//
// Fl_Anim_GIF_Image class header for the Fast Light Tool Kit (FLTK).
//
// Copyright 2016-2023 by Christian Grabner <wcout@gmx.net>.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef Fl_Anim_Gif_Image_H
#define Fl_Anim_Gif_Image_H

```

```
// forward declarations
class Fl_Image;
class Fl_Widget;

#include <FL/Fl_GIF_Image.H>

// Load and display animated GIF images
class FL_EXPORT Fl_Anim_GIF_Image : public Fl_GIF_Image {

  class FrameInfo; // internal helper class

public:

  enum Flags {
    DONT_START = 1,
    DONT_RESIZE_CANVAS = 2,
    DONT_SET_AS_IMAGE = 4,
    OPTIMIZE_MEMORY = 8,
    LOG_FLAG = 64,
    DEBUG_FLAG = 128
  };

  // -- constructors and destructor
  Fl_Anim_GIF_Image(const char *filename, Fl_Widget *canvas = 0, unsigned short flags = 0);
  Fl_Anim_GIF_Image(const char* imagename, const unsigned char *data,
                    const size_t length, Fl_Widget *canvas = 0,
                    unsigned short flags = 0);
  Fl_Anim_GIF_Image();
  ~Fl_Anim_GIF_Image() override;

  // -- file handling
  bool load(const char *name, const unsigned char *imgdata=NULL, size_t imglength=0);
  bool valid() const;

  // -- getters and setters
  void frame_uncache(bool uncache);
  bool frame_uncache() const;
  double delay(int frame_) const;
  void delay(int frame, double delay);
  void canvas(Fl_Widget *canvas, unsigned short flags = 0);
  Fl_Widget *canvas() const;
  int canvas_w() const;
  int canvas_h() const;
  bool is_animated() const;
  const char *name() const;
  void speed(double speed);
  double speed() const;

  // -- animation
  int frames() const;
  void frame(int frame);
  int frame() const;
  Fl_Image *image() const;
  Fl_Image *image(int frame) const;
  bool start();
  bool stop();
  bool next();

  bool playing() const { return valid() && Fl::has_timeout(cb_animate, (void *)this); }

  // -- image data
  Fl_Anim_GIF_Image& resize(int w, int h);
  Fl_Anim_GIF_Image& resize(double scale);
  int frame_x(int frame) const;
  int frame_y(int frame) const;
  int frame_w(int frame) const;
  int frame_h(int frame) const;

  // -- overridden methods
  void color_average(Fl_Color c, float i) override;
  Fl_Image *copy(int W, int H) const override;
  Fl_Image *copy() const { return Fl_Pixmap::copy(); }
  void desaturate() override;
  void draw(int x, int y, int w, int h, int cx = 0, int cy = 0) override;
  void uncache() override;

  // -- debugging and logging
  int debug() const;

  // -- static methods
  static int frame_count(const char *name, const unsigned char *imgdata = NULL, size_t imglength = 0);

  static bool loop;

  static double min_delay;

protected:
```

```
  bool next_frame();
  void clear_frames();
  void set_frame(int frame);

  static void cb_animate(void *d);
  void scale_frame();
  void set_frame();
  void on_frame_data(Fl_GIF_Image::GIF_FRAME &f) override;
  void on_extension_data(Fl_GIF_Image::GIF_FRAME &f) override;

private:

  char *name_;
  unsigned short flags_;
  Fl_Widget *canvas_;
  bool uncache_;
  bool valid_;
  int frame_; // current frame
  double speed_;
  FrameInfo *fi_;
};

#endif // Fl_Anim_Gif_Image_H
```

14.18 fl_ask.H File Reference

API for common dialogs.

```
#include <FL/Enumerations.H>
#include <FL/fl_attr.h>
#include <string>
```

Enumerations

- enum Fl_Beep {
  FL_BEEP_DEFAULT = 0 , FL_BEEP_MESSAGE , FL_BEEP_ERROR , FL_BEEP_QUESTION , FL_BEEP_PASSWORD , FL_BEEP_NOTIFICATION }

  Defines the different system beeps available.

Functions

- void void **fl_alert** (const char *,...) __fl_attr((__format__(__printf__
- void fl_beep (int type=FL_BEEP_DEFAULT)

  Emits a system beep.
- void void int **fl_choice** (const char *q, const char *b0, const char *b1, const char *b2,...) __fl_attr((__format__(__printf__
- void void int const char const char int **fl_choice_n** (const char *q, const char *b0, const char *b1, const char *b2,...) __fl_attr((__format__(__printf__
- void void int const char * **fl_input** (const char *label, const char *deflt=0,...) __fl_attr((__format__(__printf__
- void void int const char const char int const char * **fl_input** (int maxchar, const char *label, const char *deflt=0,...) __fl_attr((__format__(__printf__
- void void int const char const char int const char const char std::string std::string **fl_input_str** (int &ret, int maxchar, const char *label, const char *deflt=0,...) __fl_attr((__format__(__printf__
- void void int const char const char int const char const char std::string **fl_input_str** (int maxchar, const char *label, const char *deflt=0,...) __fl_attr((__format__(__printf__
- void **fl_message** (const char *,...) __fl_attr((__format__(__printf__
- void **fl_message_font** (Fl_Font f, Fl_Fontsize s)
- void fl_message_hotspot (int enable)

  Sets whether or not to move the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password() to follow the mouse pointer.
- int fl_message_hotspot (void)

Gets whether or not to move the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password() to follow the mouse pointer.

- void void int const char const char int const char const char std::string std::string std::string std::string Fl_Widget ∗ fl_message_icon ()

  Gets the Fl_Box icon container of the current default dialog used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().
- void fl_message_icon_label (const char ∗str)

  Sets the icon label of the dialog window used in many common dialogs.
- void fl_message_position (const int x, const int y, const int center=0)

  Sets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().
- void fl_message_position (Fl_Widget &widget)
- void fl_message_position (Fl_Widget ∗widget)

  Sets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().
- int fl_message_position (int ∗x=0, int ∗y=0)

  Gets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().
- void fl_message_title (const char ∗title)

  Sets the title of the dialog window used in many common dialogs.
- void fl_message_title_default (const char ∗title)

  Sets the default title of the dialog window used in many common dialogs.
- void void int const char const char ∗ **fl_password** (const char ∗label, const char ∗deflt=0,...) __fl_attr((__format__(__printf__
- void void int const char const char int const char const char ∗ **fl_password** (int maxchar, const char ∗label, const char ∗deflt=0,...) __fl_attr((__format__(__printf__
- void void int const char const char int const char const char std::string std::string std::string std::string **fl_password_str** (int &ret, int maxchar, const char ∗label, const char ∗deflt=0,...) __fl_attr((__format__(__printf__
- void void int const char const char int const char const char std::string std::string std::string **fl_password_str** (int maxchar, const char ∗label, const char ∗deflt=0,...) __fl_attr((__format__(__printf__

Variables

- const char ∗ **fl_cancel**

  string pointer used in common dialogs, you can change it to another language
- const char ∗ **fl_close**

  string pointer used in common dialogs, you can change it to another language
- Fl_Font **fl_message_font_**
- Fl_Fontsize **fl_message_size_**
- const char ∗ **fl_no**

  string pointer used in common dialogs, you can change it to another language
- const char ∗ **fl_ok**

  string pointer used in common dialogs, you can change it to another language
- const char ∗ **fl_yes**

  string pointer used in common dialogs, you can change it to another language

14.18.1 Detailed Description

API for common dialogs.

14.18.2 Enumeration Type Documentation

14.18.2.1 Fl_Beep

enum Fl_Beep

Defines the different system beeps available.

Some systems may play different sounds or use different sound volume depending on the Fl_Beep value. The implementation is platform dependent.

See also

fl_beep(int)

Enumerator

FL_BEEP_DEFAULT	Default beep.
FL_BEEP_MESSAGE	Message beep.
FL_BEEP_ERROR	Error beep.
FL_BEEP_QUESTION	Question beep.
FL_BEEP_PASSWORD	Password beep.
FL_BEEP_NOTIFICATION	Notification beep.

14.18.3 Function Documentation

14.18.3.1 fl_message_position()

```
void fl_message_position (
            Fl_Widget & widget )  [inline]
```

See also

fl_message_position(Fl_Widget *widget).

14.19 fl_ask.H

Go to the documentation of this file.

```
00001 //
00002 // Standard dialog header file for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 1998-2025 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file.  If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 //     https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 //     https://www.fltk.org/bugs.php
00015 //
00016
00021 #ifndef _FL_fl_ask_H_
00022 #define _FL_fl_ask_H_
00023
00024 #include <FL/Enumerations.H>
00025 #include <FL/fl_attr.h>
00026
00027 #include <string>
00028
00029 // Note to FLTK developers: fl_ask() had been deprecated since FLTK 1.1.7
00030 // (released Jan 17, 2006 !) but was still available in FLTK 1.4.x.
00031 // FLTK 1.5 disables it by using an *undocumented* preprocessor macro.
00032 // This allows us to restore it (as a last resort) if there are too many
00033 // complaints that fl_ask() is missing. If users don't complain, the
00034 // conditional implementation of fl_ask() should be removed in FLTK 1.6.
00035
00036 #ifndef FL_DOXYGEN
```

```

// undocumented feature: include fl_ask() if requested; see comment above.

#ifndef FLTK_INCLUDE_FL_ASK
#define FLTK_INCLUDE_FL_ASK 0  // set this to 1 to include fl_ask()
#endif

#endif // FL_DOXYGEN

class Fl_Widget;

enum Fl_Beep {
  FL_BEEP_DEFAULT = 0,
  FL_BEEP_MESSAGE,
  FL_BEEP_ERROR,
  FL_BEEP_QUESTION,
  FL_BEEP_PASSWORD,
  FL_BEEP_NOTIFICATION
};

FL_EXPORT void fl_beep(int type = FL_BEEP_DEFAULT);
FL_EXPORT void fl_message(const char *, ...) __fl_attr((__format__(__printf__, 1, 2)));
FL_EXPORT void fl_alert(const char *, ...) __fl_attr((__format__(__printf__, 1, 2)));

#if FLTK_INCLUDE_FL_ASK

// fl_ask() is deprecated since it uses "Yes" and "No" for the buttons,
// which does not conform to the current FLTK Human Interface Guidelines.
// Use fl_choice() or fl_choice_n() with the appropriate verbs instead.

FL_EXPORT int fl_ask(const char *, ...) __fl_attr((__format__(__printf__, 1, 2), __deprecated__));

#endif // FLTK_INCLUDE_FL_ASK

FL_EXPORT int fl_choice(const char *q, const char *b0, const char *b1, const char *b2, ...)
    __fl_attr((__format__(__printf__, 1, 5)));
FL_EXPORT const char *fl_input(const char *label, const char *deflt = 0, ...)
__fl_attr((__format__(__printf__, 1, 3)));
FL_EXPORT const char *fl_password(const char *label, const char *deflt = 0, ...)
    __fl_attr((__format__(__printf__, 1, 3)));

// since FLTK 1.3.8:
// - fl_choice_n()     with extended return value (-2, -1, 0, 1, 2)

FL_EXPORT int fl_choice_n(const char *q, const char *b0, const char *b1, const char *b2, ...)
    __fl_attr((__format__(__printf__, 1, 5)));

// since FLTK 1.4.0: with 'maxchar' to limit input size

FL_EXPORT const char *fl_input(int maxchar, const char *label, const char *deflt = 0, ...)
    __fl_attr((__format__(__printf__, 2, 4)));

FL_EXPORT const char *fl_password(int maxchar, const char *label, const char *deflt = 0, ...)
    __fl_attr((__format__(__printf__, 2, 4)));

// since FLTK 1.4.0 -- optional
// since FLTK 1.5.0 -- always

// - fl_input_str()    with limited input size, returns std::string
// - fl_password_str() with limited input size, returns std::string

FL_EXPORT std::string fl_input_str(int maxchar, const char *label, const char *deflt = 0, ...)
    __fl_attr((__format__(__printf__, 2, 4)));

FL_EXPORT std::string fl_input_str(int &ret, int maxchar, const char *label, const char *deflt = 0,
...)
    __fl_attr((__format__(__printf__, 3, 5)));

FL_EXPORT std::string fl_password_str(int maxchar, const char *label, const char *deflt = 0, ...)
    __fl_attr((__format__(__printf__, 2, 4)));

FL_EXPORT std::string fl_password_str(int &ret, int maxchar, const char *label, const char *deflt = 0,
...)
    __fl_attr((__format__(__printf__, 3, 5)));

FL_EXPORT Fl_Widget *fl_message_icon();
extern FL_EXPORT Fl_Font fl_message_font_;
extern FL_EXPORT Fl_Fontsize fl_message_size_;
inline void fl_message_font(Fl_Font f, Fl_Fontsize s) {
  fl_message_font_ = f;
  fl_message_size_ = s;
}

FL_EXPORT void fl_message_hotspot(int enable);
FL_EXPORT int fl_message_hotspot(void);

// since FLTK 1.4.0: fl_message_position()
```

```

FL_EXPORT void fl_message_position(const int x, const int y, const int center = 0);
FL_EXPORT void fl_message_position(Fl_Widget *widget);
FL_EXPORT int fl_message_position(int *x = 0, int *y = 0);

inline void fl_message_position(Fl_Widget &widget) {
  fl_message_position(&widget);
}

FL_EXPORT void fl_message_title(const char *title);
FL_EXPORT void fl_message_title_default(const char *title);

FL_EXPORT void fl_message_icon_label(const char *str);

// pointers you can use to change FLTK to another language:
extern FL_EXPORT const char *fl_no;
extern FL_EXPORT const char *fl_yes;
extern FL_EXPORT const char *fl_ok;
extern FL_EXPORT const char *fl_cancel;
extern FL_EXPORT const char *fl_close;

#endif // !_FL_fl_ask_H_
```

14.20 fl_attr.h File Reference

This file defines compiler-specific macros.

Macros

- #define __fl_attr(x)

  This section lists macros for Doxygen documentation only.
- #define FL_DEPRECATED(msg, func)

  Enclosing a function or method in FL_DEPRECATED marks it as no longer recommended.
- #define FL_OVERRIDE override

  This macro makes it safe to use the C++11 keyword `override` with older compilers.

14.20.1 Detailed Description

This file defines compiler-specific macros.

14.20.2 Macro Definition Documentation

14.20.2.1 __fl_attr

```
#define __fl_attr(
              x )
```

This section lists macros for Doxygen documentation only.
The next section will define the actual macros based on the compile used and based on the capabilities of the version of that compiler. To be used in prototypes with a variable list of arguments. This macro helps detection of mismatches between format string and argument list at compilation time.
Usage example: FL/fl_ask.H

14.20.2.2 FL_DEPRECATED

```
#define FL_DEPRECATED(
              msg,
              func )
```

Value:

```
/##*##* \deprecated msg *##/ \
func
```

Enclosing a function or method in FL_DEPRECATED marks it as no longer recommended.
This macro syntax can not be used if the return type contains a comma, which is not the case in FLTK.

```
FL_DEPRECATED("Outdated, don't use", int position()) { return position_; }
```

14.20.2.3 FL_OVERRIDE

`#define FL_OVERRIDE override`

This macro makes it safe to use the C++11 keyword `override` with older compilers.
Since FLTK 1.5.0 a C++11 capable compiler is required, hence we can safely define FL_OVERRIDE unconditionally as 'override'.
This macro is kept for backwards compatibility with old (1.4.x) user code that uses the FL_OVERRIDE macro.
We recommend to use 'override' explicitly if C++11 or higher is available.

14.21 fl_attr.h

Go to the documentation of this file.

```
00001 /*
00002  * Function attribute declarations for the Fast Light Tool Kit (FLTK).
00003  *
00004  * Copyright 1998-2025 by Bill Spitzak and others.
00005  *
00006  * This library is free software. Distribution and use rights are outlined in
00007  * the file "COPYING" which should have been included with this file.  If this
00008  * file is missing or damaged, see the license at:
00009  *
00010  *     https://www.fltk.org/COPYING.php
00011  *
00012  * Please see the following page on how to report bugs and issues:
00013  *
00014  *     https://www.fltk.org/bugs.php
00015  */
00016
00022 #ifndef _FL_fl_attr_h_
00023 #define _FL_fl_attr_h_
00024
00025
00038 #define FL_OVERRIDE override
00039
00040
00046 #ifdef FL_DOXYGEN
00047
00048
00055 #define __fl_attr(x)
00056
00057
00067 #define FL_DEPRECATED(msg, func) \
00068   /##*##* \deprecated msg *##/ \
00069   func
00070
00071
00072 #else /* FL_DOXYGEN */
00073
00074 // If FL_NO_DEPRECATED is defined FLTK 1.4 can compile 1.3.x code without
00075 // issuing several "deprecated" warnings (1.3 "compatibility" mode).
00076 // FL_DEPRECATED will be defined as a no-op.
00077
00078 // If FL_NO_DEPRECATED is not defined (default) FLTK 1.4 will issue several
00079 // "deprecated" warnings depending on the compiler in use: FL_DEPRECATED
00080 // will be defined according to the capabilities of the compiler (below).
00081 // The definition below this comment must match the one at the end of this file.
00082
00083 #if defined(FL_NO_DEPRECATED)
00084 #define FL_DEPRECATED(msg, func) func
00085 #endif
00086
00087 #ifdef __cplusplus
00088
00089 /*
00090   Declare macros specific to Visual Studio.
00091
00092   Visual Studio defines __cplusplus = '199711L' in all its versions which is
00093   not helpful for us here. For VS version number encoding see:
00094   https://learn.microsoft.com/en-us/cpp/preprocessor/predefined-macros
00095
00096   This document specifies that the macro _MSVC_LANG is defined since
00097   "Visual Studio 2015 Update 3" as 201402L (default) and undefined in
00098   earlier versions. It can be used to determine the C++ standard as
00099   specified by the /std:c++ compiler option:
00100
00101   - /std:c++14      201402L  (also if /std:c++ is not used)
00102   - /std:c++17      201703L
00103   - /std:c++20      202002L
00104   - /std:c++latest  a "higher, unspecified value" (docs of VS 2022)
00105
00106   As of this writing (02/2023) _MSVC_LANG is not yet used in this file
```

```
 but it is documented for future use.
*/

#if defined(_MSC_VER)

#if (_MSC_VER >= 1900) // Visual Studio 2015 (14.0)
#endif // Visual Studio 2015 (14.0)

#if (_MSC_VER >= 1400) // Visual Studio 2005 (8.0)
#ifndef FL_DEPRECATED
#define FL_DEPRECATED(msg, func) __declspec(deprecated(msg)) func
#endif
#endif // Visual Studio 2005 (8.0)

#if (_MSC_VER >= 1310) // Visual Studio .NET 2003 (7.1)
#ifndef FL_DEPRECATED
#define FL_DEPRECATED(msg, func) __declspec(deprecated) func
#endif
#endif // Visual Studio .NET 2003 (7.1)

#endif // Visual Studio


/*
 Declare macros specific to the C++ standard used.

 Macros may have been declared already in previous sections.
 */
#if (__cplusplus >= 202002L) // C++20
#endif // C++20

#if (__cplusplus >= 201703L) // C++17
#endif // C++17

#if (__cplusplus >= 201402L) // C++14
#ifndef FL_DEPRECATED
#define FL_DEPRECATED(msg, func) [[deprecated(msg)]] func
#endif
#endif // C++14

#if (__cplusplus >= 201103L) // C++11
#endif // C+11

#if (__cplusplus >= 199711L) // C++89
#endif // C++89

#endif // __cplusplus

/*
 Declare macros specific to clang

 Macros may have been declared already in previous sections.
 */
#if defined(__clang__)

#define FL_CLANG_VERSION (__clang_major__ * 10000 + __clang_minor__ * 100 + __clang_patchlevel__)

// -- nothing yet --

#endif /* __clang__ */


/*
 Declare macros specific to gcc.

 Macros may have been declared already in previous sections.
 */
#if defined(__GNUC__)

#define FL_GCC_VERSION (__GNUC__ * 10000 + __GNUC_MINOR__ * 100 + __GNUC_PATCHLEVEL__)

#ifndef __fl_attr
#define __fl_attr(x) __attribute__ (x)
#endif

#if FL_GCC_VERSION > 40500 /* gcc 4.5.0 */
#ifndef FL_DEPRECATED
#define FL_DEPRECATED(msg, func) func __attribute__((deprecated(msg)))
#endif
#endif /* gcc 4.5.0 */

#if FL_GCC_VERSION >= 30400 /* gcc 3.4.0 */
#ifndef FL_DEPRECATED
#define FL_DEPRECATED(msg, func) func __attribute__((deprecated))
#endif
#endif /* gcc 3.4.0 */

```

```
#endif /* __GNUC__ */


/*
 If a macro was not defined in any of the sections above, set it to no-op here.
 */

#ifndef __fl_attr
#define __fl_attr(x)
#endif

#ifndef FL_DEPRECATED
#define FL_DEPRECATED(msg, func) func
#endif


#endif /* FL_DOXYGEN */


#endif /* !_FL_fl_attr_h_ */
```

14.22 Fl_Bitmap.H

```
//
// Bitmap header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2017 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Bitmap widget . */

#ifndef Fl_Bitmap_H
#define Fl_Bitmap_H
#include "Fl_Image.H"
#include "Fl_Widget.H" // for fl_uintptr_t

class Fl_Widget;
struct Fl_Menu_Item;

class FL_EXPORT Fl_Bitmap : public Fl_Image {
  friend class Fl_Graphics_Driver;
public:
  const uchar *array;
  int alloc_array;

private:
  fl_uintptr_t id_;
  int cache_w_, cache_h_; // size of bitmap when cached

public:
  Fl_Bitmap(const uchar *bits, int W, int H) :
    Fl_Image(W,H,0), array(bits), alloc_array(0), id_(0), cache_w_(0),cache_h_(0) {data((const char **)&array, 1);}
  Fl_Bitmap(const char *bits, int W, int H) :
    Fl_Image(W,H,0), array((const uchar *)bits), alloc_array(0), id_(0), cache_w_(0),cache_h_(0) {data((const char **)&array, 1);}
  Fl_Bitmap(const uchar *bits, int bits_length, int W, int H);
  Fl_Bitmap(const char *bits, int bits_length, int W, int H);
  virtual ~Fl_Bitmap();
  Fl_Image *copy(int W, int H) const override;
  Fl_Image *copy() const { return Fl_Image::copy(); }
  void draw(int X, int Y, int W, int H, int cx=0, int cy=0) override;
  void draw(int X, int Y) {draw(X, Y, w(), h(), 0, 0);}
  void label(Fl_Widget*w) override;
  void label(Fl_Menu_Item*m) override;
  void uncache() override;
  int cache_w() const { return cache_w_;}
  int cache_h() const { return cache_h_;}
};

#endif
```

14.23 Fl_BMP_Image.H

```
//
// BMP image header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2022 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_BMP_Image widget . */

#ifndef Fl_BMP_Image_H
#define Fl_BMP_Image_H
#  include "Fl_Image.H"

class FL_EXPORT Fl_BMP_Image : public Fl_RGB_Image {

public:

  Fl_BMP_Image(const char* filename);
  Fl_BMP_Image(const char* imagename, const unsigned char *data, const long length = -1);

protected:

  void load_bmp_(class Fl_Image_Reader &rdr, int ico_height = 0, int ico_width = 0);

};

#endif
```

14.24 Fl_Box.H File Reference

Fl_Box widget.

`#include "Fl_Widget.H"`

Classes

- class Fl_Box

  This widget simply draws its box, and possibly its label.

14.24.1 Detailed Description

Fl_Box widget.

14.25 Fl_Box.H

Go to the documentation of this file.

```
//
// Box header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2025 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef Fl_Box_H
```

```
#define Fl_Box_H

#ifndef Fl_Widget_H
#include "Fl_Widget.H"
#endif

class FL_EXPORT Fl_Box : public Fl_Widget {
protected:
  void draw() override;
public:
  Fl_Box(int X, int Y, int W, int H, const char *L = 0);

  Fl_Box(Fl_Boxtype B, int X, int Y, int W, int H, const char *L);

  int handle(int) override;
};

#endif
```

14.26 Fl_Browser.H

```
//
// Browser header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2026 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Browser widget . */

// Forms-compatible browser.  Probably useful for other
// lists of textual data.  Notice that the line numbers
// start from 1, and 0 means "no line".

#ifndef Fl_Browser_H
#define Fl_Browser_H

#include "Fl_Browser_.H"
#include "Fl_Image.H"

struct FL_BLINE;

class FL_EXPORT Fl_Browser : public Fl_Browser_ {

  FL_BLINE *first;              // the array of lines
  FL_BLINE *last;
  FL_BLINE *cache;
  int cacheline;                // line number of cache
  int lines;                    // Number of lines
  int full_height_;
  const int* column_widths_;
  char format_char_;            // alternative to @-sign
  char column_char_;            // alternative to tab

protected:

  static constexpr char BLINE_SELECTED = 1;
  static constexpr char BLINE_NOTDISPLAYED = 2;

  // required routines for Fl_Browser_ subclass:
  void* item_first() const override;
  void* item_next(void* item) const override;
  void* item_prev(void* item) const override;
  void* item_last()const override;
  int item_selected(void* item) const override;
  void item_select(void* item, int val) override;
  int item_height(void* item) const override;
  int item_width(void* item) const override;
  void item_draw(void* item, int X, int Y, int W, int H) const override;
  int full_height() const override;
  int incr_height() const override;
  const char *item_text(void *item) const override;
  void item_swap(void *a, void *b) override { swap((FL_BLINE*)a, (FL_BLINE*)b); }
  void *item_at(int line) const override { return (void*)find_line(line); }

```

```
  FL_BLINE* find_line(int line) const ;
  FL_BLINE* _remove(int line) ;
  void insert(int line, FL_BLINE* item);
  int lineno(void *item) const ;
  void swap(FL_BLINE *a, FL_BLINE *b);

  void*& bline_data(FL_BLINE* b) const;
  const void* bline_data(const FL_BLINE* b) const;
  char& bline_flags(FL_BLINE* b) const;
  char bline_flags(const FL_BLINE* b) const;
  char* bline_txt(FL_BLINE* b) const;
  const char* bline_txt(const FL_BLINE* b) const;
  Fl_Image*& bline_icon(FL_BLINE* b) const;
  const Fl_Image* bline_icon(const FL_BLINE* b) const;
  short bline_length(const FL_BLINE* b) const;

public:

  void remove(int line);
  void add(const char* newtext, void* d = 0);
  void insert(int line, const char* newtext, void* d = 0);
  void move(int to, int from);
  int  load(const char* filename);
  void swap(int a, int b);
  void clear();

  int size() const { return lines; }
  void size(int W, int H) { Fl_Widget::size(W, H); }

  Fl_Fontsize textsize() const { return Fl_Browser_::textsize(); }

  /*
    Sets the default text size for the lines in the browser to newSize.
    Defined and documented in Fl_Browser.cxx
  */
  void textsize(Fl_Fontsize newSize);

  int topline() const ;
  enum Fl_Line_Position { TOP, BOTTOM, MIDDLE };
  void lineposition(int line, Fl_Line_Position pos);
  void topline(int line) { lineposition(line, TOP); }
  void bottomline(int line) { lineposition(line, BOTTOM); }
  void middleline(int line) { lineposition(line, MIDDLE); }

  int select(int line, int val=1);
  int selected(int line) const ;
  void show(int line);
  void show() override { Fl_Widget::show(); }
  void hide(int line);
  void hide() override { Fl_Widget::hide(); }
  int visible(int line) const ;

  int value() const ;
  void value(int line) { select(line); }
  const char* text(int line) const ;
  void text(int line, const char* newtext);
  void* data(int line) const ;
  void data(int line, void* d);

  Fl_Browser(int X, int Y, int W, int H, const char *L = 0);
  ~Fl_Browser() { clear(); }

  char format_char() const { return format_char_; }
  void format_char(char c) { format_char_ = c; }
  char column_char() const { return column_char_; }
  void column_char(char c) { column_char_ = c; }
  const int* column_widths() const { return column_widths_; }
  void column_widths(const int* arr) { column_widths_ = arr; }

  int displayed(int line) const { return Fl_Browser_::displayed(find_line(line)); }

  void make_visible(int line) {
    if (line < 1) Fl_Browser_::display(find_line(1));
    else if (line > lines) Fl_Browser_::display(find_line(lines));
    else Fl_Browser_::display(find_line(line));
  }

  // icon support
  void icon(int line, Fl_Image* icon);
  Fl_Image* icon(int line) const;
  void remove_icon(int line);

  void replace(int a, const char* b) { text(a, b); }
  void display(int line, int val=1);
};

#endif
```

14.27 Fl_Browser_.H

```
//
// Common browser header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2026 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Browser_ widget . */

// Yes, I know this should be a template...

#ifndef Fl_Browser__H
#define Fl_Browser__H

#ifndef Fl_Group_H
#include "Fl_Group.H"
#endif
#include "Fl_Scrollbar.H"
#include <FL/Fl.H>              // Fl::scrollbar_size()

#define FL_NORMAL_BROWSER       0
#define FL_SELECT_BROWSER       1
#define FL_HOLD_BROWSER         2
#define FL_MULTI_BROWSER        3
#define FL_SORT_ASCENDING       0
#define FL_SORT_DESCENDING      1
#define FL_SORT_CASEINSENSITIVE 0x2
class FL_EXPORT Fl_Browser_ : public Fl_Group {
  int position_;        // where user wants it scrolled to
  int real_position_;   // the current vertical scrolling position
  int hposition_;       // where user wants it panned to
  int real_hposition_;  // the current horizontal scrolling position
  int offset_;          // how far down top_ item the real_position is
  int max_width;        // widest object seen so far
  uchar has_scrollbar_; // which scrollbars are enabled
  Fl_Font textfont_;
  Fl_Fontsize textsize_;
  Fl_Color textcolor_;
  void* top_;           // which item scrolling position is in
  void* selection_;     // which is selected (except for FL_MULTI_BROWSER)
  void *redraw1,*redraw2; // minimal update pointers
  void* max_width_item; // which item has max_width_
  int scrollbar_size_;  // size of scrollbar trough
  int linespacing_;

  void update_top();

protected:

  // All of the following must be supplied by the subclass:
  virtual void *item_first() const = 0;
  virtual void *item_next(void *item) const = 0;
  virtual void *item_prev(void *item) const = 0;
  virtual void *item_last() const { return 0L; }
  virtual int item_height(void *item) const = 0;
  virtual int item_width(void *item) const = 0;
  virtual int item_quick_height(void *item) const ;
  virtual void item_draw(void *item,int X,int Y,int W,int H) const = 0;
  virtual const char *item_text(void *item) const { (void)item; return 0L; }
  virtual void item_swap(void *a,void *b) { (void)a; (void)b; }
  virtual void *item_at(int index) const { (void)index; return 0L; }
  // you don't have to provide these but it may help speed it up:
  virtual int full_width() const ;      // current width of all items
  virtual int full_height() const ;     // current height of all items
  virtual int incr_height() const ;     // average height of an item
  // These only need to be done by subclass if you want a multi-browser:
  virtual void item_select(void *item,int val=1);
  virtual int item_selected(void *item) const ;

  // things the subclass may want to call:
  void *top() const { return top_; }
  void *selection() const { return selection_; }
  void new_list(); // completely clobber all data, as though list replaced
  void deleting(void *item); // get rid of any pointers to item
  void replacing(void *a,void *b); // change a pointers to b
```

```
  void swapping(void *a,void *b); // exchange pointers a and b
  void inserting(void *a,void *b); // insert b near a
  int displayed(void *item) const ; // true if this item is visible
  void redraw_line(void *item); // minimal update, no change in size
  void redraw_lines() { damage(FL_DAMAGE_SCROLL); } // redraw all of them
  void bbox(int &X,int &Y,int &W,int &H) const;
  int leftedge() const; // x position after scrollbar & border
  void *find_item(int ypos); // item under mouse

  void draw() override;
  Fl_Browser_(int X,int Y,int W,int H,const char *L=0);

public:

  Fl_Scrollbar scrollbar;
  Fl_Scrollbar hscrollbar;

  int handle(int event) override;
  void resize(int X,int Y,int W,int H) override;

  int select(void *item,int val=1,int docallbacks=0);
  int select_only(void *item,int docallbacks=0);
  int deselect(int docallbacks=0);
  int vposition() const { return position_; }
  FL_DEPRECATED("since 1.4.0 - use vposition() instead",
  int position() const) { return vposition(); }

  void vposition(int pos); // scroll to here
  FL_DEPRECATED("since 1.4.0 - use vposition(pos) instead",
  void position(int pos)) { return vposition(pos); }
  void position(int x, int y) { Fl_Group::position(x, y); }

  int hposition() const { return hposition_; }
  void hposition(int); // pan to here
  void display(void *item); // scroll so this item is shown

  enum { // values for has_scrollbar()
    HORIZONTAL = 1,
    VERTICAL = 2,
    BOTH = 3,
    ALWAYS_ON = 4,
    HORIZONTAL_ALWAYS = 5,
    VERTICAL_ALWAYS = 6,
    BOTH_ALWAYS = 7
  };
  uchar has_scrollbar() const { return has_scrollbar_; }
  void has_scrollbar(uchar mode) { has_scrollbar_ = mode; }

  Fl_Font textfont() const { return textfont_; }
  void textfont(Fl_Font font) { textfont_ = font; }

  Fl_Fontsize textsize() const { return textsize_; }
  void textsize(Fl_Fontsize newSize) { textsize_ = newSize; }

  Fl_Color textcolor() const { return textcolor_; }
  void textcolor(Fl_Color col) { textcolor_ = col; }

  int scrollbar_size() const {
      return(scrollbar_size_);
  }

  void scrollbar_size(int newSize) {
      scrollbar_size_ = newSize;
  }

  void scrollbar_right() { scrollbar.align(FL_ALIGN_RIGHT); }
  void scrollbar_left() { scrollbar.align(FL_ALIGN_LEFT); }
  void sort(int flags=0);

  void linespacing(int pixels) { linespacing_ = pixels; }

  int linespacing() const { return linespacing_; }
};

#endif
```

14.28 Fl_Button.H

```
//
// Button header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2026 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
```

```
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Button widget . */

#ifndef Fl_Button_H
#define Fl_Button_H

#ifndef Fl_Widget_H
#include "Fl_Widget.H"
#endif

// values for type()
#define FL_NORMAL_BUTTON        0
#define FL_TOGGLE_BUTTON        1
#define FL_RADIO_BUTTON         (FL_RESERVED_TYPE+2)
#define FL_HIDDEN_BUTTON        3


class Fl_Widget_Tracker;

class FL_EXPORT Fl_Button : public Fl_Widget {

  int shortcut_;
  char value_;
  char oldval;
  uchar down_box_;
  uchar compact_;

protected:

  static Fl_Widget_Tracker *key_release_tracker;
  static void key_release_timeout(void*);
  void simulate_key_action();

  void draw() override;

public:

  int handle(int) override;

  Fl_Button(int X, int Y, int W, int H, const char *L = 0);

  int value(int v);

  char value() const {return value_;}

  int set() {return value(1);}

  int clear() {return value(0);}

  void setonly(); // this should only be called on FL_RADIO_BUTTONs

  int shortcut() const {return shortcut_;}

  void shortcut(int s) {shortcut_ = s;}

  Fl_Boxtype down_box() const {return (Fl_Boxtype)down_box_;}

  void down_box(Fl_Boxtype b) {down_box_ = (uchar)b;}

  void shortcut(const char *s);

  Fl_Color down_color() const {return selection_color();}

  void down_color(unsigned c) {selection_color(c);}

  // handle flag for compact buttons, documentation in source code
  void compact(uchar v);

  uchar compact() const { return compact_; }
};

#endif
```

14.29 Fl_Cairo.H File Reference

Cairo is currently supported for the following platforms: Windows, macOS, Unix/Linux (X11 + Wayland).

```
#include <FL/Fl.H>
#include <cairo.h>
```

Classes

- class Fl_Cairo_State

  Contains all the necessary info on the current cairo context.

14.29.1 Detailed Description

Cairo is currently supported for the following platforms: Windows, macOS, Unix/Linux (X11 + Wayland).

Note

In FLTK 1.3.x this header file (Fl_Cairo.H) included the platform specific Cairo headers. This is no longer true since 1.4.0.

This header file is platform agnostic. If you need platform specific Cairo headers you need to #include them in your source file.
To use FLTK's builtin Cairo support you need to `#include <FL/Fl.H>` **before** you include any other FLTK header which is officially required anyway. Since FLTK 1.4.0 the preprocessor constants `FLTK_HAVE_CAIRO` and/or `FLTK_HAVE_CAIROEXT` are defined in <FL/Fl.H> by including <FL/fl_config.h>.

14.30 Fl_Cairo.H

Go to the documentation of this file.

```
00001 //
00002 // Main Cairo support header file for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 1998-2025 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file.  If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 //     https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 //     https://www.fltk.org/bugs.php
00015 //
00016
00033 #ifndef FL_CAIRO_H
00034 #define FL_CAIRO_H
00035
00036 #include <FL/Fl.H>
00037
00038 # ifdef FLTK_HAVE_CAIRO
00039
00040 # include <cairo.h>
00041
00055 class FL_EXPORT Fl_Cairo_State {
00056 public:
00057   Fl_Cairo_State()
00058     : cc_(0)
00059     , own_cc_(false)
00060     , autolink_(false)
00061     , window_(0)
00062     , gc_(0) {}
00063
00064   // access attributes
00065   cairo_t *cc() const { return cc_; }
00066   bool autolink() const { return autolink_; }
00075   void cc(cairo_t *c, bool own = true) {
00076     if (cc_ && own_cc_)
00077       cairo_destroy(cc_);
00078     cc_ = c;
00079     if (!cc_)
00080       window_ = 0;
```

```
    own_cc_ = own;
  }
  void autolink(bool b);
  void window(void *w) { window_ = w; }
  void *window() const { return window_; }
  void gc(void *c) { gc_ = c; }
  void *gc() const { return gc_; }

private:
  cairo_t *cc_;        // contains the unique autoupdated cairo context
  bool own_cc_;        // indicates whether we must delete the cc, useful for internal cleanup
  bool autolink_;      // false by default, prevents the automatic cairo mapping on fltk windows
                       // for custom cairo implementations.
  void *window_, *gc_; // for keeping track internally of last win+gc treated
};

#endif // FLTK_HAVE_CAIRO
#endif // FL_CAIRO_H
```

14.31 Fl_Cairo_Window.H File Reference

Fl_Cairo_Window, an FLTK window incorporating a Cairo draw callback.

```
#include <FL/fl_config.h>
#include <FL/Fl.H>
#include <FL/Fl_Double_Window.H>
```

Classes

- class Fl_Cairo_Window

  This defines an FLTK window with Cairo support.

14.31.1 Detailed Description

Fl_Cairo_Window, an FLTK window incorporating a Cairo draw callback.

14.32 Fl_Cairo_Window.H

Go to the documentation of this file.

```
//
// Fl_Cairo_Window header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2025 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef FL_CAIRO_WINDOW_H
# define FL_CAIRO_WINDOW_H

#include <FL/fl_config.h>

# ifdef FLTK_HAVE_CAIRO

// Cairo is currently supported for the following platforms:
// Win32, Apple Quartz, X11, Wayland

#  include <FL/Fl.H>
#  include <FL/Fl_Double_Window.H>

class FL_EXPORT Fl_Cairo_Window : public Fl_Double_Window {

public:
  Fl_Cairo_Window(int W, int H, const char *L = 0)
    : Fl_Double_Window(W, H, L), draw_cb_(0) {}
  Fl_Cairo_Window(int X, int Y, int W, int H, const char *L = 0)
```

```
    : Fl_Double_Window(X, Y, W, H, L), draw_cb_(0) {}

protected:
  void draw() override {
    Fl_Double_Window::draw();
    if (draw_cb_) { // call the Cairo draw callback
      // manual method ? if yes explicitly get a cairo_context here
      if (!Fl::cairo_autolink_context())
        Fl::cairo_make_current(this);
      draw_cb_(this, Fl::cairo_cc());
      // flush Cairo drawings: necessary at least for Windows
      Fl::cairo_flush(Fl::cairo_cc());
    }
  }

public:
  typedef void (*cairo_draw_cb) (Fl_Cairo_Window* self, cairo_t* def);

  void set_draw_cb(cairo_draw_cb cb) { draw_cb_ = cb; }
private:
    cairo_draw_cb draw_cb_;
};


# endif // FLTK_HAVE_CAIRO
#endif // FL_CAIRO_WINDOW_H
```

14.33 fl_callback_macros.H File Reference

This file provides macros for easy function and method callbacks with multiple type safe arguments.

`#include <stdlib.h>`

Macros

- #define FL_FUNCTION_CALLBACK_3(WIDGET, FUNC, TYPE0, VALUE0, TYPE1, VALUE1, TYPE2, VALUE2)

  Declare a C function callback with custom parameters.
- #define FL_INLINE_CALLBACK_2(WIDGET, TYPE0, NAME0, VALUE0, TYPE1, NAME1, VALUE1, LAMBDA)

  Creates code to declare a callback function in line with instantiating a widget.
- #define FL_METHOD_CALLBACK_1(WIDGET, CLASS, SELF, METH, TYPE0, VALUE0)

  Declare a non-static class method callback with custom parameters.

14.33.1 Detailed Description

This file provides macros for easy function and method callbacks with multiple type safe arguments.

14.33.2 Macro Definition Documentation

14.33.2.1 FL_FUNCTION_CALLBACK_3

```
#define FL_FUNCTION_CALLBACK_3(
            WIDGET,
            FUNC,
            TYPE0,
            VALUE0,
            TYPE1,
            VALUE1,
            TYPE2,
            VALUE2 )
```

Declare a C function callback with custom parameters.

You can declare a plain C function callback or a static method callback with custom parameters using this macro. It simplifies the process of calling arbitrary functions with up to five custom parameters. The macro generates code that ensures type safety and expands FLTK's standard callbacks, which are limited to a single `void*` or `long` argument.

To use the macro, you provide the widget that will handle the callback as the first argument. The second argument can be either a regular function or a static method in any class.
Following these arguments, you can include up to five pairs, where each pair consists of a type and a value. For example, `int, 3` specifies an integer parameter with a value of 3. If you need to pass two arguments, you can use two pairs, like this: `int, 3, int, 4`. The last digit of the macro name must be the same as the number of pairs (0..5)
Whenever the code generated by the macro is called, the custom parameters are duplicated and marked for automatic deallocation using `delete` when the callback widget is destroyed.

```
#include <FL/fl_callback_macros.H>
...
Fl_Button *btn1 = new Fl_Button(10, 10, 100, 20, "Beep");
FL_FUNCTION_CALLBACK_0(btn1, fl_beep);
...
Fl_Button *btn2 = new Fl_Button(10, 40, 100, 20, "Hello");
FL_FUNCTION_CALLBACK_5(btn2,
  fl_message,
  const char *, "Hello\n%d %d %d %d",
  int, 1,  int, 2,  int, 3,  int, 4
);
```

You can find a small demonstration program showcasing the usage of `FL_*_CALLBACK_*` in the `examples/callbacks.cxx` file.

Parameters

WIDGET	the widget that will call the callback
FUNC	a C/C++ function or a static class method
TYPE0,VALUE0,TYPE1,VALUE1,TYPE2,VALUE2	a list of zero to five type/value pairs, all separated by commas

See also

FL_METHOD_CALLBACK_1, FL_INLINE_CALLBACK_2

14.33.2.2 FL_INLINE_CALLBACK_2

```
#define FL_INLINE_CALLBACK_2(
              WIDGET,
              TYPE0,
              NAME0,
              VALUE0,
              TYPE1,
              NAME1,
              VALUE1,
              LAMBDA )
```

Creates code to declare a callback function in line with instantiating a widget.
You can use this macro to create a function as a callback, allowing you to define the callback function right where the widget and callback are declared, similar to a Lambda function.
The first argument of the macro specifies the widget that will handle the callback. Next, you can include up to five triplets, where each triplet consists of a type, a parameter name, and a value. For example, `int, x, 3` specifies an integer parameter with a value of 3. If you need to pass two arguments, you can use two triplets, such as `int, x, 3, int, y, 4`. The last digit of the macro name must be the same as the number of triplets (0..5).
The last argument is the actual function body itself.
The function body is limited to a syntax that the macro preprocessor can handle. It should include the leading '{' and trailing '}' and may contain local variable declarations, use global variables and functions, and use also the variables listed and initialized in the argument triples of the macro. Very large function bodies should be avoided because they may exceed the admissible size of a macro argument.
Whenever the code generated by the macro is called, the custom parameters are duplicated and marked for automatic deallocation using `delete` when the callback widget is destroyed.

```
#include <FL/fl_callback_macros.H>
...
Fl_Button *btn = new Fl_Button(10, 10, 100, 20, "Test");
FL_INLINE_CALLBACK_1(btn,
  const char *, name, btn->label(),
  {
```

```
    fl_message("Greetings from the %s button", name);
  }
);
```

You can find a small demonstration program showcasing the usage of `FL_*_CALLBACK_*` in the `examples/callbacks.cxx` file.

Parameters

WIDGET	the widget that will call the callback
TYPE0	the type of the first parameter in the function call
NAME0	an arbitrary variable name that can be used as a parameter in the function body
VALUE0	a constant value or a variable; the value of the variable is copied when the callback is created
TYPE1,NAME1,VALUE1	as above; there are six macros that support 0 to 5 parameters
LAMBDA	the function body within the limits of the C macro preprocessor

See also

FL_METHOD_CALLBACK_1, FL_FUNCTION_CALLBACK_3

14.33.2.3 FL_METHOD_CALLBACK_1

```
#define FL_METHOD_CALLBACK_1(
              WIDGET,
              CLASS,
              SELF,
              METH,
              TYPE0,
              VALUE0 )
```

Declare a non-static class method callback with custom parameters.

You can declare a callback for a non-static class method with custom parameters using this macro. It provides a convenient way to call arbitrary methods in any class, overcoming FLTK's limitation of passing only a single `void*` or `long` argument. Furthermore, it ensures type safety.

The first argument of the macro specifies the widget that will handle the callback. The second argument indicates the class type to be called. The third argument must be a pointer to an instance of that class. The fourth argument is the name of the method within the class. That method must be public and should not be static.

Following these arguments, you can include up to five pairs, where each pair consists of a type and a value. For example, `int, 3` specifies an integer parameter with a value of 3. If you need to pass two arguments, you can use two pairs, like this: `int, 3, int, 4`. The last digit of the macro name must be the same as the number of pairs (0..5)

Whenever the code generated by the macro is called, the custom parameters are duplicated and marked for automatic deallocation using `delete` when the callback widget is destroyed.

```
#include <FL/fl_callback_macros.H>
...
Fl_Button *btn = new Fl_Button(10, 10, 100, 20, "Test");
FL_METHOD_CALLBACK_1(btn, Fl_Button, btn, color, Fl_Color, FL_GREEN);
```

You can find a small demonstration program showcasing the usage of `FL_*_CALLBACK_*` in the `examples/callbacks.cxx` file.

Parameters

WIDGET	the widget that will call the callback
CLASS	the class type
SELF	a pointer to an instance of the class
METH	a C++ class method that must be public and not static
TYPE0,VALUE0	a list of zero to five type/value pairs, all separated by commas

See also

FL_FUNCTION_CALLBACK_3, FL_INLINE_CALLBACK_2

14.34 fl_callback_macros.H

Go to the documentation of this file.

```
/*
 * Macros for easy callbacks for the Fast Light Tool Kit (FLTK).
 *
 * Copyright 2023-2025 by Bill Spitzak and others.
 *
 * This library is free software. Distribution and use rights are outlined in
 * the file "COPYING" which should have been included with this file.  If this
 * file is missing or damaged, see the license at:
 *
 *     https://www.fltk.org/COPYING.php
 *
 * Please see the following page on how to report bugs and issues:
 *
 *     https://www.fltk.org/bugs.php
 */

#ifndef _FL_FL_CALLBACK_MACROS_H_
#define _FL_FL_CALLBACK_MACROS_H_

#include <stdlib.h>

#ifdef FL_DOXYGEN

#define FL_FUNCTION_CALLBACK_3(WIDGET, FUNC, TYPE0, VALUE0, TYPE1, VALUE1, TYPE2, VALUE2)

#define FL_METHOD_CALLBACK_1(WIDGET, CLASS, SELF, METH, TYPE0, VALUE0)

#define FL_INLINE_CALLBACK_2(WIDGET, TYPE0, NAME0, VALUE0, TYPE1, NAME1, VALUE1, LAMBDA)

#else // FL_DOXYGEN

/*
 These two macros make it possible to call macros with names that are created
 by concatenating the name in x and (in this context) the number in y.
 */
#define _FL_CBD_CONCAT_IMPL(x, y) x##y
#define _FL_CBD_CONCAT(x, y) _FL_CBD_CONCAT_IMPL(x, y)

/*
 Create a unique name for the derived class based on the current source code
 line number.
 */
#define _FL_CBD_CLASS_NAME _FL_CBD_CONCAT(Fl_Callback_User_Data_,__LINE__)


/*
 These macros create boilerplate code for callbacks to functions and
  static class methods with up to five arguments.

  This macro invocation for example
  ```
  FL_FUNCTION_CALLBACK_2(func_cb_btn_2, hello_2_args_cb,
                          const char *, text, "FLTK",
                          int, number, 2);
  ```
  will generate the following code:

  ```
  do {
    class Fl_Callback_User_Data_92 : public Fl_Callback_User_Data {
      public:
        const char * p0_;
        int p1_;
        static void cb(Fl_Widget *w, void *user_data) {
          Fl_Callback_User_Data_92 *d = (Fl_Callback_User_Data_92*)user_data;
          hello_2_args_cb(d->p0_, d->p1_);
        }
        Fl_Callback_User_Data_92(const char * p0, int p1)
        : p0_(p0),
          p1_(p1)
        { }
    };
    func_cb_btn_2->callback(Fl_Callback_User_Data_92::cb,
                            new Fl_Callback_User_Data_92("FLTK", 2),
                            true);
  } while(0)
  ```
```

```

  Clicking the Fl_Button `func_cb_btn_2` will call `hello_2_args_cb("FLTK", 2)`.
  Deleting the button will also delete the data that was created in our
  boilerplate code.
 */
#define FL_FUNCTION_CALLBACK_5(WIDGET, FUNC, TYPE0, VALUE0, TYPE1, VALUE1, TYPE2, VALUE2, TYPE3, VALUE3, TYPE4, VALUE4) \
  do { \
    class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data { \
    public: \
      TYPE0 p0_; TYPE1 p1_; TYPE2 p2_; TYPE3 p3_; TYPE4 p4_; \
      static void cb(Fl_Widget *w, void *user_data) { \
        _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data; \
        FUNC(d->p0_, d->p1_, d->p2_, d->p3_, d->p4_); \
      } \
      _FL_CBD_CLASS_NAME(TYPE0 p0, TYPE1 p1, TYPE2 p2, TYPE3 p3, TYPE4 p4) \
      : p0_(p0), p1_(p1), p2_(p2), p3_(p3), p4_(p4) { } \
    }; \
    WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(VALUE0, VALUE1, VALUE2, VALUE3, VALUE4), true); \
  } while(0)

#define FL_FUNCTION_CALLBACK_4(WIDGET, FUNC, TYPE0, VALUE0, TYPE1, VALUE1, TYPE2, VALUE2, TYPE3, VALUE3) \
  do { \
    class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data { \
    public: \
      TYPE0 p0_; TYPE1 p1_; TYPE2 p2_; TYPE3 p3_; \
      static void cb(Fl_Widget *w, void *user_data) { \
        _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data; \
        FUNC(d->p0_, d->p1_, d->p2_, d->p3_); \
      } \
      _FL_CBD_CLASS_NAME(TYPE0 p0, TYPE1 p1, TYPE2 p2, TYPE3 p3) \
      : p0_(p0), p1_(p1), p2_(p2), p3_(p3) { } \
    }; \
    WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(VALUE0, VALUE1, VALUE2, VALUE3), true); \
  } while(0)

#define FL_FUNCTION_CALLBACK_3(WIDGET, FUNC, TYPE0, VALUE0, TYPE1, VALUE1, TYPE2, VALUE2) \
  do { \
    class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data { \
    public: \
      TYPE0 p0_; TYPE1 p1_; TYPE2 p2_; \
      static void cb(Fl_Widget *w, void *user_data) { \
        _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data; \
        FUNC(d->p0_, d->p1_, d->p2_); \
      } \
      _FL_CBD_CLASS_NAME(TYPE0 p0, TYPE1 p1, TYPE2 p2) \
      : p0_(p0), p1_(p1), p2_(p2) { } \
    }; \
    WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(VALUE0, VALUE1, VALUE2), true); \
  } while(0)

#define FL_FUNCTION_CALLBACK_2(WIDGET, FUNC, TYPE0, VALUE0, TYPE1, VALUE1) \
  do { \
    class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data { \
    public: \
      TYPE0 p0_; TYPE1 p1_; \
      static void cb(Fl_Widget *w, void *user_data) { \
        _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data; \
        FUNC(d->p0_, d->p1_); \
      } \
      _FL_CBD_CLASS_NAME(TYPE0 p0, TYPE1 p1) \
      : p0_(p0), p1_(p1) { } \
    }; \
    WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(VALUE0, VALUE1), true); \
  } while(0)

#define FL_FUNCTION_CALLBACK_1(WIDGET, FUNC, TYPE0, VALUE0) \
  do { \
    class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data { \
    public: \
      TYPE0 p0_; \
      static void cb(Fl_Widget *w, void *user_data) { \
        _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data; \
        FUNC(d->p0_); \
      }; \
      _FL_CBD_CLASS_NAME(TYPE0 p0) \
      : p0_(p0) { }; \
    }; \
    WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(VALUE0), true); \
  } while(0)

#define FL_FUNCTION_CALLBACK_0(WIDGET, FUNC) \
  do { \
    class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data { \
```

```
    public: \
      static void cb(Fl_Widget *w, void *user_data) { \
        FUNC(); \
      } \
      _FL_CBD_CLASS_NAME() { } \
    }; \
    WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(), true); \
  } while(0)

/*
 These macros create boilerplate code for callbacks to class methods
  with up to five arguments.

  This macro invocation for example
  ```
  FL_METHOD_CALLBACK_4(btn,
                       MyWindow, win, resize,
                       int, test_x+10,
                       int, test_y+10,
                       int, 320,
                       int, 400);
  ```
  will generate the following code:

  ```
  do {
    class Fl_Callback_User_Data_73 : public Fl_Callback_User_Data {
    public:
      int p0_;
      int p1_;
      int p2_;
      int p3_;
      MyWindow *self_;
      static void cb(Fl_Widget *w, void *user_data) {
        Fl_Callback_User_Data_73 *d = (Fl_Callback_User_Data_73*)user_data;
        d->self_->resize(d->p0_, d->p1_, d->p2_, d->p3_);
      }
      Fl_Callback_User_Data_73(MyWindow *self, int p0, int p1, int p2, int p3)
      : self_(self), p0_(p0), p1_(p1), p2_(p2), p3_(p3) { }
    };
    btn->callback(Fl_Callback_User_Data_73::cb,
                  new Fl_Callback_User_Data_73(win, test_x+10, test_y+10, 320, 400),
                  true);
  } while(0);
  ```

  Clicking the Fl_Button `btn` will call
  `win->resize(test_x+10, test_y+10, 320, 400);`.
  Deleting the button will also delete the data that was created in our
  boilerplate code.
 */

#define FL_METHOD_CALLBACK_5(WIDGET, CLASS, SELF, METHOD, TYPE0, VALUE0, TYPE1, VALUE1, TYPE2, VALUE2, TYPE3, VALUE3, TYPE4, VALUE4) \
  do { \
    class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data { \
    public: \
      CLASS *self_; \
      TYPE0 p0_; TYPE1 p1_; TYPE2 p2_; TYPE3 p3_; TYPE4 p4_; \
      static void cb(Fl_Widget *w, void *user_data) { \
        _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data; \
        d->self_->METHOD(d->p0_, d->p1_, d->p2_, d->p3_, d->p4_); \
      } \
      _FL_CBD_CLASS_NAME(CLASS *self, TYPE0 p0, TYPE1 p1, TYPE2 p2, TYPE3 p3, TYPE4 p4) \
      : self_(self), p0_(p0), p1_(p1), p2_(p2), p3_(p3), p4_(p4) { } \
    }; \
    WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(SELF, VALUE0, VALUE1, VALUE2, VALUE3, VALUE4), true); \
  } while(0)

#define FL_METHOD_CALLBACK_4(WIDGET, CLASS, SELF, METHOD, TYPE0, VALUE0, TYPE1, VALUE1, TYPE2, VALUE2, TYPE3, VALUE3) \
  do { \
    class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data { \
    public: \
      CLASS *self_; \
      TYPE0 p0_; TYPE1 p1_; TYPE2 p2_; TYPE3 p3_; \
      static void cb(Fl_Widget *w, void *user_data) { \
        _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data; \
        d->self_->METHOD(d->p0_, d->p1_, d->p2_, d->p3_); \
      } \
      _FL_CBD_CLASS_NAME(CLASS *self, TYPE0 p0, TYPE1 p1, TYPE2 p2, TYPE3 p3) \
      : self_(self), p0_(p0), p1_(p1), p2_(p2), p3_(p3) { } \
    }; \
    WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(SELF, VALUE0, VALUE1, VALUE2, VALUE3), true); \
  } while(0)
```

```
#define FL_METHOD_CALLBACK_3(WIDGET, CLASS, SELF, METHOD, TYPE0, VALUE0, TYPE1, VALUE1, TYPE2, VALUE2) \
  do { \
    class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data { \
    public: \
      CLASS *self_; \
      TYPE0 p0_; TYPE1 p1_; TYPE2 p2_; \
      static void cb(Fl_Widget *w, void *user_data) { \
        _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data; \
        d->self_->METHOD(d->p0_, d->p1_, d->p2_); \
      } \
      _FL_CBD_CLASS_NAME(CLASS *self, TYPE0 p0, TYPE1 p1, TYPE2 p2) \
      : self_(self), p0_(p0), p1_(p1), p2_(p2) { } \
    }; \
    WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(SELF, VALUE0, VALUE1, VALUE2), true); \
  } while(0)

#define FL_METHOD_CALLBACK_2(WIDGET, CLASS, SELF, METHOD, TYPE0, VALUE0, TYPE1, VALUE1) \
  do { \
    class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data { \
    public: \
      CLASS *self_; \
      TYPE0 p0_; TYPE1 p1_; \
      static void cb(Fl_Widget *w, void *user_data) { \
        _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data; \
        d->self_->METHOD(d->p0_, d->p1_); \
      } \
      _FL_CBD_CLASS_NAME(CLASS *self, TYPE0 p0, TYPE1 p1) \
      : self_(self), p0_(p0), p1_(p1) { } \
    }; \
    WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(SELF, VALUE0, VALUE1), true); \
  } while(0)

#define FL_METHOD_CALLBACK_1(WIDGET, CLASS, SELF, METHOD, TYPE0, VALUE0) \
  do { \
    class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data { \
    public: \
      CLASS *self_; \
      TYPE0 p0_; \
      static void cb(Fl_Widget *w, void *user_data) { \
        _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data; \
        d->self_->METHOD(d->p0_); \
      } \
      _FL_CBD_CLASS_NAME(CLASS *self, TYPE0 p0) \
      : self_(self), p0_(p0) { } \
    }; \
    WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(SELF, VALUE0), true); \
  } while(0)

#define FL_METHOD_CALLBACK_0(WIDGET, CLASS, SELF, METHOD) \
  do { \
    class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data { \
    public: \
      CLASS *self_; \
      static void cb(Fl_Widget *w, void *user_data) { \
        _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data; \
        d->self_->METHOD(); \
      } \
      _FL_CBD_CLASS_NAME(CLASS *self) \
      : self_(self) { } \
    }; \
    WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(SELF), true); \
  } while(0)

/*
  These macros create boilerplate code for callback functions inlined into
  the widget creation code (similar to lambda functions in C++11 and up)
  with up to five arguments.

  This macro invocation for example
  ```
  FL_INLINE_CALLBACK_2(// callback has two parameters
    btn,   // attach callback to this button
    const char *, text, "FLTK",   // first parameter (type, name, value)
    int, number, 2,   // second parameter
      {   // function body
        fl_message("We received the message %s with %d!", text, number);
      }
    );
  ```
  will generate the following code:
  ```
  do {
    class Fl_Callback_User_Data_133 : public Fl_Callback_User_Data {
    public:
```
```

```
      const char * p0_;           // store first parameter here
      int p1_;                    // store second parameter here
      // lambda style function
      static void fn(const char * text, int number ) {
        fl_message("We received the message %s with %d!", text, number);
      }
      // FLTK style callback
      static void cb(Fl_Widget *w, void *user_data) {
        Fl_Callback_User_Data_133 *d = (Fl_Callback_User_Data_133*)user_data;
        fn(d->p0_, d->p1_);
      }
      // class constructor
      Fl_Callback_User_Data_133(const char * p0, int p1)
      : p0_(p0),                  // copy parameter 0
        p1_(p1)                   // copy parameter 1
      { }                         // constructor body
    };
    // connect our class to the widget callback
    btn->callback(Fl_Callback_User_Data_133::cb,
                  new Fl_Callback_User_Data_133("FLTK", 2),
                  true);
  } while(0);                     // user code adds semicolon
  ```

  Clicking the Fl_Button `btn` will call
 `fl_message("We received the message %s with %d!", "FLTK", 2);`.
  Deleting the button will also delete the data that was created in our
  boilerplate code.
 */

#define FL_INLINE_CALLBACK_5(WIDGET, TYPE0, NAME0, VALUE0, TYPE1, NAME1, VALUE1, TYPE2, NAME2, VALUE2, TYPE3, NAME3, VALUE3, TYPE4, NAME4, VALUE4, LAMBDA) \
  do { \
    class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data { \
    public: \
      TYPE0 p0_; TYPE1 p1_; TYPE2 p2_; TYPE3 p3_; TYPE4 p4_; \
      static void fn(TYPE0 NAME0, TYPE1 NAME1, TYPE2 NAME2, TYPE3 NAME3, TYPE4 NAME4) \
        LAMBDA \
      static void cb(Fl_Widget *w, void *user_data) { \
        _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data; \
        _FL_CBD_CLASS_NAME::fn(d->p0_, d->p1_, d->p2_, d->p3_, d->p4_); \
      } \
      _FL_CBD_CLASS_NAME(TYPE0 p0, TYPE1 p1, TYPE2 p2, TYPE3 p3, TYPE4 p4) \
      : p0_(p0), p1_(p1), p2_(p2), p3_(p3), p4_(p4) { } \
    }; \
    WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(VALUE0, VALUE1, VALUE2, VALUE3, VALUE4), true); \
  } while(0)

#define FL_INLINE_CALLBACK_4(WIDGET, TYPE0, NAME0, VALUE0, TYPE1, NAME1, VALUE1, TYPE2, NAME2, VALUE2, TYPE3, NAME3, VALUE3, LAMBDA) \
  do { \
    class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data { \
    public: \
      TYPE0 p0_; TYPE1 p1_; TYPE2 p2_; TYPE3 p3_; \
      static void fn(TYPE0 NAME0, TYPE1 NAME1, TYPE2 NAME2, TYPE3 NAME3) \
        LAMBDA \
      static void cb(Fl_Widget *w, void *user_data) { \
        _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data; \
        _FL_CBD_CLASS_NAME::fn(d->p0_, d->p1_, d->p2_, d->p3_); \
      } \
      _FL_CBD_CLASS_NAME(TYPE0 p0, TYPE1 p1, TYPE2 p2, TYPE3 p3) \
      : p0_(p0), p1_(p1), p2_(p2), p3_(p3) { } \
    }; \
    WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(VALUE0, VALUE1, VALUE2, VALUE3), true); \
  } while(0)

#define FL_INLINE_CALLBACK_3(WIDGET, TYPE0, NAME0, VALUE0, TYPE1, NAME1, VALUE1, TYPE2, NAME2, VALUE2, LAMBDA) \
  do { \
    class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data { \
    public: \
      TYPE0 p0_; TYPE1 p1_; TYPE2 p2_; \
      static void fn(TYPE0 NAME0, TYPE1 NAME1, TYPE2 NAME2) \
        LAMBDA \
      static void cb(Fl_Widget *w, void *user_data) { \
        _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data; \
        _FL_CBD_CLASS_NAME::fn(d->p0_, d->p1_, d->p2_); \
      } \
      _FL_CBD_CLASS_NAME(TYPE0 p0, TYPE1 p1, TYPE2 p2) \
      : p0_(p0), p1_(p1), p2_(p2) { } \
    }; \
    WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(VALUE0, VALUE1, VALUE2), true); \
  } while(0)

#define FL_INLINE_CALLBACK_2(WIDGET, TYPE0, NAME0, VALUE0, TYPE1, NAME1, VALUE1, LAMBDA) \
```
```

```
  do { \
    class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data { \
    public: \
      TYPE0 p0_; TYPE1 p1_; \
      static void fn(TYPE0 NAME0, TYPE1 NAME1) \
        LAMBDA \
      static void cb(Fl_Widget *w, void *user_data) { \
        _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data; \
        _FL_CBD_CLASS_NAME::fn(d->p0_, d->p1_); \
      } \
      _FL_CBD_CLASS_NAME(TYPE0 p0, TYPE1 p1) \
      : p0_(p0), p1_(p1) { } \
    }; \
    WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(VALUE0, VALUE1), true); \
  } while(0)

#define FL_INLINE_CALLBACK_1(WIDGET, TYPE0, NAME0, VALUE0, LAMBDA) \
  do { \
    class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data { \
    public: \
      TYPE0 p0_; \
      static void fn(TYPE0 NAME0) \
        LAMBDA \
      static void cb(Fl_Widget *w, void *user_data) { \
        _FL_CBD_CLASS_NAME *d = (_FL_CBD_CLASS_NAME*)user_data; \
        _FL_CBD_CLASS_NAME::fn(d->p0_); \
      } \
      _FL_CBD_CLASS_NAME(TYPE0 p0) \
      : p0_(p0) { } \
    }; \
    WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(VALUE0), true); \
  } while(0)

#define FL_INLINE_CALLBACK_0(WIDGET, LAMBDA) \
  do { \
    class _FL_CBD_CLASS_NAME : public Fl_Callback_User_Data { \
    public: \
    static void fn() \
      LAMBDA \
    static void cb(Fl_Widget *w, void *user_data) { \
      _FL_CBD_CLASS_NAME::fn(); \
    } \
    _FL_CBD_CLASS_NAME() { } \
    }; \
    WIDGET->callback(_FL_CBD_CLASS_NAME::cb, new _FL_CBD_CLASS_NAME(), true); \
  } while(0)

#endif // FL_DOXYGEN

#endif /* !_FL_FL_CALLBACK_MACROS_H_ */
```

14.35 fl_casts.H

```
//
// Experimental inline "cast functions" for the Fast Light Toolkit (FLTK).
// See also issue #109: "VS2017 warnings when building fltk 1.4.x"
//
// Copyright 1998-2021 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef _FL_fl_casts_H_
#define _FL_fl_casts_H_

#include <FL/platform_types.h>

inline char fl_char(void *v)             { return (char)(fl_intptr_t)v; }
inline int  fl_int(void *v)              { return (int)(fl_intptr_t)v; }
inline long fl_long(void *v)             { return (long)(fl_intptr_t)v; }

inline unsigned char fl_uchar(void *v)   { return (unsigned char)(fl_uintptr_t)v; }
inline unsigned int  fl_uint(void *v)    { return (unsigned int)(fl_uintptr_t)v; }
inline unsigned long fl_ulong(void *v)   { return (unsigned long)(fl_uintptr_t)v; }

// the following conversions can be used to silence MSVC warning C4312:
// 'type cast': conversion from '<type>' to 'void *' of greater size
```

```
00033
00034 inline void *fl_voidptr(int v)           { return (void *)(fl_intptr_t)v; }
00035 inline void *fl_voidptr(unsigned int v)  { return (void *)(fl_uintptr_t)v; }
00036 inline void *fl_voidptr(long v)          { return (void *)(fl_intptr_t)v; }
00037 inline void *fl_voidptr(unsigned long v) { return (void *)(fl_uintptr_t)v; }
00038
00039 #endif /* _FL_fl_casts_H_ */
```

14.36 Fl_Chart.H File Reference

Fl_Chart widget.

`#include "Fl_Widget.H"`

Classes

- class Fl_Chart

  Fl_Chart displays simple charts.
- struct FL_CHART_ENTRY

  For internal use only.

Macros

- #define **FL_BAR_CHART** 0

  type() for Bar Chart variant
- #define **FL_CHART_LABEL_MAX** 18

  max label length for entry
- #define **FL_CHART_MAX** 128

  max entries per chart
- #define **FL_FILL_CHART** 3

  type() for Fill Line Chart variant
- #define **FL_FILLED_CHART** FL_FILL_CHART

  for compatibility
- #define **FL_HORBAR_CHART** 1

  type() for Horizontal Bar Chart variant
- #define **FL_LINE_CHART** 2

  type() for Line Chart variant
- #define **FL_PIE_CHART** 5

  type() for Pie Chart variant
- #define **FL_SPECIALPIE_CHART** 6

  type() for Special Pie Chart variant
- #define **FL_SPIKE_CHART** 4

  type() for Spike Chart variant

14.36.1 Detailed Description

Fl_Chart widget.

14.37 Fl_Chart.H

Go to the documentation of this file.

```
00001 //
00002 // Fl_Chart widget header file for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 1998-2023 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file.  If this
```

```
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef Fl_Chart_H
#define Fl_Chart_H

#ifndef Fl_Widget_H
#include "Fl_Widget.H"
#endif

// values for type()
#define FL_BAR_CHART            0
#define FL_HORBAR_CHART         1
#define FL_LINE_CHART           2
#define FL_FILL_CHART           3
#define FL_SPIKE_CHART          4
#define FL_PIE_CHART            5
#define FL_SPECIALPIE_CHART     6

#define FL_FILLED_CHART FL_FILL_CHART

#define FL_CHART_MAX          128
#define FL_CHART_LABEL_MAX     18

struct FL_CHART_ENTRY {
  float val;
  unsigned col;
  char str[FL_CHART_LABEL_MAX + 1];
};

class FL_EXPORT Fl_Chart : public Fl_Widget {
  int numb;
  int maxnumb;
  int sizenumb;
  FL_CHART_ENTRY *entries;
  double min, max;
  uchar autosize_;
  Fl_Font textfont_;
  Fl_Fontsize textsize_;
  Fl_Color textcolor_;

protected:
  void draw() override;

  // (static) protected draw methods (STR 2022)
  // these methods are documented in src/Fl_Chart.cxx

  static void draw_barchart(int x, int y, int w, int h, int numb, FL_CHART_ENTRY entries[],
                            double min, double max, int autosize, int maxnumb, Fl_Color textcolor);

  static void draw_horbarchart(int x, int y, int w, int h, int numb, FL_CHART_ENTRY entries[],
                               double min, double max, int autosize, int maxnumb,
                               Fl_Color textcolor);

  static void draw_linechart(int type, int x, int y, int w, int h, int numb,
                             FL_CHART_ENTRY entries[], double min, double max, int autosize,
                             int maxnumb, Fl_Color textcolor);

  static void draw_piechart(int x, int y, int w, int h, int numb, FL_CHART_ENTRY entries[],
                            int special, Fl_Color textcolor);

public:
  Fl_Chart(int X, int Y, int W, int H, const char *L = 0);

  ~Fl_Chart();

  void clear();

  void add(double val, const char *str = 0, unsigned col = 0);

  void insert(int ind, double val, const char *str = 0, unsigned col = 0);

  void replace(int ind, double val, const char *str = 0, unsigned col = 0);

  void bounds(double *a, double *b) const {
    *a = min;
    *b = max;
  }

  void bounds(double a, double b);

```

```
  int size() const { return numb; }

  void size(int W, int H) { Fl_Widget::size(W, H); }

  int maxsize() const { return maxnumb; }

  void maxsize(int m);

  Fl_Font textfont() const { return textfont_; }

  void textfont(Fl_Font s) { textfont_ = s; }

  Fl_Fontsize textsize() const { return textsize_; }

  void textsize(Fl_Fontsize s) { textsize_ = s; }

  Fl_Color textcolor() const { return textcolor_; }

  void textcolor(Fl_Color n) { textcolor_ = n; }

  uchar autosize() const { return autosize_; }

  void autosize(uchar n) { autosize_ = n; }
};

#endif
```

14.38 Fl_Check_Browser.H

```
//
// Fl_Check_Browser header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2020 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Check_Browser widget . */

#ifndef Fl_Check_Browser_H
#define Fl_Check_Browser_H

#include "Fl.H"
#include "Fl_Browser_.H"

class FL_EXPORT Fl_Check_Browser : public Fl_Browser_ {

protected:
  /* required routines for Fl_Browser_ subclass: */
  void *item_first() const override;
  void *item_next(void *) const override;
  void *item_prev(void *) const override;
  int item_height(void *) const override;
  int item_width(void *) const override;
  void item_draw(void *, int, int, int, int) const override;
  void item_select(void *, int) override;
  int item_selected(void *) const override;
  const char *item_text(void *item) const override;

public:
  void *item_at(int index) const override;
  void item_swap(int ia, int ib);
  void item_swap(void *a, void *b) override;

  /* private data */

public: // IRIX 5.3 C++ compiler doesn't support private structures...

#ifndef FL_DOXYGEN
  struct cb_item {
    cb_item *next;
    cb_item *prev;
    char checked;
    char selected;
    char *text;
  };
```

```
#endif // !FL_DOXYGEN

private:
  cb_item *first;
  cb_item *last;
  cb_item *cache;
  int cached_item;
  int nitems_;
  int nchecked_;
  cb_item *find_item(int) const;
  int lineno(cb_item *) const;

public:
  Fl_Check_Browser(int x, int y, int w, int h, const char *l = 0);
  ~Fl_Check_Browser() { clear(); }
  int add(char *s);                // add an (unchecked) item
  int add(char *s, int b);         // add an item and set checked
                                   // both return the new nitems()
  int remove(int item);            // delete an item. Returns nitems()

  // inline const char * methods to avoid breaking binary compatibility...
  int add(const char *s) { return add((char *)s); }
  int add(const char *s, int b) { return add((char *)s, b); }

  void clear();                    // delete all items
  int nitems() const { return nitems_; }
  int nchecked() const { return nchecked_; }
  int checked(int item) const;
  void checked(int item, int b);
  void set_checked(int item) { checked(item, 1); }
  void check_all();
  void check_none();
  int value() const;               // currently selected item
  char *text(int item) const;      // returns pointer to internal buffer

protected:
  int handle(int) override;
};

#endif // Fl_Check_Browser_H
```

14.39 Fl_Check_Button.H

```
//
// Check button header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2014 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef Fl_Check_Button_H
#define Fl_Check_Button_H

#include "Fl_Light_Button.H"

/*
  class: Fl_Check_Button.

  A button with a "checkmark" to show its status.
*/

class FL_EXPORT Fl_Check_Button : public Fl_Light_Button {
public:
  Fl_Check_Button(int X, int Y, int W, int H, const char *L = 0);
};

#endif
```

14.40 Fl_Choice.H

```
//
// Choice header file for the Fast Light Tool Kit (FLTK).
//
```

```
// Copyright 1998-2024 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Choice widget . */

#ifndef Fl_Choice_H
#define Fl_Choice_H

#include "Fl_Menu_.H"

class FL_EXPORT Fl_Choice : public Fl_Menu_ {
protected:
  void draw() override;
public:
  int handle(int) override;

  Fl_Choice(int X, int Y, int W, int H, const char *L = 0);

  int value() const {return Fl_Menu_::value();}

  int value(int v);

  int value(const Fl_Menu_Item* v);
};

#endif
```

14.41 Fl_Clock.H

```
//
// Clock header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2017 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Clock, Fl_Clock_Output widgets . */

#ifndef Fl_Clock_H
#define Fl_Clock_H

#ifndef Fl_Widget_H
#include "Fl_Widget.H"
#endif

// Values for type():
// Please change doxygen documentation below (class Fl_Clock_Output)
// accordingly as well when changing the following type values:

#define FL_SQUARE_CLOCK         0
#define FL_ROUND_CLOCK          1
#define FL_ANALOG_CLOCK FL_SQUARE_CLOCK
#define FL_DIGITAL_CLOCK FL_SQUARE_CLOCK
// fabien: Please keep the horizontal formatting of both images in class desc,
// don't lose vertical space for nothing!

class FL_EXPORT Fl_Clock_Output : public Fl_Widget {
  int hour_, minute_, second_;
  ulong value_;
  int shadow_; // draw shadows of hands
  void drawhands(Fl_Color,Fl_Color); // part of draw
protected:
  void draw() override;
  void draw(int X, int Y, int W, int H);
```

```
public:

  Fl_Clock_Output(int X, int Y, int W, int H, const char *L = 0);

  void value(ulong v);  // set to this Unix time

  void value(int H, int m, int s);

  ulong value() const {return value_;}

  int hour() const {return hour_;}

  int minute() const {return minute_;}

  int second() const {return second_;}

  int shadow() const {return shadow_;}

  void shadow(int mode) { shadow_ = mode ? 1 : 0; }
};

// a Fl_Clock displays the current time always by using a timeout:

class FL_EXPORT Fl_Clock : public Fl_Clock_Output {
public:
  int handle(int) override;

  Fl_Clock(int X, int Y, int W, int H,  const char *L = 0);

  Fl_Clock(uchar t, int X, int Y, int W, int H, const char *L);

  ~Fl_Clock();
};

#endif
```

14.42 Fl_Color_Chooser.H File Reference

Fl_Color_Chooser widget .

```
#include <FL/Fl_Group.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Return_Button.H>
#include <FL/Fl_Choice.H>
#include <FL/Fl_Value_Input.H>
#include <string>
```

Classes

- class Fl_Color_Chooser

  The Fl_Color_Chooser widget provides a standard RGB color chooser.

14.42.1 Detailed Description

Fl_Color_Chooser widget .

14.43 Fl_Color_Chooser.H

Go to the documentation of this file.

```
//
// Color chooser header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2026 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
```

```
//

// The color chooser object and the color chooser popup.  The popup
// is just a window containing a single color chooser and some boxes
// to indicate the current and cancelled color.

#ifndef Fl_Color_Chooser_H
#define Fl_Color_Chooser_H

#include <FL/Fl_Group.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Return_Button.H>
#include <FL/Fl_Choice.H>
#include <FL/Fl_Value_Input.H>

#include <string>

#ifndef FL_DOXYGEN

class FL_EXPORT Flcc_HueBox : public Fl_Widget {
  int px, py;
protected:
  void draw() override;
  int handle_key(int);
public:
  int handle(int) override;
  Flcc_HueBox(int X, int Y, int W, int H) : Fl_Widget(X,Y,W,H) {
  px = py = 0;}
};

class FL_EXPORT Flcc_ValueBox : public Fl_Widget {
  int py;
protected:
  void draw() override;
  int handle_key(int);
public:
  int handle(int) override;
  Flcc_ValueBox(int X, int Y, int W, int H) : Fl_Widget(X,Y,W,H) {
  py = 0;}
};

class FL_EXPORT Flcc_Value_Input : public Fl_Value_Input {
public:
  int format(char*) override;
  std::string format_str() override;
  Flcc_Value_Input(int X, int Y, int W, int H) : Fl_Value_Input(X,Y,W,H) {}
};

#endif // !FL_DOXYGEN

class FL_EXPORT Fl_Color_Chooser : public Fl_Group {
  Flcc_HueBox huebox;
  Flcc_ValueBox valuebox;
  Fl_Choice choice;
  Flcc_Value_Input rvalue;
  Flcc_Value_Input gvalue;
  Flcc_Value_Input bvalue;
  Fl_Box resize_box;
  double hue_, saturation_, value_;
  double r_, g_, b_;
  void set_valuators();
  static void rgb_cb(Fl_Widget*, void*);
  static void mode_cb(Fl_Widget*, void*);
public:

  int handle(int e) override;

  int mode() const { return choice.value(); }

  void mode(int newMode);

  double hue() const {return hue_;}

  double saturation() const {return saturation_;}

  double value() const {return value_;}

  double r() const {return r_;}

  double g() const {return g_;}

  double b() const {return b_;}

  int hsv(double H, double S, double V);

  int rgb(double R, double G, double B);

```

```
  static void hsv2rgb(double H, double S, double V, double& R, double& G, double& B);

  static void rgb2hsv(double R, double G, double B, double& H, double& S, double& V);

  Fl_Color_Chooser(int X, int Y, int W, int H, const char *L = 0);
};

FL_EXPORT int fl_color_chooser(const char* name, double& r, double& g, double& b, int m=-1);
FL_EXPORT int fl_color_chooser(const char* name, uchar& r, uchar& g, uchar& b, int m=-1);

#endif
```

14.44 Fl_Copy_Surface.H

```
//
// Copy-to-clipboard code for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2023 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef Fl_Copy_Surface_H
#define Fl_Copy_Surface_H

#include <FL/Fl_Widget_Surface.H>

class FL_EXPORT Fl_Copy_Surface : public Fl_Widget_Surface {
private:
  class Fl_Copy_Surface_Driver *platform_surface;
protected:
  void translate(int x, int y) override;
  void untranslate() override;
public:
  Fl_Copy_Surface(int w, int h);
  ~Fl_Copy_Surface();
  void set_current() override;
  bool is_current() override;
  int w();
  int h();
  void origin(int *x, int *y) override;
  void origin(int x, int y) override;
  int printable_rect(int *w, int *h) override;
};


class Fl_Copy_Surface_Driver : public Fl_Widget_Surface {
  friend class Fl_Copy_Surface;
protected:
  int width;
  int height;
  Fl_Copy_Surface_Driver(int w, int h) : Fl_Widget_Surface(NULL), width(w), height(h) {}
  virtual ~Fl_Copy_Surface_Driver() {}
  void set_current() override = 0;
  void translate(int x, int y) override = 0;
  void untranslate() override = 0;
  int printable_rect(int *w, int *h) override;
  static Fl_Copy_Surface_Driver *newCopySurfaceDriver(int w, int h);
};

#endif // Fl_Copy_Surface_H
```

14.45 Fl_Counter.H

```
//
// Counter header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2022 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
```

```
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Counter widget . */

// A numerical value with up/down step buttons.  From Forms.

#ifndef Fl_Counter_H
#define Fl_Counter_H

#ifndef Fl_Valuator_H
#include "Fl_Valuator.H"
#endif

// values for type():
#define FL_NORMAL_COUNTER       0
#define FL_SIMPLE_COUNTER       1
class FL_EXPORT Fl_Counter : public Fl_Valuator {

  Fl_Font textfont_;
  Fl_Fontsize textsize_;
  Fl_Color textcolor_;
  double lstep_;
  uchar mouseobj_;
  static void repeat_callback(void *);
  int calc_mouseobj();
  void increment_cb();

protected:

  void draw() override;
  // compute widths of arrow boxes
  void arrow_widths(int &w1, int &w2);

public:

  int handle(int) override;

  Fl_Counter(int X, int Y, int W, int H, const char* L = 0);
  ~Fl_Counter();

  void lstep(double a) {lstep_ = a;}

  void step(double a,double b) {Fl_Valuator::step(a); lstep_ = b;}

  void step(double a) {Fl_Valuator::step(a);}

  double step() const {return Fl_Valuator::step();}

  Fl_Font textfont() const {return textfont_;}
  void textfont(Fl_Font s) {textfont_ = s;}

  Fl_Fontsize textsize() const {return textsize_;}
  void textsize(Fl_Fontsize s) {textsize_ = s;}

  Fl_Color textcolor() const {return textcolor_;}
  void textcolor(Fl_Color s) {textcolor_ = s;}

};

#endif
```

14.46 Fl_Device.H File Reference

declaration of classes Fl_Surface_Device, Fl_Display_Device, Fl_Device_Plugin.

```
#include <FL/Fl_Plugin.H>
#include <FL/platform_types.h>
```

Classes

- class Fl_Device_Plugin

  This plugin socket allows the integration of new device drivers for special window or screen types.
- class Fl_Display_Device

  The computer's display.

- class Fl_Surface_Device

  A drawing surface that's susceptible to receive graphical output.

14.46.1 Detailed Description

declaration of classes Fl_Surface_Device, Fl_Display_Device, Fl_Device_Plugin.

14.47 Fl_Device.H

Go to the documentation of this file.

```
00001 //
00002 // Definition of classes Fl_Surface_Device, Fl_Display_Device
00003 // for the Fast Light Tool Kit (FLTK).
00004 //
00005 // Copyright 2010-2026 by Bill Spitzak and others.
00006 //
00007 // This library is free software. Distribution and use rights are outlined in
00008 // the file "COPYING" which should have been included with this file.  If this
00009 // file is missing or damaged, see the license at:
00010 //
00011 //     https://www.fltk.org/COPYING.php
00012 //
00013 // Please see the following page on how to report bugs and issues:
00014 //
00015 //     https://www.fltk.org/bugs.php
00016 //
00017
00022 #ifndef Fl_Device_H
00023 #define Fl_Device_H
00024
00025 #include <FL/Fl_Plugin.H>
00026 #include <FL/platform_types.h>
00027
00028 class Fl_Graphics_Driver;
00029 class Fl_RGB_Image;
00030 class Fl_Widget;
00031 class Fl_Image_Surface;
00032 class Fl_Image;
00033
00058 class FL_EXPORT Fl_Surface_Device {
00060   Fl_Graphics_Driver *pGraphicsDriver;
00061   static Fl_Surface_Device *surface_; // the surface that currently receives graphics requests
00062   static Fl_Surface_Device *default_surface(); // create surface if none exists yet
00063 protected:
00068   virtual void end_current() { surface_ = 0;}
00070   Fl_Surface_Device(Fl_Graphics_Driver *graphics_driver) {pGraphicsDriver = graphics_driver; }
00072   inline void driver(Fl_Graphics_Driver *graphics_driver) {pGraphicsDriver = graphics_driver;}
00073 public:
00074   virtual void set_current(void);
00075   virtual bool is_current();
00077   inline Fl_Graphics_Driver *driver() {return pGraphicsDriver; }
00082   static inline Fl_Surface_Device *surface() {
00083     return surface_ ? surface_ : default_surface();
00084   }
00086   virtual ~Fl_Surface_Device();
00087   static void push_current(Fl_Surface_Device *new_current);
00088   static Fl_Surface_Device *pop_current();
00089 };
00090
00096 class FL_EXPORT Fl_Display_Device : public Fl_Surface_Device {
00097   Fl_Display_Device(Fl_Graphics_Driver *graphics_driver);
00098 public:
00099   static Fl_Display_Device *display_device();
00100 };
00101
00109 class Fl_Device_Plugin : public Fl_Plugin {
00110 public:
00112   Fl_Device_Plugin(const char *pluginName)
00113   : Fl_Plugin(klass(), pluginName) { }
00115   virtual const char *klass() { return "fltk:device"; }
00117   virtual const char *name() = 0;
00119   virtual int print(Fl_Widget* w) = 0;
00123   virtual Fl_RGB_Image* rectangle_capture(Fl_Widget *widget, int x, int y, int w, int h) = 0;
00125   static Fl_Device_Plugin *opengl_plugin();
00126   virtual void delete_image_texture(Fl_Image*) = 0;
00127 };
00128
00129 #endif // Fl_Device_H
```

14.48 Fl_Dial.H

```
//
// Dial header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2022 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Dial widget . */

#ifndef Fl_Dial_H
#define Fl_Dial_H

#ifndef Fl_Valuator_H
#include "Fl_Valuator.H"
#endif

// values for type():
#define FL_NORMAL_DIAL  0
#define FL_LINE_DIAL    1
#define FL_FILL_DIAL    2
class FL_EXPORT Fl_Dial : public Fl_Valuator {

  short a1,a2;

protected:

  // these allow subclasses to put the dial in a smaller area:
  void draw(int X, int Y, int W, int H);
  int handle(int event, int X, int Y, int W, int H);
  void draw() override;

public:

  int handle(int) override;
  Fl_Dial(int x,int y,int w,int h, const char *l = 0);
  short angle1() const {return a1;}
  void angle1(short a) {a1 = a;}
  short angle2() const {return a2;}
  void angle2(short a) {a2 = a;}
  void angles(short a, short b) {a1 = a; a2 = b;}

};

#endif
```

14.49 Fl_Double_Window.H

```
//
// Double-buffered window header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Double_Window widget . */

#ifndef Fl_Double_Window_H
#define Fl_Double_Window_H

#include "Fl_Window.H"

class FL_EXPORT Fl_Double_Window : public Fl_Window
```

```
{
public:
  Fl_Double_Window *as_double_window() override {return this; }
  void show() override;
  void show(int a, char **b) {Fl_Window::show(a,b);}
  void resize(int,int,int,int) override;
  void hide() override;
  void flush() override;
  ~Fl_Double_Window();

  Fl_Double_Window(int W, int H, const char *l = 0);

  Fl_Double_Window(int X, int Y, int W, int H, const char *l = 0);

};

#endif
```

14.50 fl_draw.H File Reference

utility header to pull drawing functions together

```
#include <FL/Enumerations.H>
#include <FL/Fl_Graphics_Driver.H>
#include <FL/Fl_Rect.H>
```

Enumerations

- enum {
  FL_SOLID = 0 , FL_DASH = 1 , FL_DOT = 2 , FL_DASHDOT = 3 ,
  FL_DASHDOTDOT = 4 , FL_CAP_FLAT = 0x100 , FL_CAP_ROUND = 0x200 , FL_CAP_SQUARE = 0x300 ,
  FL_JOIN_MITER = 0x1000 , FL_JOIN_ROUND = 0x2000 , FL_JOIN_BEVEL = 0x3000 , FL_UNIFORM_WIDTH = 0x10000 }

Functions

- int fl_add_symbol (const char *name, void(*draw_in_rect)(int x, int y, int w, int h, Fl_Color c), int scalable)

  Registers (or replaces) a named symbol drawn into an explicit pixel rectangle.
- int fl_add_symbol (const char *name, void(*drawit)(Fl_Color c), int scalable)

  Registers (or replaces) a named symbol drawn using complex vector drawing.
- int **fl_antialias** ()

  Return whether line drawings are currently antialiased.
- void fl_antialias (int state)

  Turn antialiased line drawings ON or OFF, if supported by platform.
- void fl_arc (double x, double y, double r, double start, double end)

  Add a series of points to the current path on the arc of a circle.
- void fl_arc (int x, int y, int w, int h, double a1, double a2)

  Draw ellipse sections using integer coordinates.
- void fl_begin_complex_polygon ()

  Start drawing a complex filled polygon.
- void **fl_begin_line** ()

  Start drawing a list of lines.
- void **fl_begin_loop** ()

  Start drawing a closed sequence of lines.
- void fl_begin_offscreen (Fl_Offscreen ctx)

  Send all subsequent drawing commands to this offscreen buffer.
- void fl_begin_points ()

  Start drawing a list of points.
- void **fl_begin_polygon** ()

Start drawing a convex filled polygon.

- char fl_can_do_alpha_blending ()

  Check whether platform supports true alpha blending for RGBA images.
- Fl_RGB_Image * fl_capture_window (Fl_Window *win, int x, int y, int w, int h)

  Captures the content of a rectangular zone of a mapped window.
- void **fl_chord** (int x, int y, int w, int h, double a1, double a2)

  fl_chord declaration is a place holder - the function does not yet exist
- void fl_circle (double x, double y, double r)

  fl_circle(x,y,r) is equivalent to fl_arc(x,y,r,0,360), but may be faster.
- int fl_clip_box (int x, int y, int w, int h, int &X, int &Y, int &W, int &H)

  Intersect a rectangle with the current clip region and return the bounding box of the result.
- Fl_Region fl_clip_region ()

  Return the current clipping region.
- void fl_clip_region (Fl_Region r)

  Replace the top of the clipping stack with a clipping region of any shape.
- Fl_Color fl_color ()

  Return the last fl_color() that was set.
- void fl_color (Fl_Color c)

  Set the color for all subsequent drawing operations.
- void **fl_color** (int c)

  for back compatibility - use fl_color(Fl_Color c) instead
- void fl_color (uchar r, uchar g, uchar b)

  Set the color for all subsequent drawing operations.
- void fl_copy_offscreen (int x, int y, int w, int h, Fl_Offscreen pixmap, int srcx, int srcy)

  Copy a rectangular area of the given offscreen buffer into the current drawing destination.
- Fl_Offscreen fl_create_offscreen (int w, int h)

  Creation of an offscreen graphics buffer.
- void fl_cursor (Fl_Cursor)

  Sets the cursor for the current window to the specified shape and colors.
- void **fl_cursor** (Fl_Cursor, Fl_Color fg, Fl_Color bg=FL_WHITE)
- void fl_curve (double X0, double Y0, double X1, double Y1, double X2, double Y2, double X3, double Y3)

  Add a series of points on a Bézier curve to the path.
- void fl_delete_offscreen (Fl_Offscreen ctx)

  Deletion of an offscreen graphics buffer.
- int fl_descent ()

  Return the recommended distance above the bottom of a fl_height() tall box to draw the text at so it looks centered vertically in that box.
- void **fl_draw** (const char *str, int n, int x, int y)

  Draws starting at the given `x`, `y` location a UTF-8 string of length `n` bytes.
- void fl_draw (const char *str, int x, int y)

  Draw a nul-terminated UTF-8 string starting at the given `x`, `y` location.
- void fl_draw (const char *str, int x, int y, int w, int h, Fl_Align align, Fl_Image *img=0, int draw_symbols=1, int spacing=0)

  Fancy string drawing function which is used to draw all the labels.
- void fl_draw (const char *str, int x, int y, int w, int h, Fl_Align align, void(*callthis)(const char *, int, int, int), Fl_Image *img=0, int draw_symbols=1, int spacing=0)

  The same as fl_draw(const char,int,int,int,int,Fl_Align,Fl_Image*,int) with the addition of the `callthis` parameter, which is a pointer to a text drawing function such as fl_draw(const char*, int, int, int) to do the real work.*
- void fl_draw (int angle, const char *str, int n, int x, int y)

  Draw at the given `x`, `y` location a UTF-8 string of length `n` bytes rotating `angle` degrees counter-clockwise.
- void fl_draw (int angle, const char *str, int x, int y)

Draw a nul-terminated UTF-8 string starting at the given `x`, `y` location and rotating `angle` degrees counter-clockwise.

- void fl_draw_arrow (Fl_Rect bb, Fl_Arrow_Type t, Fl_Orientation o, Fl_Color color)

  Draw an "arrow like" GUI element for the selected scheme.
- void fl_draw_box (Fl_Boxtype, int x, int y, int w, int h, Fl_Color)

  Draws a box using given type, position, size and color.
- void fl_draw_box_focus (Fl_Boxtype, int x, int y, int w, int h, Fl_Color, Fl_Color)

  Draws the focus rectangle inside a box using given type, position, size and color.
- void fl_draw_check (Fl_Rect bb, Fl_Color col)

  Draw a check mark inside the given bounding box.
- void fl_draw_circle (int x, int y, int d, Fl_Color color)

  Draw a potentially small, filled circle using a given color.
- void fl_draw_image (const uchar ∗buf, int X, int Y, int W, int H, int D=3, int L=0)

  Draw an 8-bit per color RGB or luminance image.
- void fl_draw_image (Fl_Draw_Image_Cb cb, void ∗data, int X, int Y, int W, int H, int D=3)

  Draw an image using a callback function to generate image data.
- void fl_draw_image_mono (const uchar ∗buf, int X, int Y, int W, int H, int D=1, int L=0)

  Draw a gray-scale (1 channel) image.
- void fl_draw_image_mono (Fl_Draw_Image_Cb cb, void ∗data, int X, int Y, int W, int H, int D=1)

  Draw a gray-scale image using a callback function to generate image data.
- int fl_draw_pixmap (char ∗const ∗data, int x, int y, Fl_Color bg=FL_GRAY)

  Draw XPM image data, with the top-left corner at the given position.
- int fl_draw_pixmap (const char ∗const ∗cdata, int x, int y, Fl_Color bg=FL_GRAY)

  Draw XPM image data, with the top-left corner at the given position.
- void fl_draw_radio (int x, int y, int d, Fl_Color color)

  Draw a round check mark (circle) of a radio button.
- int fl_draw_symbol (const char ∗label, int x, int y, int w, int h, Fl_Color)

  Draw the named symbol in the given rectangle using the given color.
- void **fl_end_complex_polygon** ()

  End complex filled polygon, and draw.
- void **fl_end_line** ()

  End list of lines, and draw.
- void **fl_end_loop** ()

  End closed sequence of lines, and draw.
- void **fl_end_offscreen** ()

  Quit sending drawing commands to the current offscreen buffer.
- void **fl_end_points** ()

  End list of points, and draw.
- void **fl_end_polygon** ()

  End convex filled polygon, and draw.
- const char ∗ fl_expand_text (const char ∗from, char ∗buf, int maxbuf, double maxw, int &n, double &width, int wrap, int draw_symbols=0)

  Copy `from` to `buf`, replacing control characters with ^X.
- void fl_focus_rect (int x, int y, int w, int h)

  Draw a dotted rectangle, used to indicate keyboard focus on a widget.
- Fl_Font fl_font ()

  Return the `face` set by the most recent call to fl_font().
- void fl_font (Fl_Font face, Fl_Fontsize fsize)

  Sets the current font, which is then used in various drawing routines.
- void fl_frame (const char ∗s, int x, int y, int w, int h)

  Draws a series of line segments around the given box.
- void fl_frame2 (const char ∗s, int x, int y, int w, int h)

Draws a series of line segments around the given box.

- void fl_gap ()

  Separate loops of the path.
- int fl_height ()

  Return the recommended minimum line spacing for the current font.
- int fl_height (int font, int size)

  This function returns the actual height of the specified `font` and `size`.
- const char * fl_latin1_to_local (const char *t, int n=-1)

  Convert text from Windows/X11 latin1 character set to local encoding.
- void **fl_line** (int x, int y, int x1, int y1)

  Draw a line from (x,y) to (x1,y1)
- void **fl_line** (int x, int y, int x1, int y1, int x2, int y2)

  Draw a line from (x,y) to (x1,y1) and another from (x1,y1) to (x2,y2)
- void fl_line_style (int style, int width=0, char *dashes=0)

  Set how to draw lines (the "pen").
- void **fl_load_identity** ()

  Set the transformation matrix to identity.
- void fl_load_matrix (double a, double b, double c, double d, double x, double y)

  Set the current transformation matrix.
- const char * fl_local_to_latin1 (const char *t, int n=-1)

  Convert text from local encoding to Windows/X11 latin1 character set.
- const char * fl_local_to_mac_roman (const char *t, int n=-1)

  Convert text from local encoding to Mac Roman character set.
- void **fl_loop** (int x, int y, int x1, int y1, int x2, int y2)

  Outline a 3-sided polygon with lines.
- void **fl_loop** (int x, int y, int x1, int y1, int x2, int y2, int x3, int y3)

  Outline a 4-sided polygon with lines.
- const char * fl_mac_roman_to_local (const char *t, int n=-1)

  Convert text from Mac Roman character set to local encoding.
- void fl_measure (const char *str, int &x, int &y, int draw_symbols=1)

  Measure how wide and tall the string will be when printed by the fl_draw() function with `align` parameter.
- int fl_measure_pixmap (char *const *data, int &w, int &h)

  Get the dimensions of a pixmap.
- int fl_measure_pixmap (const char *const *cdata, int &w, int &h)

  Get the dimensions of a pixmap.
- void fl_mult_matrix (double a, double b, double c, double d, double x, double y)

  Concatenate another transformation onto the current one.
- int fl_not_clipped (int x, int y, int w, int h)

  Does the rectangle intersect the current clip region?
- unsigned int fl_old_shortcut (const char *s)

  Emulation of XForms named shortcuts.
- void fl_overlay_clear ()

  Erase a selection rectangle without drawing a new one.
- void fl_overlay_rect (int x, int y, int w, int h)

  Draw a transient dotted selection rectangle.
- float fl_override_scale ()

  Removes any GUI scaling factor in subsequent drawing operations.
- void fl_pie (int x, int y, int w, int h, double a1, double a2)

  Draw filled ellipse sections using integer coordinates.
- void **fl_point** (int x, int y)

  Draw a single pixel at the given coordinates.

- void **fl_polygon** (int x, int y, int x1, int y1, int x2, int y2)

  Fill a 3-sided polygon.
- void fl_polygon (int x, int y, int x1, int y1, int x2, int y2, int x3, int y3)

  Fill a 4-sided polygon.
- void fl_pop_clip ()

  Restore the previous clip region.
- void **fl_pop_matrix** ()

  Restore the current transformation matrix from the stack.
- void fl_push_clip (int x, int y, int w, int h)

  Intersect the current clip region with a rectangle and push this new region onto the stack.
- void fl_push_matrix ()

  Save the current transformation matrix on the stack.
- void **fl_push_no_clip** ()

  Push an empty clip region onto the stack so nothing will be clipped.
- uchar * fl_read_image (uchar *p, int X, int Y, int W, int H, int alpha=0)

  Reads an RGB(A) image from the current window or off-screen buffer.
- void fl_rect (Fl_Rect r)

  Draw a border inside the given bounding box.
- void fl_rect (int x, int y, int w, int h)

  Draw a border inside the given bounding box.
- void fl_rect (int x, int y, int w, int h, Fl_Color c)

  Draw with passed color a border inside the given bounding box.
- void fl_rectf (Fl_Rect bb, uchar r, uchar g, uchar b)

  Color a rectangle with "exactly" the passed `r,g,b` color.
- void **fl_rectf** (Fl_Rect r)

  Color with current color a rectangle that exactly fills the given bounding box.
- void fl_rectf (Fl_Rect r, Fl_Color c)

  Color with passed color a rectangle that exactly fills the given bounding box.
- void **fl_rectf** (int x, int y, int w, int h)

  Color with current color a rectangle that exactly fills the given bounding box.
- void fl_rectf (int x, int y, int w, int h, Fl_Color c)

  Color with passed color a rectangle that exactly fills the given bounding box.
- void fl_rectf (int x, int y, int w, int h, uchar r, uchar g, uchar b)

  Color a rectangle with "exactly" the passed `r,g,b` color.
- int fl_remove_symbol (const char *name)

  Removes the symbol with name `name` from the system.
- void fl_rescale_offscreen (Fl_Offscreen &ctx)

  Adapts an offscreen buffer to a changed value of the scale factor.
- void fl_reset_spot (void)

  Resets marked text.
- void **fl_restore_clip** ()

  Undo any clobbering of the clip region done by your program.
- void fl_restore_scale (float s)

  Restores the GUI scaling factor in subsequent drawing operations.
- void fl_rotate (double d)

  Concatenate rotation transformation onto the current one.
- void fl_rounded_rect (int x, int y, int w, int h, int r)

  Draw a rounded border inside the given bounding box.
- void fl_rounded_rectf (int x, int y, int w, int h, int r)

  Color with current color a rounded rectangle that exactly fills the given bounding box.
- void **fl_rtl_draw** (const char *str, int n, int x, int y)

Draw a UTF-8 string of length n bytes right to left starting at the given x, y location.

- void fl_scale (double x)

  Concatenate scaling transformation onto the current one.
- void fl_scale (double x, double y)

  Concatenate scaling transformation onto the current one.
- void fl_scroll (int X, int Y, int W, int H, int dx, int dy, void(∗draw_area)(void ∗, int, int, int, int), void ∗data)

  Scroll a rectangle and draw the newly exposed portions.
- void fl_set_spot (int font, int size, int X, int Y, int W, int H, Fl_Window ∗win=0)

  Inform text input methods about the current text insertion cursor.
- void fl_set_status (int X, int Y, int W, int H)

  Related to text input methods under X11.
- const char ∗ fl_shortcut_label (unsigned int shortcut)

  Get a human-readable string from a shortcut value.
- const char ∗ fl_shortcut_label (unsigned int shortcut, const char ∗∗eom)

  Get a human-readable string from a shortcut value.
- Fl_Fontsize fl_size ()

  Return the size set by the most recent call to fl_font().
- void fl_text_extents (const char ∗t, int &dx, int &dy, int &w, int &h)

  Determine the minimum pixel dimensions of a nul-terminated string using the current fl_font().
- void fl_text_extents (const char ∗t, int n, int &dx, int &dy, int &w, int &h)

  Determine the minimum pixel dimensions of a sequence of n bytes of text using the current font face and size.
- double fl_transform_dx (double x, double y)

  Transform distance using current transformation matrix.
- double fl_transform_dy (double x, double y)

  Transform distance using current transformation matrix.
- double fl_transform_x (double x, double y)

  Transform coordinate using the current transformation matrix.
- double fl_transform_y (double x, double y)

  Transform coordinate using the current transformation matrix.
- void fl_transformed_vertex (double xf, double yf)

  Add coordinate pair to the vertex list without further transformations.
- void fl_translate (double x, double y)

  Concatenate translation transformation onto the current one.
- void fl_vertex (double x, double y)

  Add a single vertex to the current path.
- double fl_width (const char ∗txt)

  Return the typographical width of a nul-terminated string using the current font face and size.
- double fl_width (const char ∗txt, int n)

  Return the typographical width of a sequence of n bytes of text in UTF-8 encoding, using the current font face and size.
- double fl_width (unsigned int c)

  Return the typographical width of a single character using the current font face and size.
- void **fl_xyline** (int x, int y, int x1)

  Draw a horizontal line from (x,y) to (x1,y).
- void **fl_xyline** (int x, int y, int x1, int y2)

  Draw a horizontal line from (x,y) to (x1,y), then vertical from (x1,y) to (x1,y2).
- void **fl_xyline** (int x, int y, int x1, int y2, int x3)

  Draw a horizontal line from (x,y) to (x1,y), then a vertical from (x1,y) to (x1,y2) and then another horizontal from (x1,y2) to (x3,y2).
- void **fl_yxline** (int x, int y, int y1)

  Draw a vertical line from (x,y) to (x,y1)

- void **fl_yxline** (int x, int y, int y1, int x2)

  Draw a vertical line from (x,y) to (x,y1), then a horizontal from (x,y1) to (x2,y1).
- void **fl_yxline** (int x, int y, int y1, int x2, int y3)

  Draw a vertical line from (x,y) to (x,y1), then a horizontal from (x,y1) to (x2,y1), then another vertical from (x2,y1) to (x2,y3).

Variables

- char **fl_draw_shortcut**

14.50.1 Detailed Description

utility header to pull drawing functions together

14.51 fl_draw.H

Go to the documentation of this file.

```
00001 //
00002 // Portable drawing function header file for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 1998-2026 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file.  If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 //     https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 //     https://www.fltk.org/bugs.php
00015 //
00016
00022 #ifndef fl_draw_H
00023 #define fl_draw_H
00024
00025 #include <FL/Enumerations.H>       // color names
00026 #include <FL/Fl_Graphics_Driver.H> // fl_graphics_driver + Fl_Region
00027 #include <FL/Fl_Rect.H>
00028
00029 // Image class...
00030 class Fl_Image;
00031 class Fl_Window;
00032
00033 // Label flags...
00034 FL_EXPORT extern char fl_draw_shortcut;
00035
00040 // Colors:
00050 inline void fl_color(Fl_Color c) {
00051   fl_graphics_driver->color(c);
00052 } // select indexed color
00054 inline void fl_color(int c) {
00055   fl_color((Fl_Color)c);
00056 }
00067 inline void fl_color(uchar r, uchar g, uchar b) {
00068   fl_graphics_driver->color(r, g, b);
00069 }
00074 inline Fl_Color fl_color() {
00075   return fl_graphics_driver->color();
00076 }
00082 // clip:
00088 inline void fl_push_clip(int x, int y, int w, int h) {
00089   fl_graphics_driver->push_clip(x, y, w, h);
00090 }
00094 inline void fl_push_no_clip() {
00095   fl_graphics_driver->push_no_clip();
00096 }
00104 inline void fl_pop_clip() {
00105   fl_graphics_driver->pop_clip();
00106 }
00107
00121 inline int fl_not_clipped(int x, int y, int w, int h) {
00122   return fl_graphics_driver->not_clipped(x, y, w, h);
00123 }
00124
00167 inline int fl_clip_box(int x, int y, int w, int h, int &X, int &Y, int &W, int &H) {
00168   return fl_graphics_driver->clip_box(x, y, w, h, X, Y, W, H);
00169 }
```

```
inline void fl_restore_clip() {
  fl_graphics_driver->restore_clip();
}

inline void fl_clip_region(Fl_Region r) {
  fl_graphics_driver->clip_region(r);
}

inline Fl_Region fl_clip_region() {
  return fl_graphics_driver->clip_region();
}


// points:
inline void fl_point(int x, int y) {
  fl_graphics_driver->point(x, y);
}

// line type:
inline void fl_line_style(int style, int width = 0, char *dashes = 0) {
  fl_graphics_driver->line_style(style, width, dashes);
}

enum {
  FL_SOLID      = 0,
  FL_DASH       = 1,
  FL_DOT        = 2,
  FL_DASHDOT    = 3,
  FL_DASHDOTDOT = 4,

  FL_CAP_FLAT   = 0x100,
  FL_CAP_ROUND  = 0x200,
  FL_CAP_SQUARE = 0x300,

  FL_JOIN_MITER = 0x1000,
  FL_JOIN_ROUND = 0x2000,
  FL_JOIN_BEVEL = 0x3000,

  FL_UNIFORM_WIDTH = 0x10000
};

inline void fl_antialias(int state) {
  fl_graphics_driver->antialias(state);
}

inline int fl_antialias() {
  return fl_graphics_driver->antialias();
}

// rectangles tweaked to exactly fill the pixel rectangle:

inline void fl_rect(int x, int y, int w, int h) {
  fl_graphics_driver->rect(x, y, w, h);
}

inline void fl_rounded_rect(int x, int y, int w, int h, int r) {
  fl_graphics_driver->rounded_rect(x, y, w, h, r);
}

inline void fl_rect(Fl_Rect r) {
  fl_rect(r.x(), r.y(), r.w(), r.h());
}

inline void fl_focus_rect(int x, int y, int w, int h) {
  fl_graphics_driver->focus_rect(x, y, w, h);
}

inline void fl_rect(int x, int y, int w, int h, Fl_Color c) {
  fl_color(c);
  fl_rect(x, y, w, h);
}

inline void fl_rectf(int x, int y, int w, int h) {
  fl_graphics_driver->rectf(x, y, w, h);
}

inline void fl_rounded_rectf(int x, int y, int w, int h, int r) {
  fl_graphics_driver->rounded_rectf(x, y, w, h, r);
}

inline void fl_rectf(int x, int y, int w, int h, Fl_Color c) {
  fl_color(c);
  fl_rectf(x, y, w, h);
}

inline void fl_rectf(Fl_Rect r) {
```

```
  fl_graphics_driver->rectf(r.x(), r.y(), r.w(), r.h());
}

inline void fl_rectf(Fl_Rect r, Fl_Color c) {
  fl_color(c);
  fl_rectf(r);
}

inline void fl_rectf(int x, int y, int w, int h, uchar r, uchar g, uchar b) {
  fl_graphics_driver->colored_rectf(x, y, w, h, r, g, b);
}

inline void fl_rectf(Fl_Rect bb, uchar r, uchar g, uchar b) {
  fl_graphics_driver->colored_rectf(bb.x(), bb.y(), bb.w(), bb.h(), r, g, b);
}

// line segments:
inline void fl_line(int x, int y, int x1, int y1) {
  fl_graphics_driver->line(x, y, x1, y1);
}
inline void fl_line(int x, int y, int x1, int y1, int x2, int y2) {
  fl_graphics_driver->line(x, y, x1, y1, x2, y2);
}

// closed line segments:
inline void fl_loop(int x, int y, int x1, int y1, int x2, int y2) {
  fl_graphics_driver->loop(x, y, x1, y1, x2, y2);
}
inline void fl_loop(int x, int y, int x1, int y1, int x2, int y2, int x3, int y3) {
  fl_graphics_driver->loop(x, y, x1, y1, x2, y2, x3, y3);
}

// filled polygons
inline void fl_polygon(int x, int y, int x1, int y1, int x2, int y2) {
  fl_graphics_driver->polygon(x, y, x1, y1, x2, y2);
}
inline void fl_polygon(int x, int y, int x1, int y1, int x2, int y2, int x3, int y3) {
  fl_graphics_driver->polygon(x, y, x1, y1, x2, y2, x3, y3);
}

// draw rectilinear lines, horizontal segment first:
inline void fl_xyline(int x, int y, int x1) {
  fl_graphics_driver->xyline(x, y, x1);
}
inline void fl_xyline(int x, int y, int x1, int y2) {
  fl_graphics_driver->xyline(x, y, x1, y2);
}
inline void fl_xyline(int x, int y, int x1, int y2, int x3) {
  fl_graphics_driver->xyline(x, y, x1, y2, x3);
}

// draw rectilinear lines, vertical segment first:
inline void fl_yxline(int x, int y, int y1) {
  fl_graphics_driver->yxline(x, y, y1);
}
inline void fl_yxline(int x, int y, int y1, int x2) {
  fl_graphics_driver->yxline(x, y, y1, x2);
}
inline void fl_yxline(int x, int y, int y1, int x2, int y3) {
  fl_graphics_driver->yxline(x, y, y1, x2, y3);
}

// circular lines and pie slices (code in fl_arci.C):
inline void fl_arc(int x, int y, int w, int h, double a1, double a2) {
  fl_graphics_driver->arc(x, y, w, h, a1, a2);
}
inline void fl_pie(int x, int y, int w, int h, double a1, double a2) {
  fl_graphics_driver->pie(x, y, w, h, a1, a2);
}
FL_EXPORT void fl_chord(int x, int y, int w, int h, double a1, double a2); // nyi

// scalable drawing code (code in fl_vertex.cxx and fl_arc.cxx):
inline void fl_push_matrix() {
  fl_graphics_driver->push_matrix();
}
inline void fl_pop_matrix() {
  fl_graphics_driver->pop_matrix();
}
inline void fl_scale(double x, double y) {
  fl_graphics_driver->mult_matrix(x, 0, 0, y, 0, 0);
}
inline void fl_scale(double x) {
  fl_graphics_driver->mult_matrix(x, 0, 0, x, 0, 0);
}
inline void fl_translate(double x, double y) {
  fl_graphics_driver->translate(x, y);
}
```

```
inline void fl_rotate(double d) {
  fl_graphics_driver->rotate(d);
}
inline void fl_load_identity() {
  fl_graphics_driver->load_identity();
}
inline void fl_load_matrix(double a, double b, double c, double d, double x, double y) {
  fl_graphics_driver->load_matrix(a, b, c, d, x, y);
}
inline void fl_mult_matrix(double a, double b, double c, double d, double x, double y) {
  fl_graphics_driver->mult_matrix(a, b, c, d, x, y);
}
inline void fl_begin_points() {
  fl_graphics_driver->begin_points();
}
inline void fl_begin_line() {
  fl_graphics_driver->begin_line();
}
inline void fl_begin_loop() {
  fl_graphics_driver->begin_loop();
}
inline void fl_begin_polygon() {
  fl_graphics_driver->begin_polygon();
}
inline void fl_vertex(double x, double y) {
  fl_graphics_driver->vertex(x, y);
}
inline void fl_curve(double X0, double Y0, double X1, double Y1, double X2, double Y2, double X3,
    double Y3) {
  fl_graphics_driver->curve(X0, Y0, X1, Y1, X2, Y2, X3, Y3);
}
inline void fl_arc(double x, double y, double r, double start, double end) {
  fl_graphics_driver->arc(x, y, r, start, end);
}
inline void fl_circle(double x, double y, double r) {
  fl_graphics_driver->circle(x, y, r);
}
inline void fl_end_points() {
  fl_graphics_driver->end_points();
}
inline void fl_end_line() {
  fl_graphics_driver->end_line();
}
inline void fl_end_loop() {
  fl_graphics_driver->end_loop();
}
inline void fl_end_polygon() {
  fl_graphics_driver->end_polygon();
}
inline void fl_begin_complex_polygon() {
  fl_graphics_driver->begin_complex_polygon();
}
inline void fl_gap() {
  fl_graphics_driver->gap();
}
inline void fl_end_complex_polygon() {
  fl_graphics_driver->end_complex_polygon();
}
// get and use transformed positions:
inline double fl_transform_x(double x, double y) {
  return fl_graphics_driver->transform_x(x, y);
}
inline double fl_transform_y(double x, double y) {
  return fl_graphics_driver->transform_y(x, y);
}
inline double fl_transform_dx(double x, double y) {
  return fl_graphics_driver->transform_dx(x, y);
}
inline double fl_transform_dy(double x, double y) {
  return fl_graphics_driver->transform_dy(x, y);
}
inline void fl_transformed_vertex(double xf, double yf) {
  fl_graphics_driver->transformed_vertex(xf, yf);
}

inline void fl_copy_offscreen(int x, int y, int w, int h, Fl_Offscreen pixmap, int srcx, int srcy) {
  fl_graphics_driver->copy_offscreen(x, y, w, h, pixmap, srcx, srcy);
}

FL_EXPORT Fl_Offscreen fl_create_offscreen(int w, int h);
FL_EXPORT void fl_begin_offscreen(Fl_Offscreen b);
FL_EXPORT void fl_end_offscreen(void);
FL_EXPORT void fl_delete_offscreen(Fl_Offscreen bitmap);
FL_EXPORT void fl_rescale_offscreen(Fl_Offscreen &ctx);

/* NOTE: doxygen comments here to avoid triplication in os-specific sources */

```

```
// Fonts:
/*
  Set the current font, which is then used in various drawing routines.
  Implemented and documented in src/fl_draw.cxx
*/
FL_EXPORT void fl_font(Fl_Font face, Fl_Fontsize fsize);

inline Fl_Font fl_font() {
  return fl_graphics_driver->font();
}
inline Fl_Fontsize fl_size() {
  return fl_graphics_driver->size();
}

// Information you can get about the current font:
inline int fl_height() {
  return fl_graphics_driver->height();
}
FL_EXPORT int fl_height(int font, int size);
inline int fl_descent() {
  return fl_graphics_driver->descent();
}
FL_EXPORT double fl_width(const char *txt);

inline double fl_width(const char *txt, int n) {
  return fl_graphics_driver->width(txt, n);
}
inline double fl_width(unsigned int c) {
  return fl_graphics_driver->width(c);
}
FL_EXPORT void fl_text_extents(const char *t, int &dx, int &dy, int &w, int &h);

inline void fl_text_extents(const char *t, int n, int &dx, int &dy, int &w, int &h) {
  fl_graphics_driver->text_extents(t, n, dx, dy, w, h);
}

// font encoding:
// Note: doxygen comments here to avoid duplication for os-specific cases
FL_EXPORT const char *fl_latin1_to_local(const char *t, int n = -1);
FL_EXPORT const char *fl_local_to_latin1(const char *t, int n = -1);
FL_EXPORT const char *fl_mac_roman_to_local(const char *t, int n = -1);
FL_EXPORT const char *fl_local_to_mac_roman(const char *t, int n = -1);
FL_EXPORT float fl_override_scale();

FL_EXPORT void fl_restore_scale(float s);

FL_EXPORT void fl_draw(const char *str, int x, int y);
FL_EXPORT void fl_draw(int angle, const char *str, int x, int y);
inline void fl_draw(const char *str, int n, int x, int y) {
  fl_graphics_driver->draw(str, n, x, y);
}
inline void fl_draw(int angle, const char *str, int n, int x, int y) {
  fl_graphics_driver->draw(angle, str, n, x, y);
}
inline void fl_rtl_draw(const char *str, int n, int x, int y) {
  fl_graphics_driver->rtl_draw(str, n, x, y);
}
FL_EXPORT void fl_measure(const char *str, int &x, int &y, int draw_symbols = 1);
FL_EXPORT void fl_draw(const char *str, int x, int y, int w, int h,
                       Fl_Align align, Fl_Image *img = 0,
                       int draw_symbols = 1, int spacing = 0);
FL_EXPORT void fl_draw(const char *str, int x, int y, int w, int h, Fl_Align align,
                       void (*callthis)(const char *, int, int, int),
                       Fl_Image *img = 0, int draw_symbols = 1, int spacing = 0);

// boxtypes:

FL_EXPORT void fl_frame(const char *s, int x, int y, int w, int h);
FL_EXPORT void fl_frame2(const char *s, int x, int y, int w, int h);
FL_EXPORT void fl_draw_box(Fl_Boxtype, int x, int y, int w, int h, Fl_Color);
FL_EXPORT void fl_draw_box_focus(Fl_Boxtype, int x, int y, int w, int h, Fl_Color, Fl_Color);

// basic GUI objects (check marks, arrows, more to come ...):

// Draw a check mark in the given color inside the bounding box bb.
void fl_draw_check(Fl_Rect bb, Fl_Color col);

// Draw one or more "arrows" (triangles)
FL_EXPORT void fl_draw_arrow(Fl_Rect bb, Fl_Arrow_Type t, Fl_Orientation o, Fl_Color color);

// Draw a potentially small, filled circle
FL_EXPORT void fl_draw_circle(int x, int y, int d, Fl_Color color);

// Draw the full "radio button" of a radio menu entry or radio button
// This requires scheme specific handling (particularly gtk+ scheme)
FL_EXPORT void fl_draw_radio(int x, int y, int d, Fl_Color color);

```

```
// images:

inline void fl_draw_image(const uchar *buf, int X, int Y, int W, int H, int D = 3, int L = 0) {
  fl_graphics_driver->draw_image_general_(buf, X, Y, W, H, D, L);
}

inline void fl_draw_image_mono(const uchar *buf, int X, int Y, int W, int H, int D = 1, int L = 0) {
  fl_graphics_driver->draw_image_mono_general_(buf, X, Y, W, H, D, L);
}

inline void fl_draw_image(Fl_Draw_Image_Cb cb, void *data, int X, int Y, int W, int H, int D = 3) {
  fl_graphics_driver->draw_image(cb, data, X, Y, W, H, D);
}

inline void fl_draw_image_mono(Fl_Draw_Image_Cb cb, void *data, int X, int Y, int W, int H, int D = 1)
    {
  fl_graphics_driver->draw_image_mono(cb, data, X, Y, W, H, D);
}

inline char fl_can_do_alpha_blending() {
  return Fl_Graphics_Driver::default_driver().can_do_alpha_blending();
}

FL_EXPORT uchar *fl_read_image(uchar *p, int X, int Y, int W, int H, int alpha = 0);
FL_EXPORT Fl_RGB_Image *fl_capture_window(Fl_Window *win, int x, int y, int w, int h);

// pixmaps:
FL_EXPORT int fl_draw_pixmap(const char *const *cdata, int x, int y, Fl_Color bg = FL_GRAY);
inline int fl_draw_pixmap(/*const*/ char *const *data, int x, int y, Fl_Color bg = FL_GRAY) {
  return fl_draw_pixmap((const char *const *)data, x, y, bg);
}
FL_EXPORT int fl_measure_pixmap(/*const*/ char *const *data, int &w, int &h);
FL_EXPORT int fl_measure_pixmap(const char *const *cdata, int &w, int &h);

// other:
FL_EXPORT void fl_scroll(int X, int Y, int W, int H, int dx, int dy,
                         void (*draw_area)(void *, int, int, int, int), void *data);
FL_EXPORT const char *fl_shortcut_label(unsigned int shortcut);
FL_EXPORT const char *fl_shortcut_label(unsigned int shortcut, const char **eom);
FL_EXPORT unsigned int fl_old_shortcut(const char *s);
FL_EXPORT void fl_overlay_rect(int x, int y, int w, int h);
FL_EXPORT void fl_overlay_clear();
FL_EXPORT void fl_cursor(Fl_Cursor);
FL_EXPORT void fl_cursor(Fl_Cursor, Fl_Color fg, Fl_Color bg = FL_WHITE);
FL_EXPORT const char *fl_expand_text(const char *from, char *buf, int maxbuf, double maxw,
                                     int &n, double &width, int wrap, int draw_symbols = 0);

// XIM:
FL_EXPORT void fl_set_status(int X, int Y, int W, int H);
FL_EXPORT void fl_set_spot(int font, int size, int X, int Y, int W, int H, Fl_Window *win = 0);
FL_EXPORT void fl_reset_spot(void);


// Symbols (originally from XForms):
FL_EXPORT int fl_draw_symbol(const char *label, int x, int y, int w, int h, Fl_Color);
FL_EXPORT int fl_add_symbol(const char *name, void (*drawit)(Fl_Color c), int scalable);
FL_EXPORT int fl_add_symbol(const char *name, void (*draw_in_rect)(int x, int y, int w, int h,
    Fl_Color c), int scalable);
FL_EXPORT int fl_remove_symbol(const char *name);
#endif
```

14.52 Fl_Export.H

```
/*
 * Windows DLL export .
 *
 * Copyright 1998-2018 by Bill Spitzak and others.
 *
 * This library is free software. Distribution and use rights are outlined in
 * the file "COPYING" which should have been included with this file.  If this
 * file is missing or damaged, see the license at:
 *
 *     https://www.fltk.org/COPYING.php
 *
 * Please see the following page on how to report bugs and issues:
 *
 *     https://www.fltk.org/bugs.php
 */

#ifndef Fl_Export_H
#  define Fl_Export_H

/*
 * The following is used when building DLLs under Windows
 * and when building .so's under unix/linux.
```

```
 */

#  if defined(FL_DLL)
#    ifdef FL_LIBRARY
#      define FL_EXPORT __declspec(dllexport)
#    else
#      define FL_EXPORT __declspec(dllimport)
#    endif /* FL_LIBRARY */
#  elif __GNUC__ >= 4
#    define FL_EXPORT __attribute__ ((visibility ("default")))
#  else
#    define FL_EXPORT
#  endif /* FL_DLL */

#endif /* !Fl_Export_H */
```

14.53 Fl_File_Browser.H

```
//
// FileBrowser definitions.
//
// Copyright 1999-2010 by Michael Sweet.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_File_Browser widget . */

//
// Include necessary header files...
//

#ifndef _Fl_File_Browser_H_
#  define _Fl_File_Browser_H_

#  include "Fl_Browser.H"
#  include "Fl_File_Icon.H"
#  include "filename.H"


//
// Fl_File_Browser class...
//

class FL_EXPORT Fl_File_Browser : public Fl_Browser {

  int           filetype_;
  const char    *directory_;
  uchar         iconsize_;
  const char    *pattern_;
  const char    *errmsg_;

  int   full_height() const override;
  int   item_height(void *) const override;
  int   item_width(void *) const override;
  void  item_draw(void *, int, int, int, int) const override;
  int   incr_height() const override { return (item_height(0) + linespacing()); }

public:
  enum { FILES, DIRECTORIES };

  Fl_File_Browser(int, int, int, int, const char * = 0);
  ~Fl_File_Browser();

  uchar         iconsize() const { return (iconsize_); }
  void          iconsize(uchar s) { iconsize_ = s; redraw(); }

  void  filter(const char *pattern);
  const char    *filter() const { return (pattern_); }
  int           load(const char *directory, Fl_File_Sort_F *sort = fl_numericsort);
  Fl_Fontsize  textsize() const { return Fl_Browser::textsize(); }
  void          textsize(Fl_Fontsize s) { Fl_Browser::textsize(s); iconsize_ = (uchar)(3 * s / 2); }

  int           filetype() const { return (filetype_); }
  void          filetype(int t) { filetype_ = t; }
  void errmsg(const char *emsg);
```

```
  const char* errmsg() const { return errmsg_; }
};

#endif // !_Fl_File_Browser_H_
```

14.54 Fl_File_Chooser.H

```
//
// Fl_File_Chooser dialog for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2024 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//
// ======================================================================
//  DO NOT EDIT FL/Fl_File_Chooser.H and src/Fl_File_Chooser.cxx !!!
// ======================================================================
//  Please use fluid to change src/Fl_File_Chooser.fl interactively
//  and then use fluid to "write code" or edit and use fluid -c .
// ======================================================================
//

// generated by Fast Light User Interface Designer (fluid) version 1.0500

#ifndef Fl_File_Chooser_H
#define Fl_File_Chooser_H
#include <FL/Fl.H>
#include <FL/Fl_Double_Window.H>
#include <FL/Fl_Group.H>
#include <FL/Fl_Choice.H>
#include <FL/Fl_Menu_Button.H>
#include <FL/Fl_Button.H>
#include <FL/Fl_Preferences.H>
#include <FL/Fl_Tile.H>
#include <FL/Fl_File_Browser.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Check_Button.H>
#include <FL/Fl_File_Input.H>
#include <FL/Fl_Return_Button.H>
#include <FL/fl_ask.H>

class FL_EXPORT Fl_File_Chooser {
public:
  enum Type {
    SINGLE    = 0,
    MULTI     = 1,
    CREATE    = 2,
    DIRECTORY = 4
  };
private:
  static Fl_Preferences *prefs_;
  void (*callback_)(Fl_File_Chooser*, void *);
  void *data_;
  char directory_[FL_PATH_MAX];
  char pattern_[FL_PATH_MAX];
  char preview_text_[2048];
  int type_;
  void favoritesButtonCB();
  void favoritesCB(Fl_Widget *w);
  void fileListCB();
  void fileNameCB();
  void newdir();
  static void previewCB(Fl_File_Chooser *fc);
  void showChoiceCB();
  void update_favorites();
  void update_preview();
public:
  Fl_File_Chooser(const char *pathname, const char *pattern, int type_val, const char *title);
private:
  Fl_Double_Window* window;
  inline void cb_window_i(Fl_Double_Window*, void*);
  static void cb_window(Fl_Double_Window*, void*);
  Fl_Choice* showChoice;
  inline void cb_showChoice_i(Fl_Choice*, void*);
  static void cb_showChoice(Fl_Choice*, void*);
  Fl_Menu_Button* favoritesButton;
  inline void cb_favoritesButton_i(Fl_Menu_Button*, void*);
```

```
  static void cb_favoritesButton(Fl_Menu_Button*, void*);
public:
  Fl_Button* newButton;
private:
  inline void cb_newButton_i(Fl_Button*, void*);
  static void cb_newButton(Fl_Button*, void*);
  inline void cb__i(Fl_Tile*, void*);
  static void cb_(Fl_Tile*, void*);
  Fl_File_Browser* fileList;
  inline void cb_fileList_i(Fl_File_Browser*, void*);
  static void cb_fileList(Fl_File_Browser*, void*);
  Fl_Box* errorBox;
  Fl_Box* previewBox;
public:
  Fl_Check_Button* previewButton;
private:
  inline void cb_previewButton_i(Fl_Check_Button*, void*);
  static void cb_previewButton(Fl_Check_Button*, void*);
public:
  Fl_Check_Button* showHiddenButton;
private:
  inline void cb_showHiddenButton_i(Fl_Check_Button*, void*);
  static void cb_showHiddenButton(Fl_Check_Button*, void*);
  Fl_File_Input* fileName;
  inline void cb_fileName_i(Fl_File_Input*, void*);
  static void cb_fileName(Fl_File_Input*, void*);
  Fl_Return_Button* okButton;
  inline void cb_okButton_i(Fl_Return_Button*, void*);
  static void cb_okButton(Fl_Return_Button*, void*);
  Fl_Button* cancelButton;
  inline void cb_cancelButton_i(Fl_Button*, void*);
  static void cb_cancelButton(Fl_Button*, void*);
  Fl_Double_Window* favWindow;
  Fl_File_Browser* favList;
  inline void cb_favList_i(Fl_File_Browser*, void*);
  static void cb_favList(Fl_File_Browser*, void*);
  Fl_Button* favUpButton;
  inline void cb_favUpButton_i(Fl_Button*, void*);
  static void cb_favUpButton(Fl_Button*, void*);
  Fl_Button* favDeleteButton;
  inline void cb_favDeleteButton_i(Fl_Button*, void*);
  static void cb_favDeleteButton(Fl_Button*, void*);
  Fl_Button* favDownButton;
  inline void cb_favDownButton_i(Fl_Button*, void*);
  static void cb_favDownButton(Fl_Button*, void*);
  Fl_Button* favCancelButton;
  inline void cb_favCancelButton_i(Fl_Button*, void*);
  static void cb_favCancelButton(Fl_Button*, void*);
  Fl_Return_Button* favOkButton;
  inline void cb_favOkButton_i(Fl_Return_Button*, void*);
  static void cb_favOkButton(Fl_Return_Button*, void*);
public:
  ~Fl_File_Chooser();
  void callback(void (*cb)(Fl_File_Chooser *, void *), void *d = 0);
  void color(Fl_Color c);
  Fl_Color color();
  int count();
  void directory(const char *d);
  char * directory();
  void filter(const char *p);
  const char * filter();
  int filter_value();
  void filter_value(int f);
  void iconsize(uchar s);
  uchar iconsize();
  void label(const char *l);
  const char * label();
  void ok_label(const char *l);
  const char * ok_label();
  void preview(int e);
  int preview() const { return previewButton->value(); }
private:
  void showHidden(int e);
  void remove_hidden_files();
public:
  void rescan();
  void rescan_keep_filename();
  void show();
  void hide();
  int shown();
  void textcolor(Fl_Color c);
  Fl_Color textcolor();
  void textfont(Fl_Font f);
  Fl_Font textfont();
  void textsize(Fl_Fontsize s);
  Fl_Fontsize textsize();
  void type(int t);
```

```
  int type();
  void * user_data() const;
  void user_data(void *d);
  const char *value(int f = 1);
  void value(const char *filename);
  int visible();
  void position(int x, int y);
  int x() const;
  int y() const;
  int w() const;
  int h() const;
  void size(int w, int h);
  void resize(int x, int y, int w, int h);
  static const char *add_favorites_label;
  static const char *all_files_label;
  static const char *custom_filter_label;
  static const char *existing_file_label;
  static const char *favorites_label;
  static const char *filename_label;
  static const char *filesystems_label;
  static const char *manage_favorites_label;
  static const char *new_directory_label;
  static const char *new_directory_tooltip;
  static const char *preview_label;
  static const char *save_label;
  static const char *show_label;
  static const char *hidden_label;
  static Fl_File_Sort_F *sort;
private:
  Fl_Widget* ext_group;
public:
  Fl_Widget* add_extra(Fl_Widget* gr);
protected:
  void show_error_box(int val);
};
FL_EXPORT char *fl_dir_chooser(const char *message,const char *fname,int relative=0);
FL_EXPORT char *fl_file_chooser(const char *message,const char *pat,const char *fname,int relative=0);
FL_EXPORT void fl_file_chooser_callback(void (*cb)(const char*));
FL_EXPORT void fl_file_chooser_ok_label(const char*l);
#endif
```

14.55 Fl_File_Icon.H

```
//
// Fl_File_Icon definitions.
//
// Copyright 1999-2026 by Michael Sweet.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_File_Icon widget . */

//
// Include necessary header files...
//

#ifndef _Fl_Fl_File_Icon_H_
#  define _Fl_Fl_File_Icon_H_

#  include "Fl.H"


//
// Special color value for the icon color.
//

#  define FL_ICON_COLOR (Fl_Color)0xffffffff
//
// Fl_File_Icon class...
//

class FL_EXPORT Fl_File_Icon {

  static Fl_File_Icon *first_;  // Pointer to first icon/filetype
  Fl_File_Icon  *next_;         // Pointer to next icon/filetype
```

```
  const char    *pattern_;      // Pattern string
  int           type_;          // Match only if directory or file?
  int           num_data_;      // Number of data elements
  int           alloc_data_;    // Number of allocated elements
  short         *data_;         // Icon data

  public:

  enum                          // File types
  {
    ANY,                        // Any kind of file
    PLAIN,                      // Only plain files
    FIFO,                       // Only named pipes
    DEVICE,                     // Only character and block devices
    LINK,                       // Only symbolic links
    DIRECTORY                   // Only directories
  };

  enum                          // Data opcodes
  {
    END,                        // End of primitive/icon
    COLOR,                      // Followed by color value (2 shorts)
    LINE,                       // Start of line
    CLOSEDLINE,                 // Start of closed line
    POLYGON,                    // Start of polygon
    OUTLINEPOLYGON,             // Followed by outline color (2 shorts)
    VERTEX                      // Followed by scaled X,Y
  };

  Fl_File_Icon(const char *p, int t, int nd = 0, short *d = 0);
  ~Fl_File_Icon();

  short         *add(short d);

  short         *add_color(Fl_Color c)
                { short *d = add((short)COLOR); add((short)(c » 16)); add((short)c); return (d); }

  short         *add_vertex(int x, int y)
                { short *d = add((short)VERTEX); add((short)x); add((short)y); return (d); }

  short         *add_vertex(float x, float y)
                { short *d = add((short)VERTEX); add((short)(x * 10000.0));
                  add((short)(y * 10000.0)); return (d); }

  void          clear() { num_data_ = 0; }

  void          draw(int x, int y, int w, int h, Fl_Color ic, int active = 1);

  void          label(Fl_Widget *w);

  static void   labeltype(const Fl_Label *o, int x, int y, int w, int h, Fl_Align a);
  void          load(const char *f);
  int           load_fti(const char *fti);
  int           load_image(const char *i);

  Fl_File_Icon  *next() const { return next_; }

  const char    *pattern() const { return pattern_; }

  int           size() const { return num_data_; }

  int           type() const { return type_; }

  short         *value() const { return data_; }

  static Fl_File_Icon *find(const char *filename, int filetype = ANY);

  static Fl_File_Icon *first() { return first_; }
  static void   load_system_icons(void);
};

#endif // !_Fl_Fl_File_Icon_H_
```

14.56 Fl_File_Input.H

```
//
// File_Input header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2026 by Bill Spitzak and others.
// Original version Copyright 1998 by Curtis Edwards.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
```

```
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_File_Input widget . */

#ifndef Fl_File_Input_H
#  define Fl_File_Input_H

#  include <FL/Fl_Input.H>

class FL_EXPORT Fl_File_Input : public Fl_Input {

  char          ok_entry_;
  uchar         down_box_;
  short         buttons_[200];
  short         pressed_;

  void          draw_buttons();
  int           handle_button(int event);
  void          update_buttons();

public:

  Fl_File_Input(int X, int Y, int W, int H, const char *L=0);

  int           handle(int event) override;

protected:
  void          draw() override;

public:
  Fl_Boxtype    down_box() const { return (Fl_Boxtype)down_box_; }
  void          down_box(Fl_Boxtype b) { down_box_ = (uchar)b; }

  int           value(const char *str);
  int           value(const char *str, int len);

  const char    *value() const { return Fl_Input_::value(); }
};

#endif // !Fl_File_Input_H
```

14.57 Fl_Fill_Dial.H

```
//
// Filled dial header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Fill_Dial widget . */

#ifndef Fl_Fill_Dial_H
#define Fl_Fill_Dial_H

#include "Fl_Dial.H"

class FL_EXPORT Fl_Fill_Dial : public Fl_Dial {
public:
  Fl_Fill_Dial(int X,int Y,int W,int H, const char *L = NULL);
};

#endif
```

14.58 Fl_Fill_Slider.H

```
//
// Filled slider header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Fill_Slider widget . */

#ifndef Fl_Fill_Slider_H
#define Fl_Fill_Slider_H

#include "Fl_Slider.H"
class FL_EXPORT Fl_Fill_Slider : public Fl_Slider {
public:
  Fl_Fill_Slider(int X,int Y,int W,int H,const char *L=0);
};

#endif
```

14.59 Fl_Flex.H

```
//
// Fl_Flex widget header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 2020 by Karsten Pedersen
// Copyright 2022-2024 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef Fl_Flex_H
#define Fl_Flex_H

#include <FL/Fl_Group.H>

class FL_EXPORT Fl_Flex : public Fl_Group {

  int margin_left_;           // left margin
  int margin_top_;            // top margin
  int margin_right_;          // right margin
  int margin_bottom_;         // bottom margin
  int gap_;                   // gap between widgets
  int fixed_size_size_;       // number of fixed size widgets in array
  int fixed_size_alloc_;      // allocated size of fixed size array
  Fl_Widget **fixed_size_;    // array of fixed size widgets
  bool need_layout_;          // true if layout needs to be calculated

public:

  enum { // values for type(int)
    VERTICAL   = 0,
    HORIZONTAL = 1,
    COLUMN     = 0,
    ROW        = 1
  };

  // FLTK standard constructor
  Fl_Flex(int X, int Y, int W, int H, const char *L = 0);

  // original Fl_Flex constructors:
  // backwards compatible if direction enums { ROW | COLUMN } are used

  Fl_Flex(int direction);
```

```
  Fl_Flex(int w, int h, int direction);
  Fl_Flex(int x, int y, int w, int h, int direction);

  virtual ~Fl_Flex();

  virtual void end();
  void resize(int x, int y, int w, int h) override;

  void fixed(Fl_Widget &w, int size) {
    fixed(&w, size);
  }

  void fixed(Fl_Widget *w, int size);
  int fixed(Fl_Widget *w) const;

protected:

  void init(int t = VERTICAL);

  virtual int alloc_size(int size) const;

  void on_remove(int) override;
  void draw() override;

public:

  void need_layout(int set) {
    if (set) need_layout_ = true;
    else need_layout_ = false;
  }

  bool need_layout() const {
    return need_layout_;
  }

  int margin() const { return margin_left_; }

  int margin(int *left, int *top, int *right, int *bottom) const {
    if (left) *left = margin_left_;
    if (top) *top = margin_top_;
    if (right) *right = margin_right_;
    if (bottom) *bottom = margin_bottom_;
    if (margin_left_ == margin_top_ && margin_top_ == margin_right_ && margin_right_ ==
    margin_bottom_)
      return 1;
    return 0;
  }

  void margin(int m, int g = -1) {
    if (m < 0)
      m = 0;
    margin_left_ = margin_top_ = margin_right_ = margin_bottom_ = m;
    if (g >= 0)
      gap_ = g;
    need_layout(1);
  }

  void margin(int left, int top, int right, int bottom) {
    margin_left_   = left < 0 ? 0 : left;
    margin_top_    = top < 0 ? 0 : top;
    margin_right_  = right < 0 ? 0 : right;
    margin_bottom_ = bottom < 0 ? 0 : bottom;
    need_layout(1);
  }

  int gap() const {
    return gap_;
  }

  void gap(int g) {
    gap_ = g < 0 ? 0 : g;
    need_layout(1);
  }

  int horizontal() const {
    return type() == Fl_Flex::HORIZONTAL ? 1 : 0;
  }

  // Calculate the layout of the widget and redraw it.
  void layout();

  int spacing() const {
    return gap_;
  }

  void spacing(int i) {
    gap(i);
```

```
    need_layout(1);
  }

};

#endif // Fl_Flex_H
```

14.60 Fl_Float_Input.H

```
//
// Floating point input header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2011 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Float_Input widget . */

#ifndef Fl_Float_Input_H
#define Fl_Float_Input_H

#include "Fl_Input.H"

class FL_EXPORT Fl_Float_Input : public Fl_Input {
public:
  Fl_Float_Input(int X,int Y,int W,int H,const char *l = 0);
};

#endif
```

14.61 Fl_FormsBitmap.H

```
//
// Forms bitmap header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_FormsBitmap widget . */

#ifndef Fl_FormsBitmap_H
#define Fl_FormsBitmap_H

#include "Fl_Bitmap.H"

class FL_EXPORT Fl_FormsBitmap : public Fl_Widget {
    Fl_Bitmap *b;
protected:
    void draw() override;
public:
    Fl_FormsBitmap(Fl_Boxtype, int, int, int, int, const char * = 0);
    void set(int W, int H, const uchar *bits);
    void bitmap(Fl_Bitmap *B) {b = B;}
    Fl_Bitmap *bitmap() const {return b;}
};

#endif
```

14.62 Fl_FormsPixmap.H

```
//
// Forms pixmap header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_FormsPixmap widget . */

#ifndef Fl_FormsPixmap_H
#define Fl_FormsPixmap_H

#include "Fl_Pixmap.H"

class FL_EXPORT Fl_FormsPixmap : public Fl_Widget {
    Fl_Pixmap *b;
protected:
    void draw() override;
public:
    Fl_FormsPixmap(Fl_Boxtype t, int X, int Y, int W, int H, const char *L= 0);

    void set(/*const*/char * const * bits);

    void Pixmap(Fl_Pixmap *B) {b = B;}

    Fl_Pixmap *Pixmap() const {return b;}
};

#endif
```

14.63 Fl_Free.H

```
//
// Forms free header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Free widget . */

#ifndef Fl_Free_H
#define Fl_Free_H

#ifndef Fl_Widget_H
#include "Fl_Widget.H"
#endif

#define FL_NORMAL_FREE          1
#define FL_SLEEPING_FREE        2
#define FL_INPUT_FREE           3
#define FL_CONTINUOUS_FREE      4
#define FL_ALL_FREE             5
typedef int (*FL_HANDLEPTR)(Fl_Widget *, int , float, float, char);

class FL_EXPORT Fl_Free : public Fl_Widget {
    FL_HANDLEPTR hfunc;
    static void step(void *);
protected:
    void draw() override;
public:
    int handle(int e) override;
```

```
  Fl_Free(uchar t,int X,int Y,int W,int H,const char *L,FL_HANDLEPTR hdl);
  ~Fl_Free();
};

// old event names for compatibility:
#define FL_MOUSE        FL_DRAG
#define FL_DRAW         100
#define FL_STEP         101
#define FL_FREEMEM      102
#define FL_FREEZE       103
#define FL_THAW         104
#endif
```

14.64 Fl_GIF_Image.H

```
//
// GIF image header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2026 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_GIF_Image widget . */

#ifndef Fl_GIF_Image_H
#define Fl_GIF_Image_H
#  include "Fl_Pixmap.H"

class FL_EXPORT Fl_GIF_Image : public Fl_Pixmap {

public:

  Fl_GIF_Image(const char* filename);
  // deprecated constructor w/o length (for backwards compatibility)
  Fl_GIF_Image(const char* imagename, const unsigned char *data);
  // constructor with length (since 1.4.0)
  Fl_GIF_Image(const char* imagename, const unsigned char *data, const size_t length);

  static bool is_animated(const char *name_);
  static bool animate;

protected:

  // Protected constructors needed for animated GIF support through Fl_Anim_GIF_Image.
  Fl_GIF_Image(const char* filename, bool anim);
  Fl_GIF_Image(const char* imagename, const unsigned char *data, const size_t length, bool anim);
  // Protected default constructor needed for Fl_Anim_GIF_Image.
  Fl_GIF_Image();

  void load_gif_(class Fl_Image_Reader &rdr, bool anim=false);

  void load(const char* filename, bool anim);
  void load(const char* imagename, const unsigned char *data, const size_t length, bool anim);

  // Internal structure to "glue" animated GIF support into Fl_GIF_Image.
  // This data is passed during decoding to the Fl_Anim_GIF_Image class.
  struct GIF_FRAME {
    int ifrm, width, height, x, y, w, h,
        clrs, bkgd, trans,
        dispose, delay;
    const uchar *bptr;
    const struct CPAL {
      uchar r, g, b;
    } *cpal;
    GIF_FRAME(int frame, const uchar* data) : ifrm(frame), bptr(data) {}
    GIF_FRAME(int frame, int W, int H, int fx, int fy, int fw, int fh, const uchar* data) :
      ifrm(frame), width(W), height(H), x(fx), y(fy), w(fw), h(fh), bptr(data) {}
    void disposal(int mode, int time) { dispose = mode; this->delay = time; }
    void colors(int nclrs, int bg, int tp) { clrs = nclrs; bkgd = bg; trans = tp; }
  };

  // Internal virtual methods, which are called during decoding to pass data
  // to the Fl_Anim_GIF_Image class.
  virtual void on_frame_data(GIF_FRAME &) {}
  virtual void on_extension_data(GIF_FRAME &) {}
```

```

private:

  void lzw_decode(Fl_Image_Reader &rdr, uchar *Image, int Width, int Height, int CodeSize, int ColorMapSize, int Interlace);
};

#endif
```

14.65 Fl_Gl_Window.H

```
//
// OpenGL header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2026 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Gl_Window widget . */

#ifndef Fl_Gl_Window_H
#define Fl_Gl_Window_H

#include "Fl_Window.H"

class Fl_Gl_Choice; // structure to hold result of glXChooseVisual
class Fl_Gl_Window_Driver;
class FL_EXPORT Fl_Gl_Window : public Fl_Window {
  friend class Fl_Gl_Window_Driver;
  Fl_Gl_Window_Driver *pGlWindowDriver;

  int mode_;
  const int *alist;
  Fl_Gl_Choice *g;
  GLContext context_;
  char valid_f_;
  char damage1_; // damage() of back buffer
  virtual void draw_overlay();
  void init();

  void *overlay;

  static int can_do(int, const int *);
  int mode(int, const int *);
  static int gl_plugin_linkage();
protected:
  void draw_begin();
  void draw() override;
  void draw_end();

public:
  void show() override;
  void show(int a, char **b) {Fl_Window::show(a,b);}
  void flush() override;
  void hide() override;
  void resize(int,int,int,int) override;
  int handle(int) override;

  char valid() const {return valid_f_ & 1;}
  void valid(char v) {if (v) valid_f_ |= 1; else valid_f_ &= 0xfe;}
  void invalidate();

  char context_valid() const {return valid_f_ & 2;}
  void context_valid(char v) {if (v) valid_f_ |= 2; else valid_f_ &= 0xfd;}

  static int can_do(int m) {return can_do(m,0);}
  static int can_do(const int *m) {return can_do(0, m);}
  int can_do() const { return can_do(mode_,alist); }
  Fl_Mode mode() const {return (Fl_Mode)mode_;}
  int mode(int a) {return mode(a,0);}
  int mode(const int *a) {return mode(0, a);}
  GLContext context() const {return context_;}
  void context(GLContext, int destroy_flag = 0);
  void make_current();
  void swap_buffers();
```

```
  void swap_interval(int);
  int swap_interval() const;
  void ortho();

  int can_do_overlay();
  void redraw_overlay();
  void hide_overlay();
  void make_overlay_current();

  // Note: Doxygen docs in Fl_Widget.H to avoid redundancy.
  Fl_Gl_Window* as_gl_window() override { return this; }
  Fl_Gl_Window const* as_gl_window() const override { return this; }

  float pixels_per_unit();
  int pixel_w() { return int(pixels_per_unit() * w() + 0.5f); }
  int pixel_h() { return int(pixels_per_unit() * h() + 0.5f); }

  ~Fl_Gl_Window();
  Fl_Gl_Window(int W, int H, const char *l=0) : Fl_Window(W,H,l) {init();}
  Fl_Gl_Window(int X, int Y, int W, int H, const char *l=0)
    : Fl_Window(X,Y,W,H,l) {init();}
};

#endif // Fl_Gl_Window_H
```

14.66 Fl_Graphics_Driver.H

```
//
// Declaration of class Fl_Graphics_Driver for the Fast Light Tool Kit (FLTK).
//
// Copyright 2010-2026 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef FL_GRAPHICS_DRIVER_H
#define FL_GRAPHICS_DRIVER_H

#include <FL/Fl_Device.H>
#include <FL/Fl_Image.H>
#include <FL/Fl_Bitmap.H>
#include <FL/Fl_Pixmap.H>
#include <FL/Fl_RGB_Image.H>

#include <stack>

class Fl_Graphics_Driver;
class Fl_Font_Descriptor;
class Fl_Image_Surface;
FL_EXPORT extern Fl_Graphics_Driver *fl_graphics_driver;

typedef void (*Fl_Draw_Image_Cb)(void* data,int x,int y,int w,uchar* buf);

struct Fl_Fontdesc;
typedef struct _PangoFontDescription PangoFontDescription;


class FL_EXPORT Fl_Graphics_Driver {
  friend class Fl_Surface_Device;
  friend class Fl_Pixmap;
  friend class Fl_Bitmap;
  friend class Fl_RGB_Image;
  friend class Fl_SVG_Image;
  friend void fl_draw_image(const uchar* buf, int X,int Y,int W,int H, int D, int L);
  friend void fl_draw_image_mono(const uchar* buf, int X,int Y,int W,int H, int D, int L);
  friend void fl_draw_image_mono(Fl_Draw_Image_Cb cb, void* data, int X,int Y,int W,int H, int D);
  friend void fl_draw_image(Fl_Draw_Image_Cb cb, void* data, int X,int Y,int W,int H, int D);
  friend void fl_copy_offscreen(int x, int y, int w, int h, Fl_Offscreen pixmap, int srcx, int srcy);
  friend int fl_convert_pixmap(const char*const* cdata, uchar* out, Fl_Color bg);
  friend FL_EXPORT int fl_draw_pixmap(const char*const* cdata, int x, int y, Fl_Color bg);
  friend FL_EXPORT void gl_start();
  /* ============== Implementation note about image drawing ========================
   A graphics driver can implement up to 6 virtual member functions to draw images:
   virtual void draw_pixmap(Fl_Pixmap *pxm,int XP, int YP, int WP, int HP, int cx, int cy)
   virtual void draw_bitmap(Fl_Bitmap *bm,int XP, int YP, int WP, int HP, int cx, int cy)
   virtual void draw_rgb(Fl_RGB_Image *rgb,int XP, int YP, int WP, int HP, int cx, int cy)
and
```

```
  virtual void draw_fixed(Fl_Pixmap *pxm,int XP, int YP, int WP, int HP, int cx, int cy)
  virtual void draw_fixed(Fl_Bitmap *bm,int XP, int YP, int WP, int HP, int cx, int cy)
  virtual void draw_fixed(Fl_RGB_Image *rgb,int XP, int YP, int WP, int HP, int cx, int cy)
  - The 1st group of functions is used when the driver can directly map the image data,
  sized at data_w() x data_h(), to the image drawing area, sized at w()*scale x h()*scale
  where scale is the current GUI scale factor.
  - If the driver does not support such scale-and-draw operation for a given image type,
  it should implement the draw_fixed() function which is called by the library after the
  image has been internally resized to the drawing size and cached.
  - The platform-independent Fl_Graphics_Driver class implements the 1st group of functions.
  Each resizes the image, caches it, and calls the platform-specific implementation of
  draw_fixed(image-class *,....) with the cached image.
  - Consider an image object, say from class Fl_RGB_Image. Fl_RGB_Image::draw()
  calls the virtual member function draw_rgb(Fl_RGB_Image *,....). If Fl_XXX_Graphics_Driver
  re-implements this function, this code runs and is expected to draw the image
  adequately scaled to its drawing size. If Fl_XXX_Graphics_Driver does not re-implement
  this function, Fl_Graphics_Driver::draw_rgb(Fl_RGB_Image *,....) runs. It internally
  resizes the image, caches it, and calls Fl_XXX_Graphics_Driver::draw_fixed(Fl_RGB_Image *,....)
  that draws the image from its cached form which already has the adequate size.
  - Some drivers implement, for a given image class, only the function of the 1st group or
  only draw_fixed() as in these examples:
  - Fl_Quartz_Graphics_Driver implements only draw_rgb(Fl_RGB_Image *,....) because it
  can perform the scale-and-draw operation whatever the RGB image and the required scaling.
  - Fl_Xlib_Graphics_Driver implements only draw_fixed(Fl_Pixmap *,....). The library
  takes care of resizing and caching the Pixmap to the adequate drawing size.
  - Some drivers implement, for a given image class, the function of both groups, e.g. :
  Fl_GDI_Graphics_Driver implements both draw_rgb(Fl_RGB_Image *,....) and
  draw_fixed(Fl_RGB_Image *,....) because scale-and-draw may require function Alphablend()
  from MSIMG32.DLL. In the absence of that, the draw_rgb() implementation calls
  Fl_Graphics_Driver::draw_rgb() which runs Fl_GDI_Graphics_Driver::draw_fixed(Fl_RGB_Image*,...).

  Graphics drivers also implement cache(Fl_Pixmap*), cache(Fl_Bitmap*) and cache(Fl_RGB_Image*)
  to compute the cached form of all image types, and uncache(Fl_RGB_Image *,...),
  uncache_pixmap(fl_uintptr_t) and delete_bitmask(fl_uintptr_t) to destroy cached image forms.
  Graphics drivers that use the mask_ variable of class Fl_Pixmap to cache an Fl_Pixmap
  object also reimplement the uchar **Fl_Graphics_Driver::mask_bitmap() member function.
  */
private:
  virtual void draw_fixed(Fl_Pixmap *pxm,int XP, int YP, int WP, int HP, int cx, int cy);
  virtual void draw_fixed(Fl_Bitmap *bm,int XP, int YP, int WP, int HP, int cx, int cy);
  virtual void draw_fixed(Fl_RGB_Image *rgb,int XP, int YP, int WP, int HP, int cx, int cy);
  // the default implementation of make_unused_color_() is most probably enough
  virtual void make_unused_color_(unsigned char &r, unsigned char &g, unsigned char &b, int
color_count, void **data);
  // some platforms may need to reimplement this
  virtual void set_current_();
  void draw_image_general_(const uchar *buf, int X, int Y, int W, int H, int D, int L);
  void draw_image_mono_general_(const uchar *buf, int X, int Y, int W, int H, int D, int L);
  float scale_; // scale between FLTK and drawing coordinates: drawing = FLTK * scale_
public:
  static Fl_Graphics_Driver *newMainGraphicsDriver();
  struct matrix {double a, b, c, d, x, y;};
  typedef enum {
    NATIVE = 1,
    PRINTER = 2
  } driver_feature;

protected:
  int fl_clip_state_number;
  static const matrix m0;
  Fl_Font font_;
  Fl_Fontsize size_;
  Fl_Color color_;
  std::stack<matrix> matrix_stack;
  matrix m;
  int n;
  int gap_;
  struct callback_data {
    const uchar *data;
    int D, LD;
  };
  static const int dashes_flat[5][7];
  static const double dashes_cap[5][7];
  static void draw_image_cb(void *data, int x, int y, int w, uchar *buf);
  enum SHAPE {NONE=0, LINE, LOOP, POLYGON, POINTS, COMPLEX_POLYGON} what;
  std::stack<Fl_Region> rstack;
  Fl_Font_Descriptor *font_descriptor_;
  int p_size;
  typedef struct { float x; float y; } XPOINT;
  XPOINT *xpoint;
  virtual void global_gc();
  virtual void cache(Fl_Pixmap *img);
  virtual void cache(Fl_Bitmap *img);
  virtual void cache(Fl_RGB_Image *img);
  virtual void uncache(Fl_RGB_Image *img, fl_uintptr_t &id_, fl_uintptr_t &mask_);
  // --- implementation is in src/drivers/xxx/Fl_xxx_Graphics_Driver_image.cxx
  virtual void draw_image(const uchar* buf, int X,int Y,int W,int H, int D=3, int L=0);
```

```
  virtual void draw_image_mono(const uchar* buf, int X,int Y,int W,int H, int D=1, int L=0);
  virtual void draw_image(Fl_Draw_Image_Cb cb, void* data, int X,int Y,int W,int H, int D=3);
  virtual void draw_image_mono(Fl_Draw_Image_Cb cb, void* data, int X,int Y,int W,int H, int D=1);
  virtual void draw_rgb(Fl_RGB_Image * rgb,int XP, int YP, int WP, int HP, int cx, int cy);
  virtual void draw_pixmap(Fl_Pixmap * pxm,int XP, int YP, int WP, int HP, int cx, int cy);
  virtual void draw_bitmap(Fl_Bitmap *bm, int XP, int YP, int WP, int HP, int cx, int cy);
  virtual void copy_offscreen(int x, int y, int w, int h, Fl_Offscreen pixmap, int srcx, int srcy);

  static void change_image_size(Fl_Image *img, int W, int H) {
    img->w(W);
    img->h(H);
  }
  // Support function for image drawing
  virtual void uncache_pixmap(fl_uintptr_t p);
  // accessor functions to protected image members
  int start_image(Fl_Image *img, int XP, int YP, int WP, int HP, int &cx, int &cy,
                int &X, int &Y, int &W, int &H);
  static fl_uintptr_t* id(Fl_RGB_Image *rgb) {return &(rgb->id_);}
  static fl_uintptr_t* id(Fl_Pixmap *pm) {return &(pm->id_);}
  static fl_uintptr_t* id(Fl_Bitmap *bm) {return &(bm->id_);}
  static fl_uintptr_t* mask(Fl_RGB_Image *rgb) {return &(rgb->mask_);}
  static fl_uintptr_t* mask(Fl_Pixmap *pm) {return &(pm->mask_);}
  static void cache_w_h(Fl_Pixmap *pm, int*& pwidth, int*& pheight) {
    pwidth = &(pm->cache_w_);
    pheight = &(pm->cache_h_);
  }
  static void cache_w_h(Fl_Bitmap *bm, int*& pwidth, int*& pheight) {
    pwidth = &(bm->cache_w_);
    pheight = &(bm->cache_h_);
  }
  static void cache_w_h(Fl_RGB_Image *rgb, int*& pwidth, int*& pheight) {
    pwidth = &(rgb->cache_w_);
    pheight = &(rgb->cache_h_);
  }
  static Fl_Offscreen get_offscreen_and_delete_image_surface(Fl_Image_Surface*);
  static void draw_empty(Fl_Image* img, int X, int Y) {img->draw_empty(X, Y);}

  Fl_Graphics_Driver();
  virtual void cache_size(Fl_Image *img, int &width, int &height);
  void cache_size_finalize(Fl_Image *img, int &width, int &height);
  static unsigned need_pixmap_bg_color;
public:
  virtual ~Fl_Graphics_Driver();
  static Fl_Graphics_Driver &default_driver();
  // support of "complex shapes"
  void push_matrix();
  void pop_matrix();
  void load_identity();
  void load_matrix(double a, double b, double c, double d, double x, double y);
  void mult_matrix(double a, double b, double c, double d, double x, double y);
  void rotate(double d);
  void translate(double x,double y);
  double transform_x(double x, double y) const;
  double transform_y(double x, double y) const;
  double transform_dx(double x, double y) const;
  double transform_dy(double x, double y) const;
  inline Fl_Font_Descriptor *font_descriptor() const { return font_descriptor_;}
  inline void font_descriptor(Fl_Font_Descriptor *d) { font_descriptor_ = d;}
  float scale() const { return scale_; }
  virtual void scale(float f);
  virtual char can_do_alpha_blending();
  virtual void point(int x, int y);
  virtual void rect(int x, int y, int w, int h);
  virtual void focus_rect(int x, int y, int w, int h);
  virtual void rectf(int x, int y, int w, int h);
  virtual void _rbox(int fill, int x, int y, int w, int h, int r);
  virtual void rounded_rect(int x, int y, int w, int h, int r);
  virtual void rounded_rectf(int x, int y, int w, int h, int r);
  // the default implementation is most likely enough
  virtual void colored_rectf(int x, int y, int w, int h, uchar r, uchar g, uchar b);
  virtual void line(int x, int y, int x1, int y1);
  virtual void line(int x, int y, int x1, int y1, int x2, int y2);
  virtual void xyline(int x, int y, int x1);
  virtual void xyline(int x, int y, int x1, int y2);
  virtual void xyline(int x, int y, int x1, int y2, int x3);
  virtual void yxline(int x, int y, int y1);
  virtual void yxline(int x, int y, int y1, int x2);
  virtual void yxline(int x, int y, int y1, int x2, int y3);
  virtual void loop(int x0, int y0, int x1, int y1, int x2, int y2);
  virtual void loop(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3);
  virtual void polygon(int x0, int y0, int x1, int y1, int x2, int y2);
  virtual void polygon(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3);
  // --- clipping
  virtual void push_clip(int x, int y, int w, int h);
  virtual int clip_box(int x, int y, int w, int h, int &X, int &Y, int &W, int &H);
  virtual int not_clipped(int x, int y, int w, int h);
  virtual void push_no_clip();                  // has default implementation
```

```
  virtual void pop_clip();                        // has default implementation
  virtual Fl_Region clip_region();                // has default implementation
  virtual void clip_region(Fl_Region r);          // has default implementation
  virtual void restore_clip();
  virtual void begin_points();
  virtual void begin_line();
  virtual void begin_loop();
  virtual void begin_polygon();
  virtual void begin_complex_polygon();
  virtual void transformed_vertex(double xf, double yf);
  virtual void transformed_vertex0(float x, float y);
  virtual void vertex(double x, double y);
  virtual void end_points();
  virtual void end_line();
  virtual void end_loop();
  virtual void fixloop();
  virtual void end_polygon();
  virtual void end_complex_polygon();
  // default implementation is most probably enough
  virtual bool can_fill_non_convex_polygon() { return true; }
  virtual void gap();
  virtual void circle(double x, double y, double r);
  virtual void arc(double x, double y, double r, double start, double end);
  virtual void arc(int x, int y, int w, int h, double a1, double a2);
  virtual void pie(int x, int y, int w, int h, double a1, double a2);
  // To support fl_draw_circle(int x, int y, int d, Fl_Color color),
  // the default implementation is most probably enough.
  virtual void draw_circle(int x, int y, int d, Fl_Color c);
  virtual void curve(double X0, double Y0, double X1, double Y1, double X2, double Y2, double X3,
double Y3);
  virtual void line_style(int style, int width=0, char* dashes=0);
  virtual void color(Fl_Color c);
  virtual void set_color(Fl_Color i, unsigned int c);
  virtual void free_color(Fl_Color i, int overlay);
  virtual Fl_Color color();
  virtual void color(uchar r, uchar g, uchar b);
  virtual void draw(const char *str, int nChars, int x, int y);
  virtual void draw(const char *str, int nChars, float x, float y);
  virtual void draw(int angle, const char *str, int nChars, int x, int y);
  virtual void rtl_draw(const char *str, int nChars, int x, int y);
  virtual int has_feature(driver_feature feature);
  virtual void font(Fl_Font face, Fl_Fontsize fsize);
  virtual Fl_Font font();
  virtual Fl_Fontsize size();
  virtual double width(const char *str, int nChars);
  virtual double width(unsigned int c);
  virtual void text_extents(const char*, int n, int& dx, int& dy, int& w, int& h);
  virtual int height();
  virtual int descent();
  virtual void gc(void*);
  virtual void *gc(void);
  virtual uchar **mask_bitmap();
  // default implementation may be enough
  virtual float scale_font_for_PostScript(Fl_Font_Descriptor *desc, int s);
  // default implementation may be enough
  virtual float scale_bitmap_for_PostScript();
  // each platform implements these 3 functions its own way
  virtual void add_rectangle_to_region(Fl_Region r, int x, int y, int w, int h);
  virtual Fl_Region XRectangleRegion(int x, int y, int w, int h);
  virtual void XDestroyRegion(Fl_Region r);
  virtual const char* get_font_name(Fl_Font fnum, int* ap);
  virtual int get_font_sizes(Fl_Font fnum, int*& sizep);
  virtual Fl_Font set_fonts(const char *name);
  virtual Fl_Fontdesc* calc_fl_fonts(void);
  virtual unsigned font_desc_size();
  virtual const char *font_name(int num);
  virtual void font_name(int num, const char *name);
  // Defaut implementation may be enough
  virtual void overlay_rect(int x, int y, int w , int h);
  virtual float override_scale();
  virtual void restore_scale(float);
  virtual PangoFontDescription* pango_font_description() { return NULL; }
  virtual void antialias(int state);
  virtual int antialias();
  virtual void delete_bitmask(fl_uintptr_t bm);
};

#endif // FL_GRAPHICS_DRIVER_H
```

14.67 Fl_Grid.H File Reference

Fl_Grid container widget.

```
#include <FL/Fl_Group.H>
#include <FL/Fl_Rect.H>
```

Classes

- class Fl_Grid::Cell
- class Fl_Grid

  Fl_Grid is a container (layout) widget with multiple columns and rows.

Typedefs

- typedef unsigned short **Fl_Grid_Align**

  Fl_Grid type for child widget alignment control.

Variables

- const Fl_Grid_Align **FL_GRID_BOTTOM** = 0x0002

  Align the widget at the bottom of the cell.
- const Fl_Grid_Align **FL_GRID_BOTTOM_LEFT** = FL_GRID_BOTTOM | FL_GRID_LEFT
- const Fl_Grid_Align **FL_GRID_BOTTOM_RIGHT** = FL_GRID_BOTTOM | FL_GRID_RIGHT
- const Fl_Grid_Align **FL_GRID_CENTER** = 0x0000

  Align the widget in the middle of the cell (default).
- const Fl_Grid_Align **FL_GRID_FILL** = 0x0030

  Stretch the widget in both directions to fill the cell.
- const Fl_Grid_Align **FL_GRID_HORIZONTAL** = 0x0010

  Stretch the widget horizontally to fill the cell.
- const Fl_Grid_Align **FL_GRID_LEFT** = 0x0004

  Align the widget at the left side of the cell.
- const Fl_Grid_Align **FL_GRID_PROPORTIONAL** = 0x0040

  Stretch the widget proportionally.
- const Fl_Grid_Align **FL_GRID_RIGHT** = 0x0008

  Align the widget at the right side of the cell.
- const Fl_Grid_Align **FL_GRID_TOP** = 0x0001

  Align the widget at the top of the cell.
- const Fl_Grid_Align **FL_GRID_TOP_LEFT** = FL_GRID_TOP | FL_GRID_LEFT
- const Fl_Grid_Align **FL_GRID_TOP_RIGHT** = FL_GRID_TOP | FL_GRID_RIGHT
- const Fl_Grid_Align **FL_GRID_VERTICAL** = 0x0020

  Stretch the widget vertically to fill the cell.

14.67.1 Detailed Description

Fl_Grid container widget.

14.68 Fl_Grid.H

Go to the documentation of this file.

```
00001 //
00002 // Fl_Grid widget header for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 2021-2022 by Albrecht Schlosser.
00005 // Copyright 2022-2026 by Bill Spitzak and others.
00006 //
00007 // This library is free software. Distribution and use rights are outlined in
00008 // the file "COPYING" which should have been included with this file.  If this
00009 // file is missing or damaged, see the license at:
00010 //
00011 //     https://www.fltk.org/COPYING.php
```

```
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef _FL_FL_GRID_H_
#define _FL_FL_GRID_H_

#include <FL/Fl_Group.H>
#include <FL/Fl_Rect.H>

typedef unsigned short Fl_Grid_Align;

const Fl_Grid_Align  FL_GRID_CENTER          = 0x0000;

const Fl_Grid_Align  FL_GRID_TOP             = 0x0001;

const Fl_Grid_Align  FL_GRID_BOTTOM          = 0x0002;

const Fl_Grid_Align  FL_GRID_LEFT            = 0x0004;

const Fl_Grid_Align  FL_GRID_RIGHT           = 0x0008;

const Fl_Grid_Align  FL_GRID_HORIZONTAL      = 0x0010;

const Fl_Grid_Align  FL_GRID_VERTICAL        = 0x0020;

const Fl_Grid_Align  FL_GRID_FILL            = 0x0030;

const Fl_Grid_Align  FL_GRID_PROPORTIONAL    = 0x0040;

const Fl_Grid_Align  FL_GRID_TOP_LEFT        =  FL_GRID_TOP |  FL_GRID_LEFT;
const Fl_Grid_Align  FL_GRID_TOP_RIGHT       =  FL_GRID_TOP |  FL_GRID_RIGHT;
const Fl_Grid_Align  FL_GRID_BOTTOM_LEFT     =  FL_GRID_BOTTOM |  FL_GRID_LEFT;
const Fl_Grid_Align  FL_GRID_BOTTOM_RIGHT    =  FL_GRID_BOTTOM |  FL_GRID_RIGHT;

class FL_EXPORT Fl_Grid : public Fl_Group {
  friend class Fl_Grid_Type;

public:
  class Cell {
    friend class Fl_Grid;
  private:
    Cell *next_;                // next cell in the same row
    short row_;                 // row number
    short col_;                 // column number
    short rowspan_;             // row span (1 - n)
    short colspan_;             // column span (1 - n)
    Fl_Grid_Align align_;       // widget alignment in its cell
    Fl_Widget *widget_;         // assigned widget
    int w_;                     // minimal widget width
    int h_;                     // minimal widget height

  public:

    void Cell_() {              // common initialization
      next_ = NULL;
      row_ = 0;
      col_ = 0;
      rowspan_ = 1;
      colspan_ = 1;
      widget_ = NULL;
      w_ = 0;
      h_ = 0;
      align_ = 0;
    }

    Cell(int row, int col) {    // constructor
      Cell_();
      row_ = (short)row;
      col_ = (short)col;
    }

    Cell(Fl_Widget *w, int row, int col) {  // widget assignment
      Cell_();
      widget_ = w;
      row_ = (short)row;
      col_ = (short)col;
    }

    ~Cell() = default;

    Cell *next() const { return next_; }

    void next(Cell *c) {
      next_ = c;
```

```
    }

    Fl_Widget *widget() const { return widget_; }

    short row() const { return row_; }
    short col() const { return col_; }

    void rowspan(short v) { rowspan_ = v; }
    void colspan(short v) { colspan_ = v; }
    short rowspan() const { return rowspan_; }
    short colspan() const { return colspan_; }

    void align(Fl_Grid_Align align) { align_ = align; }
    Fl_Grid_Align align() const { return align_; }

    void minimum_size(int w, int h) { if (w>=0) w_ = w; if (h>=0) h_ = h; }
    void minimum_size(int *w, int *h) const { if (w) *w = w_; if (h) *h = h_; }
  }; // class Cell

private:
  class Row;
  class Col;
  short rows_;
  short cols_;

  short margin_left_;         // left margin
  short margin_top_;          // top margin
  short margin_right_;        // right margin
  short margin_bottom_;       // bottom margin
  short gap_row_;             // gap between rows
  short gap_col_;             // gap between columns
  Fl_Rect old_size;           // only for resize callback (TBD)
  Col  *Cols_;                // array of columns
  Row  *Rows_;                // array of rows
  bool need_layout_;          // true if layout needs to be calculated

protected:
  Fl_Color grid_color;        // color for drawing the grid lines (design helper)
  bool draw_grid_;            // draw the grid for testing / design

protected:
  void init();
  Cell *add_cell(int row, int col);
  void remove_cell(int row, int col);

public:
  Fl_Grid(int X, int Y, int W, int H, const char *L = 0);
  virtual ~Fl_Grid();

  // define and manage the layout and resizing

  virtual void layout(int rows, int cols, int margin = -1, int gap = -1);
  virtual void layout();
  virtual void clear_layout();
  virtual void resize(int X, int Y, int W, int H) override;

  short rows() const { return rows_; }
  short cols() const { return cols_; }

  void need_layout(int set) {
    if (set) {
      need_layout_ = true;
      redraw();
    }
    else {
      need_layout_ = false;
    }
  }

  bool need_layout() const {
    return need_layout_;
  }

protected:
  virtual void draw() override;
  void on_remove(int) override;
  virtual void draw_grid();           // draw grid lines for debugging

public:

  // get and set individual margins

  virtual void margin(int left, int top = -1, int right = -1, int bottom = -1);
  int margin(int *left, int *top, int *right, int *bottom) const;

  // get and set default row and column gaps for all rows and columns, respectively

```

```
  virtual void gap(int row_gap, int col_gap = -1);   // set default row and column gap(s)
  void gap(int *row_gap, int *col_gap) const;

  // find cells, get cell pointers

  Fl_Grid::Cell* cell(int row, int col) const;
  Fl_Grid::Cell* cell(Fl_Widget *widget) const;

  // assign a widget to a cell

  Fl_Grid::Cell* widget(Fl_Widget *wi, int row, int col, Fl_Grid_Align align = FL_GRID_FILL);
  Fl_Grid::Cell* widget(Fl_Widget *wi, int row, int col, int rowspan, int colspan, Fl_Grid_Align align
    = FL_GRID_FILL);

  // set minimal column and row sizes (widths and heights, respectively),
  // set row and column specific gaps and weights

  void col_width(int col, int value);
  void col_width(const int *value, size_t size);
  int col_width(int col) const;

  void col_weight(int col, int value);
  void col_weight(const int *value, size_t size);
  int col_weight(int col) const;

  void col_gap(int col, int value);
  void col_gap(const int *value, size_t size);
  int col_gap(int col) const;

  void row_height(int row, int value);
  void row_height(const int *value, size_t size);
  int row_height(int row) const;

  void row_weight(int row, int value);
  void row_weight(const int *value, size_t size);
  int row_weight(int row) const;

  void row_gap(int row, int value);
  void row_gap(const int *value, size_t size);
  int row_gap(int row) const;

  int computed_col_width(int col) const;
  int computed_row_height(int row) const;

  void show_grid(int set) {
    draw_grid_ = set ? true : false;
  }

  void show_grid(int set, Fl_Color col) {
    draw_grid_ = set ? true : false;
    grid_color = col;
  }

  void debug(int level = 127);

}; // class Fl_Grid

#endif // _FL_FL_GRID_H_
```

14.69 Fl_Group.H File Reference

Fl_Group and Fl_End classes.

```
#include "Fl_Widget.H"
#include <vector>
```

Classes

- class Fl_End

  This is a dummy class that allows you to end a Fl_Group in a constructor list of a class:

- class Fl_Group

  The Fl_Group class is the main FLTK container widget.

14.69.1 Detailed Description

Fl_Group and Fl_End classes.

14.70 Fl_Group.H

Go to the documentation of this file.

```
00001 //
00002 // Group header file for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 1998-2026 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file.  If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 //     https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 //     https://www.fltk.org/bugs.php
00015 //
00016
00021 #ifndef Fl_Group_H
00022 #define Fl_Group_H
00023
00024 #include "Fl_Widget.H"
00025
00026 #include <vector>
00027
00028 // Don't #include Fl_Rect.H because this would introduce lots
00029 // of unnecessary dependencies on Fl_Rect.H
00030 class Fl_Rect;
00031
00032
00059 class FL_EXPORT Fl_Group : public Fl_Widget {
00060
00061   std::vector<Fl_Widget *>child_; // vector of children
00062   Fl_Widget* savedfocus_;
00063   Fl_Widget* resizable_;
00064   Fl_Rect *bounds_; // remembered initial sizes of children
00065   int *sizes_; // remembered initial sizes of children (FLTK 1.3 compat.)
00066
00067   int navigation(int);
00068   static Fl_Group *current_;
00069
00070   // unimplemented copy ctor and assignment operator
00071   Fl_Group(const Fl_Group&);
00072   Fl_Group& operator=(const Fl_Group&);
00073
00074 protected:
00075   void draw() override;
00076   void draw_child(Fl_Widget& widget) const;
00077   void draw_children();
00078   void draw_outside_label(const Fl_Widget& widget) const ;
00079   void update_child(Fl_Widget& widget) const;
00080   Fl_Rect *bounds();
00081   int  *sizes(); // FLTK 1.3 compatibility
00082   virtual int on_insert(Fl_Widget*, int);
00083   virtual int on_move(int, int);
00084   virtual void on_remove(int);
00085
00086 public:
00087
00088   int handle(int) override;
00089   void begin();
00090   void end();
00091   static Fl_Group *current();
00092   static void current(Fl_Group *g);
00093
00097   int children() const { return (int)child_.size(); }
00098
00109   Fl_Widget *child(int n) const {
00110     if (n < 0 || n > children() - 1) return nullptr;
00111     return child_[n];
00112   }
00113
00114   int find(const Fl_Widget*) const;
00118   int find(const Fl_Widget& o) const { return find(&o); }
00119
00120   Fl_Widget* const* array() const;
00121
00122   void resize(int,int,int,int) override;
00127   Fl_Group(int,int,int,int, const char * = 0);
00128   virtual ~Fl_Group();
00129   void add(Fl_Widget&);
00133   void add(Fl_Widget* o) {add(*o);}
00134   void insert(Fl_Widget&, int i);
00139   void insert(Fl_Widget& o, Fl_Widget* before) {insert(o,find(before));}
00140   void remove(int index);
```

```
  void remove(Fl_Widget&);
  void remove(Fl_Widget* o) {remove(*o);}
  void clear();

  /* delete child n (by index) */
  virtual int delete_child(int n);

  void resizable(Fl_Widget& o) {resizable_ = &o;}
  void resizable(Fl_Widget* o) {resizable_ = o;}
  Fl_Widget* resizable() const {return resizable_;}
  void add_resizable(Fl_Widget& o) {resizable_ = &o; add(o);}
  void init_sizes();

  void clip_children(int c) { if (c) set_flag(CLIP_CHILDREN); else clear_flag(CLIP_CHILDREN); }
  unsigned int clip_children() const { return (flags() & CLIP_CHILDREN) != 0; }

  // Note: Doxygen docs in Fl_Widget.H to avoid redundancy.
  Fl_Group* as_group() override { return this; }
  Fl_Group const* as_group() const override { return this; }

  // back compatibility functions:

  Fl_Widget* & _ddfdesign_kludge() {return resizable_;}

  void forms_end();
};

// dummy class used to end child groups in constructors for complex
// subclasses of Fl_Group:
class FL_EXPORT Fl_End {
public:
  Fl_End() {Fl_Group::current()->end();}
};

#endif
```

14.71 Fl_Help_Dialog.H

```
//
// Fl_Help_Dialog dialog for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2021 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//
// ======================================================================
//  DO NOT EDIT FL/Fl_Help_Dialog.H and src/Fl_Help_Dialog.cxx !!!
// ======================================================================
//  Please use fluid to change src/Fl_Help_Dialog.fl interactively
//  and then use fluid to "write code" or edit and use fluid -c .
// ======================================================================
//

// generated by Fast Light User Interface Designer (fluid) version 1.0500

#ifndef Fl_Help_Dialog_H
#define Fl_Help_Dialog_H
#include <FL/Fl.H>
#include <FL/filename.H>
#include <FL/Fl_Double_Window.H>
#include <FL/Fl_Group.H>
#include <FL/Fl_Button.H>
#include <FL/Fl_Input.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Help_View.H>

class FL_EXPORT Fl_Help_Dialog {
  int index_;
  int max_;
  int line_[100]; // FIXME: we must remove those static numbers
  char file_[100][FL_PATH_MAX]; // FIXME: we must remove those static numbers
  int find_pos_;
public:
  Fl_Help_Dialog();
private:
  Fl_Double_Window *window_;
  Fl_Button *back_;
```

```
  inline void cb_back__i(Fl_Button*, void*);
  static void cb_back_(Fl_Button*, void*);
  Fl_Button *forward_;
  inline void cb_forward__i(Fl_Button*, void*);
  static void cb_forward_(Fl_Button*, void*);
  Fl_Button *smaller_;
  inline void cb_smaller__i(Fl_Button*, void*);
  static void cb_smaller_(Fl_Button*, void*);
  Fl_Button *larger_;
  inline void cb_larger__i(Fl_Button*, void*);
  static void cb_larger_(Fl_Button*, void*);
  Fl_Input *find_;
  inline void cb_find__i(Fl_Input*, void*);
  static void cb_find_(Fl_Input*, void*);
  Fl_Help_View *view_;
  inline void cb_view__i(Fl_Help_View*, void*);
  static void cb_view_(Fl_Help_View*, void*);
public:
  ~Fl_Help_Dialog();
  int h();
  void hide();
  int load(const char *f);
  void position(int xx, int yy);
  void resize(int xx, int yy, int ww, int hh);
  void show();
  void show(int argc, char **argv);
  void textsize(Fl_Fontsize s);
  Fl_Fontsize textsize();
  void topline(const char *n);
  void topline(int n);
  void value(const char *f);
  const char * value() const;
  int visible();
  int w();
  int x();
  int y();
};
#endif
```

14.72 Fl_Help_View.H

```
//
// Help Viewer widget definitions.
//
// Copyright 1997-2010 by Easy Software Products.
// Image support by Matthias Melcher, Copyright 2000-2009.
// Copyright 2011-2025 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef Fl_Help_View_H
#define Fl_Help_View_H

//
// FLTK header files
//

#include "Fl.H"
#include "Fl_Group.H"
#include "Fl_Scrollbar.H"

//
// System and C++ header files
//

#include <memory>   // std::unique_ptr<>

//
// Forward declarations and typedefs
//

class Fl_Shared_Image;
typedef const char *(Fl_Help_Func)(Fl_Widget *, const char *);

//
// Class declaration
```

```
//

class FL_EXPORT Fl_Help_View : public Fl_Group
{
private:
  class Impl;
  std::unique_ptr<Impl> impl_;

  Fl_Scrollbar  scrollbar_;
  Fl_Scrollbar  hscrollbar_;

protected:

  // Widget management

  void          draw() override;

public:

  static const char *copy_menu_text;

  // Widget management

  Fl_Help_View(int xx, int yy, int ww, int hh, const char *l = 0);
  ~Fl_Help_View() override;

  int           handle(int) override;
  void          resize(int,int,int,int) override;
  void          size(int W, int H) { Fl_Widget::size(W, H); }

  // HTML source and raw data

  void          value(const char *val);
  const char    *value() const;
  int           load(const char *f);
  int           find(const char *s, int p = 0);
  void          link(Fl_Help_Func *fn);

  const char    *filename() const;
  const char    *directory() const;
  const char    *title() const;

  // Rendering attributes

  int           size() const;
  void          textcolor(Fl_Color c);
  Fl_Color      textcolor() const;
  void          textfont(Fl_Font f);
  Fl_Font       textfont() const;
  void          textsize(Fl_Fontsize s);
  Fl_Fontsize   textsize() const;
  void          topline(const char *n);
  void          topline(int);
  int           topline() const;
  void          leftline(int);
  int           leftline() const;

  // Text selection

  void          clear_selection();
  void          select_all();
  int           text_selected() const;
  int           copy(int clipboard=1);

  // Scroll bars

  int scrollbar_size() const;
  void scrollbar_size(int newSize);
  Fl_Scrollbar  *scrollbar() { return &scrollbar_; }
  Fl_Scrollbar  *hscrollbar() { return &hscrollbar_; }

};

#endif // !Fl_Help_View_H
```

14.73 Fl_Hold_Browser.H

```
//
// Hold browser header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
```

```
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Hold_Browser widget . */

#ifndef Fl_Hold_Browser_H
#define Fl_Hold_Browser_H

#include "Fl_Browser.H"

class FL_EXPORT Fl_Hold_Browser : public Fl_Browser {
public:
  Fl_Hold_Browser(int X,int Y,int W,int H,const char *L=0);
};

#endif
```

14.74 Fl_Hor_Fill_Slider.H

```
//
// Horizontal fill slider header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Hor_Fill_Slider widget . */

#ifndef Fl_Hor_Fill_Slider_H
#define Fl_Hor_Fill_Slider_H

#include "Fl_Slider.H"

class FL_EXPORT Fl_Hor_Fill_Slider : public Fl_Slider {
public:
  Fl_Hor_Fill_Slider(int X,int Y,int W,int H,const char *L=0);
};

#endif
```

14.75 Fl_Hor_Nice_Slider.H

```
//
// Horizontal "nice" slider header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Hor_Nice_Slider widget . */

#ifndef Fl_Hor_Nice_Slider_H
#define Fl_Hor_Nice_Slider_H

#include "Fl_Slider.H"

```

```
class FL_EXPORT Fl_Hor_Nice_Slider : public Fl_Slider {
public:
  Fl_Hor_Nice_Slider(int X,int Y,int W,int H,const char *L=0);
};

#endif
```

14.76 Fl_Hor_Slider.H

```
//
// Horizontal slider header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2011 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Hor_Slider widget . */

#ifndef Fl_Hor_Slider_H
#define Fl_Hor_Slider_H

#include "Fl_Slider.H"

class FL_EXPORT Fl_Hor_Slider : public Fl_Slider {
public:

  Fl_Hor_Slider(int X,int Y,int W,int H,const char *l=0);
};

#endif
```

14.77 Fl_Hor_Value_Slider.H

```
//
// Horizontal value slider header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Hor_Value_Slider widget . */

#ifndef Fl_Hor_Value_Slider_H
#define Fl_Hor_Value_Slider_H

#include "Fl_Value_Slider.H"

class FL_EXPORT Fl_Hor_Value_Slider : public Fl_Value_Slider {
public:
  Fl_Hor_Value_Slider(int X,int Y,int W,int H,const char *l=0);
};

#endif
```

14.78 Fl_ICO_Image.H

```
//
// ICO image header file for the Fast Light Tool Kit (FLTK).
//
```

```
// Copyright 2022-2023 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     http://www.fltk.org/COPYING.php
//
// Please report all bugs and problems on the following page:
//
//     http://www.fltk.org/str.php
//

// https://en.wikipedia.org/wiki/ICO_(file_format)
// http://www.daubnet.com/en/file-format-ico

#ifndef Fl_ICO_Image_H
#  define Fl_ICO_Image_H
#  include "Fl_BMP_Image.H"

class FL_EXPORT Fl_ICO_Image : public Fl_BMP_Image {

public:
   struct IconDirEntry {
    int bWidth;
    int bHeight;
    int bColorCount;
    int bReserved;
    int wPlanes;
    int wBitCount;
    int dwBytesInRes;
    int dwImageOffset;
  };

  Fl_ICO_Image(const char *filename, int id = -1, const unsigned char *data = NULL, const size_t
datasize = 0);
  ~Fl_ICO_Image();

  int idcount() const { return idcount_; }

  const IconDirEntry * icondirentry() const { return icondirentry_; }

private:
  int idcount_;
  struct IconDirEntry *icondirentry_;
  void load_ico_(class Fl_Image_Reader &rdr, int id);
};

#endif // Fl_ICO_Image_H
```

14.79 Fl_Image.H File Reference

Fl_Image, Fl_RGB_Image classes.

```
#include "Enumerations.H"
#include "Fl_Widget.H"
```

Classes

- class Fl_Image

  Base class for image caching, scaling and drawing.
- class Fl_RGB_Image

  The Fl_RGB_Image class supports caching and drawing of full-color images with 1 to 4 channels of color information.

Enumerations

- enum Fl_RGB_Scaling { FL_RGB_SCALING_NEAREST = 0 , FL_RGB_SCALING_BILINEAR }

  The scaling algorithm to use for RGB images.

14.79.1 Detailed Description

Fl_Image, Fl_RGB_Image classes.

14.79.2 Enumeration Type Documentation

14.79.2.1 Fl_RGB_Scaling

enum Fl_RGB_Scaling

The scaling algorithm to use for RGB images.

Enumerator

FL_RGB_SCALING_NEAREST	default RGB image scaling algorithm
FL_RGB_SCALING_BILINEAR	more accurate, but slower RGB image scaling algorithm

14.80 Fl_Image.H

Go to the documentation of this file.

```
00001 //
00002 // Image header file for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 1998-2026 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file.  If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 //     https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 //     https://www.fltk.org/bugs.php
00015 //
00016
00020 #ifndef Fl_Image_H
00021 #define Fl_Image_H
00022
00023 #include "Enumerations.H"
00024 #include "Fl_Widget.H" // for fl_uintptr_t
00025
00026 class Fl_Widget;
00027 class Fl_Pixmap;
00028 struct Fl_Menu_Item;
00029 struct Fl_Label;
00030 class Fl_RGB_Image;
00031
00032
00036 enum Fl_RGB_Scaling {
00037   FL_RGB_SCALING_NEAREST = 0,
00038   FL_RGB_SCALING_BILINEAR
00039 };
00040
00041
00060 class FL_EXPORT Fl_Image {
00061   friend class Fl_Graphics_Driver;
00062 public:
00063   static const int ERR_NO_IMAGE       = -1;
00064   static const int ERR_FILE_ACCESS    = -2;
00065   static const int ERR_FORMAT         = -3;
00066   static const int ERR_MEMORY_ACCESS  = -4;
00067
00068 private:
00069   int w_, h_, d_, ld_, count_;
00070   int data_w_, data_h_;
00071   const char * const *data_;
00072   static Fl_RGB_Scaling RGB_scaling_; // method used when copying RGB images
00073   static Fl_RGB_Scaling scaling_algorithm_; // method used to rescale RGB source images before drawing
00074   // Forbid use of copy constructor and assign operator
00075   Fl_Image & operator=(const Fl_Image &);
00076   Fl_Image(const Fl_Image &);
00077   // Presently redefined in Fl_SVG_Image
00078   virtual void cache_size_(int &/*width*/, int &/*height*/) {}
00079
00080 protected:
00081
00087   void w(int W) {w_ = W; data_w_ = W;}
00093   void h(int H) {h_ = H; data_h_ = H;}
00097   void d(int D) {d_ = D;}
00109   void ld(int LD) {ld_ = LD;}
00117   void data(const char * const *p, int c) {data_ = p; count_ = c;}
00118   void draw_empty(int X, int Y);
```

```
  static void labeltype(const Fl_Label *lo, int lx, int ly, int lw, int lh, Fl_Align la);
  static void measure(const Fl_Label *lo, int &lw, int &lh);
  int draw_scaled(int X, int Y, int W, int H);

public:

  int w() const {return w_;}
  int h() const {return h_;}
  int data_w() const {return data_w_;}
  int data_h() const {return data_h_;}
  int d() const {return d_;}
  int ld() const {return ld_;}
  int count() const {return count_;}
  const char * const *data() const {return data_;}
  int fail() const;
  virtual void release() {
    delete this;
  }

  virtual class Fl_Shared_Image *as_shared_image() {
    return 0;
  }

  Fl_Image(int W, int H, int D);
  virtual ~Fl_Image();
  virtual Fl_Image *copy(int W, int H) const;
  Fl_Image *copy() const { return copy(w(), h()); }
  virtual void color_average(Fl_Color c, float i);
  void inactive() { color_average(FL_GRAY, .33f); }
  virtual void desaturate();
  virtual void label(Fl_Widget*w);
  virtual void label(Fl_Menu_Item*m);
  virtual void draw(int X, int Y, int W, int H, int cx=0, int cy=0); // platform dependent
  void draw(int X, int Y) {draw(X, Y, w(), h(), 0, 0);} // platform dependent
  virtual void uncache();

  // used by fl_define_FL_IMAGE_LABEL() to avoid 'friend' declaration
  static Fl_Labeltype define_FL_IMAGE_LABEL();

  // set RGB image scaling method
  static void RGB_scaling(Fl_RGB_Scaling);
  // get RGB image scaling method
  static Fl_RGB_Scaling RGB_scaling();

  // set the image drawing size
  virtual void scale(int width, int height, int proportional = 1, int can_expand = 0);
  static void scaling_algorithm(Fl_RGB_Scaling algorithm) {scaling_algorithm_ = algorithm; }
  static Fl_RGB_Scaling scaling_algorithm() {return scaling_algorithm_;}
  static bool register_images_done;
};

class Fl_SVG_Image;

class FL_EXPORT Fl_RGB_Image : public Fl_Image {
  friend class Fl_Graphics_Driver;
  static size_t max_size_;
public:

  const uchar *array;
  int alloc_array;

private:
  // These two variables are used to cache the image and mask for the main display graphics driver
  fl_uintptr_t id_;
  fl_uintptr_t mask_;
  int cache_w_, cache_h_; // size of image when cached

  Fl_RGB_Image *copy_scale_down_2h_() const;
  Fl_RGB_Image *copy_scale_down_2v_() const;
  Fl_RGB_Image *copy_bilinear_(uint32_t W, uint32_t H) const;
  Fl_RGB_Image *copy_nearest_neighbor_(int W, int H) const;
  Fl_RGB_Image *copy_optimize_(int W, int H) const;
public:

  Fl_RGB_Image(const uchar *bits, int W, int H, int D=3, int LD=0);
  Fl_RGB_Image(const uchar *bits, int bits_length, int W, int H, int D, int LD);
  Fl_RGB_Image(const Fl_Pixmap *pxm, Fl_Color bg=FL_GRAY);
  ~Fl_RGB_Image() override;
  Fl_Image *copy(int W, int H) const override;
  Fl_Image *copy() const { return Fl_Image::copy(); }
  void color_average(Fl_Color c, float i) override;
  void desaturate() override;
  void draw(int X, int Y, int W, int H, int cx=0, int cy=0) override;
  void draw(int X, int Y) {draw(X, Y, w(), h(), 0, 0);}
  void label(Fl_Widget*w) override;
  void label(Fl_Menu_Item*m) override;
```

```
  void uncache() override;
  int cache_w() const { return cache_w_; }
  int cache_h() const { return cache_h_; }
  static void max_size(size_t size) { max_size_ = size;}
  static size_t max_size() {return max_size_;}
  virtual Fl_SVG_Image *as_svg_image() { return NULL; }
  virtual const Fl_SVG_Image *as_svg_image() const { return NULL; }
  virtual void normalize() {}
};

#endif // !Fl_Image_H
```

14.81 Fl_Image_Surface.H

```
//
// Draw-to-image code for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2026 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef Fl_Image_Surface_H
#define Fl_Image_Surface_H

#include <FL/Fl_Widget_Surface.H>
#include <FL/Fl_Image.H>
#include <FL/Fl_Shared_Image.H>
#include <FL/platform_types.h> // for Fl_Offscreen


class FL_EXPORT Fl_Image_Surface : public Fl_Widget_Surface {
  friend class Fl_Graphics_Driver;
private:
  class Fl_Image_Surface_Driver *platform_surface;
  Fl_Offscreen get_offscreen_before_delete_();
protected:
  void translate(int x, int y) override;
  void untranslate() override;
public:
  Fl_Image_Surface(int w, int h, int high_res = 0, Fl_Offscreen off = 0);
  ~Fl_Image_Surface();
  void set_current() override;
  bool is_current() override;
  Fl_RGB_Image *image();
  Fl_Shared_Image *highres_image();
  void origin(int *x, int *y) override;
  void origin(int x, int y) override;
  int printable_rect(int *w, int *h) override;
  Fl_Offscreen offscreen();
  void rescale();
  void mask(const Fl_RGB_Image *);
};


class Fl_Image_Surface_Driver : public Fl_Widget_Surface {
  friend class Fl_Image_Surface;
private:
  Fl_Image_Surface *image_surface_;
protected:
  int width;
  int height;
  Fl_Offscreen offscreen;
  int external_offscreen;
  Fl_Image_Surface_Driver(int w, int h, int /*high_res*/, Fl_Offscreen off)
  : Fl_Widget_Surface(NULL), image_surface_(nullptr), width(w), height(h), offscreen(off)
  {
    external_offscreen = (off != 0);
  }
  virtual ~Fl_Image_Surface_Driver() {}
  static void copy_with_mask(Fl_RGB_Image* mask, uchar *dib_dst, uchar *dib_src,
                             int line_size, bool bottom_to_top);
  static Fl_RGB_Image *RGB3_to_RGB1(const Fl_RGB_Image *rgb3, int W, int H);
  void set_current() override = 0;
  void translate(int x, int y) override = 0;
  void untranslate() override = 0;
  int printable_rect(int *w, int *h) override;
```

```
  virtual Fl_RGB_Image *image() = 0;
  virtual void mask(const Fl_RGB_Image *) {}
  static Fl_Image_Surface_Driver *newImageSurfaceDriver(int w, int h, int high_res, Fl_Offscreen off);
public:
  Fl_Image_Surface *image_surface() const { return image_surface_; }
};

#endif // Fl_Image_Surface_H
```

14.82 Fl_Input.H

```
//
// Input header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2023 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Input widget . */

#ifndef Fl_Input_H
#define Fl_Input_H

#include "Fl_Input_.H"

class FL_EXPORT Fl_Input : public Fl_Input_ {
  friend class Fl_Screen_Driver;
  friend class Fl_Cocoa_Screen_Driver; // Not ideal, but probably no other platform will use it
  int shift_position(int p);
  int shift_up_down_position(int p);
  void handle_mouse(int keepmark=0);

  // Private keyboard functions
  int kf_lines_up(int repeat_num);
  int kf_lines_down(int repeat_num);
  int kf_page_up();
  int kf_page_down();
  int kf_insert_toggle();
  int kf_delete_word_right();
  int kf_delete_word_left();
  int kf_delete_sol();
  int kf_delete_eol();
  int kf_delete_char_right();
  int kf_delete_char_left();
  int kf_move_sol();
  int kf_move_eol();
  int kf_clear_eol();
  int kf_move_char_left();
  int kf_move_char_right();
  int kf_move_word_left();
  int kf_move_word_right();
  int kf_move_up_and_sol();
  int kf_move_down_and_eol();
  int kf_top();
  int kf_bottom();
  int kf_select_all();
  int kf_undo();
  int kf_redo();
  int kf_copy();
  int kf_paste();
  int kf_copy_cut();

protected:
  void draw() override;
  int handle_key();
  int handle_rmb();

public:
  int handle(int) override;
  Fl_Input(int,int,int,int,const char * = 0);
  static const char *cut_menu_text;
  static const char *copy_menu_text;
  static const char *paste_menu_text;
};

```

```
#endif
```

14.83 Fl_Input_.H

```
//
// Input base class header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2015 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Input_ widget . */

#ifndef Fl_Input__H
#define Fl_Input__H

#ifndef Fl_Widget_H
#include "Fl_Widget.H"
#endif

#define FL_NORMAL_INPUT         0
#define FL_FLOAT_INPUT          1
#define FL_INT_INPUT            2
#define FL_HIDDEN_INPUT         3
#define FL_MULTILINE_INPUT      4
#define FL_SECRET_INPUT         5
#define FL_INPUT_TYPE           7
#define FL_INPUT_READONLY       8
#define FL_NORMAL_OUTPUT        (FL_NORMAL_INPUT | FL_INPUT_READONLY)
#define FL_MULTILINE_OUTPUT     (FL_MULTILINE_INPUT | FL_INPUT_READONLY)
#define FL_INPUT_WRAP           16
#define FL_MULTILINE_INPUT_WRAP (FL_MULTILINE_INPUT | FL_INPUT_WRAP)
#define FL_MULTILINE_OUTPUT_WRAP (FL_MULTILINE_INPUT | FL_INPUT_READONLY | FL_INPUT_WRAP)

class Fl_Input_Undo_Action;
class Fl_Input_Undo_Action_List;

class FL_EXPORT Fl_Input_ : public Fl_Widget {

  const char* value_;

  char* buffer;

  char* placeholder_ { nullptr };

  int size_;

  int bufsize;

  int position_;

  int mark_;

  int tab_nav_;

  int xscroll_, yscroll_;

  int mu_p;

  int maximum_size_;

  int shortcut_;

  uchar erase_cursor_only;

  Fl_Font textfont_;

  Fl_Fontsize textsize_;

  Fl_Color textcolor_;

  Fl_Color cursor_color_;

  Fl_Input_Undo_Action* undo_;
  Fl_Input_Undo_Action_List* undo_list_;
```

```
  Fl_Input_Undo_Action_List* redo_list_;

  static double up_down_pos;

  static int was_up_down;

  /* Convert a given text segment into the text that will be rendered on screen. */
  const char* expand(const char*, char*) const;

  /* Calculates the width in pixels of part of a text buffer. */
  double expandpos(const char*, const char*, const char*, int*) const;

  /* Mark a range of characters for update. */
  void minimal_update(int, int);

  /* Mark a range of characters for update. */
  void minimal_update(int p);

  /* Copy the value from a possibly static entry into the internal buffer. */
  void put_in_buffer(int newsize);

  /* Set the current font and font size. */
  void setfont() const;

protected:

  /* Find the start of a word. */
  int word_start(int i) const;

  /* Find the end of a word. */
  int word_end(int i) const;

  /* Find the start of a line. */
  int line_start(int i) const;

  /* Find the end of a line. */
  int line_end(int i) const;

  /* Draw the text in the passed bounding box. */
  void drawtext(int, int, int, int);

  /* Draw the text in the passed bounding box. */
  void drawtext(int, int, int, int, bool draw_active);

  /* Move the cursor to the column given by up_down_pos. */
  int up_down_position(int, int keepmark=0);

  /* Handle mouse clicks and mouse moves. */
  void handle_mouse(int, int, int, int, int keepmark=0);

  /* Handle all kinds of text field related events. */
  int handletext(int e, int, int, int, int);

  /* Check the when() field and do a callback if indicated. */
  void maybe_do_callback(Fl_Callback_Reason reason = FL_REASON_UNKNOWN);

  int xscroll() const {return xscroll_;}

  int yscroll() const {return yscroll_;}
  void yscroll(int yOffset) { yscroll_ = yOffset; damage(FL_DAMAGE_EXPOSE);}

  /* Return the number of lines displayed on a single page.  */
  int linesPerPage();

  /* Apply the current undo/redo operation, called from undo() or redo() */
  int apply_undo();

public:

  /* Change the size of the widget. */
  void resize(int, int, int, int) override;

  /* Constructor */
  Fl_Input_(int, int, int, int, const char* = 0);

  /* Destructor */
  ~Fl_Input_();

  /* Changes the widget text. */
  int value(const char*);

  /* Changes the widget text. */
  int value(const char*, int);

  /* Changes the widget text. */
  int value(int value);

```

```
/* Changes the widget text. */
int value(double value);

/* Changes the widget text. */
int static_value(const char*);

/* Changes the widget text. */
int static_value(const char*, int);

const char* value() const {return value_;}

int ivalue() const;

double dvalue() const;

/* Set a placeholder text that is displayed when the widget is empty */
void placeholder(const char* text);

/* Get the placeholder text */
const char* placeholder() const;

/* Returns the Unicode character at index \p i. */
unsigned int index(int i) const;

int size() const {return size_;}

void size(int W, int H) { Fl_Widget::size(W, H); }

int maximum_size() const {return maximum_size_;}

void maximum_size(int m) {maximum_size_ = m;}

int insert_position() const { return position_; }
FL_DEPRECATED("since 1.4.0 - use insert_position() instead",
int position() const ) { return insert_position(); }

int mark() const {return mark_;}

/* Sets the index for the cursor and mark. */
int insert_position(int p, int m);
FL_DEPRECATED("since 1.4.0 - use insert_position(p, m) or Fl_Widget::position(x, y) instead",
int position(int p, int m)) { return insert_position(p, m); }

int insert_position(int p) { return insert_position(p, p); }
FL_DEPRECATED("since 1.4.0 - use insert_position(p) instead",
int position(int p)) { return insert_position(p); }

int mark(int m) {return insert_position(insert_position(), m);}

/* Deletes text from \p b to \p e and inserts the new string \p text. */
int replace(int b, int e, const char *text, int ilen=0);

int cut() {return replace(insert_position(), mark(), 0);}

int cut(int n) {return replace(insert_position(), insert_position()+n, 0);}

int cut(int a, int b) {return replace(a, b, 0);}

int insert(const char* t, int l=0){return replace(position_, mark_, t, l);}

/* Append text at the end. */
int append(const char* t, int l=0, char keep_selection=0);

/* Put the current selection into the clipboard. */
int copy(int clipboard);

/* Undo previous changes to the text buffer. */
int undo();

/* Return true if the last operation can be undone. */
bool can_undo() const;

/* Redo previous undo operations. */
int redo();

/* Return true if there is a redo action in the list. */
bool can_redo() const;

/* Copy the yank buffer to the clipboard. */
int copy_cuts();

int shortcut() const {return shortcut_;}

void shortcut(int s) {shortcut_ = s;}

Fl_Font textfont() const {return textfont_;}
```

```
  void textfont(Fl_Font s) {textfont_ = s;}

  Fl_Fontsize textsize() const {return textsize_;}

  void textsize(Fl_Fontsize s) {textsize_ = s;}

  Fl_Color textcolor() const {return textcolor_;}

  void textcolor(Fl_Color n) {textcolor_ = n;}

  Fl_Color cursor_color() const {return cursor_color_;}

  void cursor_color(Fl_Color n) {cursor_color_ = n;}

  int input_type() const {return type() & FL_INPUT_TYPE; }

  void input_type(int t) { type((uchar)(t | readonly())); }

  int readonly() const { return type() & FL_INPUT_READONLY; }

  void readonly(int b) { if (b) type((uchar)(type() | FL_INPUT_READONLY));
                         else type((uchar)(type() & ~FL_INPUT_READONLY)); }

  int wrap() const { return type() & FL_INPUT_WRAP; }

  void wrap(int b) { if (b) type((uchar)(type() | FL_INPUT_WRAP));
                         else type((uchar)(type() & ~FL_INPUT_WRAP)); }

  void tab_nav(int val) {
    tab_nav_ = val;
  }

  int tab_nav() const {
    return tab_nav_;
  }
};

#endif
```

14.84 Fl_Input_Choice.H

```
//
// An input/chooser widget.
//            _______________  ____
//           |               || __ |
//           | input area    || \/ |
//           |_______________||____|
//
// Copyright 2004 by Greg Ercolano.
// Copyright 1998-2026 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Input_Choice widget . */

#ifndef Fl_Input_Choice_H
#define Fl_Input_Choice_H

#include <FL/Fl.H>
#include <FL/Fl_Group.H>
#include <FL/Fl_Input.H>
#include <FL/Fl_Menu_Button.H>

/*
  A combination of the input widget and a menu button.

  The user can either type into the input area, or use the
  menu button chooser on the right to choose an item which loads
  the input area with the selected text.

  Note: doxygen docs in src/Fl_Input_Choice.cxx
*/

class FL_EXPORT Fl_Input_Choice : public Fl_Group {

```

```
  // Private class to handle slightly 'special' behavior of menu button
  class InputMenuButton : public Fl_Menu_Button {
    void draw() override;
    const Fl_Menu_Item* popup();
  public:
    InputMenuButton(int X, int Y, int W, int H, const char *L=0);
    int handle(int e) override;
  };

  Fl_Input *inp_;
  InputMenuButton *menu_;

  // note: this is used by the Fl_Input_Choice ctor.
  static void menu_cb(Fl_Widget*, void *data);

  // note: this is used by the Fl_Input_Choice ctor.
  static void inp_cb(Fl_Widget*, void *data);

protected:
  // Custom resize behavior -- input stretches, menu button doesn't

  virtual int inp_x() const { return(x() + Fl::box_dx(box())); }
  virtual int inp_y() const { return(y() + Fl::box_dy(box())); }
  virtual int inp_w() const { return(w() - Fl::box_dw(box()) - menu_w()); }
  virtual int inp_h() const { return(h() - Fl::box_dh(box())); }

  virtual int menu_x() const { return x() + w() - menu_w() - Fl::box_dx(box()); }
  virtual int menu_y() const { return y() + Fl::box_dy(box()); }
  virtual int menu_w() const { return 20; }
  virtual int menu_h() const { return h() - Fl::box_dh(box()); }

  void draw() override;

public:

  Fl_Input_Choice(int X, int Y, int W, int H, const char *L=0);

  void resize(int X, int Y, int W, int H) override;

  void add(const char *s) { menu_->add(s); }

  int changed() const { return inp_->changed() | Fl_Widget::changed(); }

  // Clears the changed() state of both input and menu button widgets.
  void clear_changed();

  // Sets the changed() state of both input and menu button widgets.
  void set_changed();

  void clear() { menu_->clear(); }

  Fl_Boxtype down_box() const { return (menu_->down_box()); }

  void down_box(Fl_Boxtype b) { menu_->down_box(b); }

  const Fl_Menu_Item *menu() { return (menu_->menu()); }

  void menu(const Fl_Menu_Item *m) { menu_->menu(m); }

  Fl_Color textcolor() const { return (inp_->textcolor());}

  void textcolor(Fl_Color c) { inp_->textcolor(c);}

  Fl_Font textfont() const { return (inp_->textfont());}

  void textfont(Fl_Font f) { inp_->textfont(f);}

  Fl_Fontsize textsize() const { return (inp_->textsize()); }

  void textsize(Fl_Fontsize s) { inp_->textsize(s); }

  const char* value() const { return (inp_->value()); }

  void value(const char *val) { inp_->value(val); }

  /* Chooses item# \p val in the menu, and sets the Fl_Input text field
    to that value. Any previous text is cleared. */
  void value(int val);

  int update_menubutton();

  Fl_Menu_Button* menubutton() const { return menu_; }

  Fl_Input* input() const { return inp_; }
};

#endif // !Fl_Input_Choice_H
```

14.85 Fl_Int_Input.H

```
//
// Integer input header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Int_Input widget . */

#ifndef Fl_Int_Input_H
#define Fl_Int_Input_H

#include "Fl_Input.H"

class FL_EXPORT Fl_Int_Input : public Fl_Input {
public:
  Fl_Int_Input(int X,int Y,int W,int H,const char *l = 0);
};

#endif
```

14.86 Fl_JPEG_Image.H

```
//
// JPEG image header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_JPEG_Image class . */

#ifndef Fl_JPEG_Image_H
#define Fl_JPEG_Image_H
#  include "Fl_Image.H"

class FL_EXPORT Fl_JPEG_Image : public Fl_RGB_Image {

public:

  Fl_JPEG_Image(const char *filename);
  Fl_JPEG_Image(const char *name, const unsigned char *data, int data_length=-1);

protected:

  void load_jpg_(const char *filename, const char *sharename, const unsigned char *data, int
data_length=-1);

};

// Support functions to write JPEG image files (since 1.4.0)

FL_EXPORT int fl_write_jpeg(const char *filename, Fl_RGB_Image *img);
FL_EXPORT int fl_write_jpeg(const char *filename, const char *pixels, int w, int h, int d=3, int
ld=0);
FL_EXPORT int fl_write_jpeg(const char *filename, const unsigned char *pixels, int w, int h, int d=3,
int ld=0);

#endif
```

14.87 Fl_Light_Button.H

```
//
// Lighted button header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2022 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Light_Button widget . */

#ifndef Fl_Light_Button_H
#define Fl_Light_Button_H

#include "Fl_Button.H"

class FL_EXPORT Fl_Light_Button : public Fl_Button {
protected:
    void draw() override;
public:
    int handle(int) override;
    Fl_Light_Button(int x,int y,int w,int h,const char *l = 0);
};

#endif
```

14.88 Fl_Line_Dial.H

```
//
// Line dial header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Line_Dial widget . */

#ifndef Fl_Line_Dial_H
#define Fl_Line_Dial_H

#include "Fl_Dial.H"

class FL_EXPORT Fl_Line_Dial : public Fl_Dial {
public:
  Fl_Line_Dial(int X,int Y,int W,int H, const char *L = 0);
};

#endif
```

14.89 Fl_Menu.H

```
//
// Old menu header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
```

```
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

// this include file is for back compatibility only
#include "Fl_Menu_Item.H"
```

14.90 Fl_Menu_.H

```
//
// Menu base class header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2024 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Menu_ widget . */

#ifndef Fl_Menu__H
#define Fl_Menu__H

#ifndef Fl_Widget_H
#include "Fl_Widget.H"
#endif
#include "Fl_Menu_Item.H"

class FL_EXPORT Fl_Menu_ : public Fl_Widget {

  Fl_Menu_Item *menu_;
  const Fl_Menu_Item *value_;
  const Fl_Menu_Item *prev_value_;

protected:

  uchar alloc;                  // flag indicates if menu_ is a dynamic copy (=1) or not (=0)
  uchar down_box_;
  Fl_Boxtype menu_box_;
  Fl_Font textfont_;
  Fl_Fontsize textsize_;
  Fl_Color textcolor_;

  int item_pathname_(char *name, int namelen, const Fl_Menu_Item *finditem,
                     const Fl_Menu_Item *menu=0) const;
public:
  Fl_Menu_(int,int,int,int,const char * =0);
  ~Fl_Menu_();

  int item_pathname(char *name, int namelen, const Fl_Menu_Item *finditem=0) const;
  const Fl_Menu_Item* picked(const Fl_Menu_Item*);
  const Fl_Menu_Item* find_item(const char *name);
  const Fl_Menu_Item* find_item(Fl_Callback*);
  const Fl_Menu_Item* find_item_with_user_data(void*);
  const Fl_Menu_Item* find_item_with_argument(long);
  int find_index(const char *name) const;
  int find_index(const Fl_Menu_Item *item) const;
  int find_index(Fl_Callback *cb) const;

  const Fl_Menu_Item* test_shortcut() {return picked(menu()->test_shortcut());}
  void global();

  const Fl_Menu_Item *menu() const {return menu_;}
  const Fl_Menu_Item *menu_end(); // in src/Fl_Menu_add.cxx
  void menu(const Fl_Menu_Item *m);
  void copy(const Fl_Menu_Item *m, void* user_data = 0);
  int insert(int index, const char*, int shortcut, Fl_Callback*, void* = 0, int = 0);
  int  add(const char*, int shortcut, Fl_Callback*, void* = 0, int = 0); // see src/Fl_Menu_add.cxx
  int  add(const char* a, const char* b, Fl_Callback* c, void* d = 0, int e = 0) {
      return add(a,fl_old_shortcut(b),c,d,e);
  }
  int insert(int index, const char* a, const char* b, Fl_Callback* c, void* d = 0, int e = 0) {
      return insert(index,a,fl_old_shortcut(b),c,d,e);
  }
  int  add(const char *);
  int  size() const ;
```

```
  void size(int W, int H) { Fl_Widget::size(W, H); }
  void clear();
  int clear_submenu(int index);
  void replace(int,const char *);
  void remove(int);
  void shortcut(int i, int s) {menu_[i].shortcut(s);}
  void mode(int i,int fl) {menu_[i].flags = fl;}
  int  mode(int i) const {return menu_[i].flags;}

  const Fl_Menu_Item *mvalue() const {return value_;}

  const Fl_Menu_Item *prev_mvalue() const {return prev_value_;}
  // Return the index into the menu() of the last item chosen by the user or -1.
  int value() const;
  // Set the internal value_ of the menu to the given Fl_Menu_Item.
  int value(const Fl_Menu_Item*);
  int value(int i) {
    if (!menu_ || i < 0 || i >= size())
      return 0;
    return value(menu_ + i);
  }

  const char *text() const {return value_ ? value_->text : 0;}
  const char *text(int i) const {return menu_[i].text;}

  Fl_Font textfont() const {return textfont_;}
  void textfont(Fl_Font c) {textfont_=c;}
  Fl_Fontsize textsize() const {return textsize_;}
  void textsize(Fl_Fontsize c) {textsize_=c;}
  Fl_Color textcolor() const {return textcolor_;}
  void textcolor(Fl_Color c) {textcolor_=c;}

  Fl_Boxtype down_box() const {return (Fl_Boxtype)down_box_;}
  void down_box(Fl_Boxtype b) {down_box_ = (uchar)b;}

  Fl_Boxtype menu_box() const { return menu_box_; }
  void menu_box(Fl_Boxtype b) { menu_box_ = b; }

  Fl_Color down_color() const {return selection_color();}
  void down_color(unsigned c) {selection_color(c);}
  void setonly(Fl_Menu_Item* item);
};

#endif
```

14.91 Fl_Menu_Bar.H

```
//
// Menu bar header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2017 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Menu_Bar widget . */

#ifndef Fl_Menu_Bar_H
#define Fl_Menu_Bar_H

#include "Fl_Menu_.H"

class FL_EXPORT Fl_Menu_Bar : public Fl_Menu_ {
  friend class Fl_Sys_Menu_Bar_Driver;
protected:
    void draw() override;
public:
    int handle(int) override;
  Fl_Menu_Bar(int X, int Y, int W, int H, const char *l=0);
  virtual void update() {}
  virtual void play_menu(const Fl_Menu_Item *item);
};

#endif
```

14.92 Fl_Menu_Button.H

```
//
// Menu button header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2022 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Menu_Button widget . */

#ifndef Fl_Menu_Button_H
#define Fl_Menu_Button_H

#include "Fl_Menu_.H"

class FL_EXPORT Fl_Menu_Button : public Fl_Menu_ {
protected:
  void draw() override;
  static Fl_Menu_Button* pressed_menu_button_;
public:
  enum popup_buttons {POPUP1 = 1,
    POPUP2,
    POPUP12,
    POPUP3,
    POPUP13,
    POPUP23,
    POPUP123
  };
  int handle(int) override;
  const Fl_Menu_Item* popup();
  Fl_Menu_Button(int,int,int,int,const char * =0);
};

#endif
```

14.93 Fl_Menu_Item.H File Reference

```
#include <FL/Fl_Widget.H>
#include <FL/Fl_Image.H>
#include <FL/Fl_Multi_Label.H>
#include <FL/platform_types.h>
#include <FL/fl_draw.H>
```

Classes

- struct Fl_Menu_Item

  The Fl_Menu_Item structure defines a single menu item that is used by the Fl_Menu_ class.

Typedefs

- typedef Fl_Menu_Item **Fl_Menu**

Enumerations

- enum {
  FL_MENU_INACTIVE = 1 , FL_MENU_TOGGLE = 2 , FL_MENU_VALUE = 4 , FL_MENU_RADIO = 8 ,
  FL_MENU_INVISIBLE = 0x10 , FL_SUBMENU_POINTER = 0x20 , FL_SUBMENU = 0x40 , FL_MENU_DIVIDER = 0x80 ,
  FL_MENU_HORIZONTAL = 0x100 , FL_MENU_CHATTY = 0x200 , FL_MENU_HEADLINE = 0x400 }

- enum {
  FL_PUP_NONE = 0 , **FL_PUP_GREY** = FL_MENU_INACTIVE , **FL_PUP_GRAY** = FL_MENU_INACTIVE ,
  FL_MENU_BOX = FL_MENU_TOGGLE ,
  FL_PUP_BOX = FL_MENU_TOGGLE , **FL_MENU_CHECK** = FL_MENU_VALUE , **FL_PUP_CHECK** = FL↩
  _MENU_VALUE , **FL_PUP_RADIO** = FL_MENU_RADIO ,
  FL_PUP_INVISIBLE = FL_MENU_INVISIBLE , **FL_PUP_SUBMENU** = FL_SUBMENU_POINTER }

14.93.1 Enumeration Type Documentation

14.93.1.1 anonymous enum

`anonymous enum`

Enumerator

FL_MENU_INACTIVE	Deactivate menu item (gray out)
FL_MENU_TOGGLE	Item is a checkbox toggle (shows checkbox for on/off state)
FL_MENU_VALUE	The on/off state for checkbox/radio buttons (if set, state is 'on')
FL_MENU_RADIO	Item is a radio button (one checkbox of many can be on)
FL_MENU_INVISIBLE	Item will not show up (shortcut will work)
FL_SUBMENU_POINTER	Indicates user_data() is a pointer to another menu array.
FL_SUBMENU	Item is a submenu to other items.
FL_MENU_DIVIDER	Creates divider line below this item. Also ends a group of radio buttons.
FL_MENU_HORIZONTAL	reserved, do not use
FL_MENU_CHATTY	Menu Item receives additional callbacks.
FL_MENU_HEADLINE	Mark menu item as a headline. Note: All other bits in `flags` are reserved: do not use them for your own purposes!

14.94 Fl_Menu_Item.H

Go to the documentation of this file.

```
00001 //
00002 // Menu item header file for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 1998-2024 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file.  If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 //     https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 //     https://www.fltk.org/bugs.php
00015 //
00016
00017 #ifndef Fl_Menu_Item_H
00018 #define Fl_Menu_Item_H
00019
00020 #include <FL/Fl_Widget.H>
00021 #include <FL/Fl_Image.H>
00022 #include <FL/Fl_Multi_Label.H>
00023 #include <FL/platform_types.h> // for FL_COMMAND and FL_CONTROL
00024 #include <FL/fl_draw.H>
00025
00026 // doxygen needs the following line to enable e.g. ::FL_MENU_TOGGLE to link to the enums
00028
00029 enum { // values for flags:
00030   FL_MENU_INACTIVE   = 1,
00031   FL_MENU_TOGGLE     = 2,
00032   FL_MENU_VALUE      = 4,
00033   FL_MENU_RADIO      = 8,
00034   FL_MENU_INVISIBLE  = 0x10,
00035   FL_SUBMENU_POINTER = 0x20,
00036   FL_SUBMENU         = 0x40,
00037   FL_MENU_DIVIDER    = 0x80,
```

```
  FL_MENU_HORIZONTAL = 0x100,
  FL_MENU_CHATTY     = 0x200,
  FL_MENU_HEADLINE   = 0x400,
};

class Fl_Menu_;

struct FL_EXPORT Fl_Menu_Item {
  const char *text;
  int shortcut_;
  Fl_Callback *callback_;
  void *user_data_;
  int flags;
  uchar labeltype_;
  Fl_Font labelfont_;
  Fl_Fontsize labelsize_;
  Fl_Color labelcolor_;

  // advance N items, skipping submenus:
  const Fl_Menu_Item *next(int=1) const;

  Fl_Menu_Item *next(int i=1) {
    return (Fl_Menu_Item*)(((const Fl_Menu_Item*)this)->next(i));}

  const Fl_Menu_Item *first() const { return next(0); }

  Fl_Menu_Item *first() { return next(0); }

  // methods on menu items:
  const char* label() const { return text; }

  void label(const char* a) { text = a; }

  void label(Fl_Labeltype a, const char* b) {
    labeltype_ = (uchar)a;
    text = b;
  }

  void multi_label(const Fl_Multi_Label *ml) {
    label(FL_MULTI_LABEL, (const char *)ml);
  }

  void image_label(const Fl_Image *image) {
    label(FL_IMAGE_LABEL, (const char *)image);
  }

  Fl_Labeltype labeltype() const {return (Fl_Labeltype)labeltype_;}

  void labeltype(Fl_Labeltype a) {labeltype_ = (uchar)a;}

  Fl_Color labelcolor() const {return labelcolor_;}

  void labelcolor(Fl_Color a) {labelcolor_ = a;}
  Fl_Font labelfont() const {return labelfont_;}

  void labelfont(Fl_Font a) {labelfont_ = a;}

  Fl_Fontsize labelsize() const {return labelsize_;}

  void labelsize(Fl_Fontsize a) {labelsize_ = a;}

  Fl_Callback_p callback() const {return callback_;}

  void callback(Fl_Callback* c, void* p) {callback_=c; user_data_=p;}

  void callback(Fl_Callback* c) {callback_=c;}

  void callback(Fl_Callback0 *c) {
    callback_ = (Fl_Callback *)(void *)c;
  }

  void callback(Fl_Callback1 *c, long p = 0) {
    callback_ = (Fl_Callback *)(void *)c;
    user_data_ = (void *)(fl_intptr_t)p;
  }

  void* user_data() const {return user_data_;}
  void user_data(void* v) {user_data_ = v;}
  long argument() const {return (long)(fl_intptr_t)user_data_;}
  void argument(long v) {user_data_ = (void*)(fl_intptr_t)v;}

  int shortcut() const {return shortcut_;}

  void shortcut(int s) {shortcut_ = s;}
  int submenu() const {return flags&(FL_SUBMENU|FL_SUBMENU_POINTER);}
  int checkbox() const {return flags&FL_MENU_TOGGLE;}
  int radio() const {return flags&FL_MENU_RADIO;}
```

```
  int value() const {return (flags & FL_MENU_VALUE) ? 1 : 0;}

  void value(int v) { v ? set() : clear(); }

  void set() {flags |= FL_MENU_VALUE;}

  void clear() {flags &= ~FL_MENU_VALUE;}

  FL_DEPRECATED("since 1.3 - use setonly(Fl_Menu_Item const* first) instead",
  void setonly());
  void setonly(Fl_Menu_Item const* first);

  int visible() const {return !(flags&FL_MENU_INVISIBLE);}

  void show() {flags &= ~FL_MENU_INVISIBLE;}

  void hide() {flags |= FL_MENU_INVISIBLE;}

  int active() const {return !(flags&FL_MENU_INACTIVE);}

  void activate() {flags &= ~FL_MENU_INACTIVE;}
  void deactivate() {flags |= FL_MENU_INACTIVE;}

  int activevisible() const {return !(flags & (FL_MENU_INACTIVE|FL_MENU_INVISIBLE));}

  int selectable() const {return !(flags & (FL_MENU_INACTIVE|FL_MENU_INVISIBLE|FL_MENU_HEADLINE));}

  void headline(bool yes) {
    if (yes) flags |= FL_MENU_HEADLINE; else flags &= ~FL_MENU_HEADLINE;
  }
  int headline() const { return (flags & FL_MENU_HEADLINE); }

  // compatibility for FLUID so it can set the image of a menu item...

  void image(Fl_Image* image) {image->label(this);}

  void image(Fl_Image& image) {image.label(this);}

  // used by menubar:
  int measure(int* h, const Fl_Menu_*) const;
  void draw(int x, int y, int w, int h, const Fl_Menu_*, int t=0) const;

  // popup menus without using an Fl_Menu_ widget:
  const Fl_Menu_Item* popup(
    int X, int Y,
    const char *title = 0,
    const Fl_Menu_Item* picked=0,
    const Fl_Menu_* = 0) const;
  const Fl_Menu_Item* pulldown(
    int X, int Y, int W, int H,
    const Fl_Menu_Item* picked = 0,
    const Fl_Menu_* = 0,
    const Fl_Menu_Item* title = 0,
    int menubar=0) const;
  const Fl_Menu_Item* test_shortcut() const;
  const Fl_Menu_Item* find_shortcut(int *ip=0, const bool require_alt = false) const;

  void do_callback(Fl_Widget* o, Fl_Callback_Reason reason=FL_REASON_UNKNOWN) const {
    Fl::callback_reason_ = reason; callback_(o, user_data_);
  }

  void do_callback(Fl_Widget* o, void* arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN) const {
    Fl::callback_reason_ = reason; callback_(o, arg);
  }

  void do_callback(Fl_Widget* o, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN) const {
    Fl::callback_reason_ = reason; callback_(o, (void*)(fl_intptr_t)arg);
  }

  int insert(int,const char*,int,Fl_Callback*,void* =0, int =0);
  int add(const char*, int shortcut, Fl_Callback*, void* =0, int = 0);

  int add(const char*a, const char* b, Fl_Callback* c,
          void* d = 0, int e = 0) {
    return add(a,fl_old_shortcut(b),c,d,e);}

  int size() const ;
};

typedef Fl_Menu_Item Fl_Menu; // back compatibility

enum {  // back-compatibility enum:
  FL_PUP_NONE    = 0,
  FL_PUP_GREY    = FL_MENU_INACTIVE,
  FL_PUP_GRAY    = FL_MENU_INACTIVE,
  FL_MENU_BOX    = FL_MENU_TOGGLE,
  FL_PUP_BOX     = FL_MENU_TOGGLE,
```

```
  FL_MENU_CHECK = FL_MENU_VALUE,
  FL_PUP_CHECK  = FL_MENU_VALUE,
  FL_PUP_RADIO  = FL_MENU_RADIO,
  FL_PUP_INVISIBLE = FL_MENU_INVISIBLE,
  FL_PUP_SUBMENU = FL_SUBMENU_POINTER
};

#endif
```

14.95 Fl_Menu_Window.H

```
//
// Menu window header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Menu_Window widget . */

#ifndef Fl_Menu_Window_H
#define Fl_Menu_Window_H

#include "Fl_Single_Window.H"

class FL_EXPORT Fl_Menu_Window : public Fl_Single_Window {
public:
  ~Fl_Menu_Window();
  Fl_Menu_Window(int W, int H, const char *l = 0);
  Fl_Menu_Window(int X, int Y, int W, int H, const char *l = 0);
};

#endif
```

14.96 fl_message.H

```
//
// Standard message header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2023 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef _FL_fl_message_H_
#define _FL_fl_message_H_

#include "fl_ask.H"

#endif
```

14.97 Fl_Multi_Browser.H

```
//
// Multi browser header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
```

```
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Multi_Browser widget . */

#ifndef Fl_Multi_Browser_H
#define Fl_Multi_Browser_H

#include "Fl_Browser.H"

class FL_EXPORT Fl_Multi_Browser : public Fl_Browser {
public:
    Fl_Multi_Browser(int X,int Y,int W,int H,const char *L=0);
};

#endif
```

14.98 Fl_Multi_Label.H

```
//
// Multi-label header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2023 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef Fl_Multi_Label_H
#define Fl_Multi_Label_H

class Fl_Widget;
struct Fl_Menu_Item;

struct FL_EXPORT Fl_Multi_Label {
  const char* labela;
  const char* labelb;
  uchar typea;
  uchar typeb;

  // This method is used to associate a Fl_Multi_Label with a Fl_Widget.
  void label(Fl_Widget*);

  // This method is used to associate a Fl_Multi_Label with a Fl_Menu_Item.
  void label(Fl_Menu_Item*);
};

#endif
```

14.99 Fl_Multiline_Input.H

```
//
// Multiline input header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2011 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Multiline_Input widget . */
```

```

#ifndef Fl_Multiline_Input_H
#define Fl_Multiline_Input_H

#include "Fl_Input.H"

class FL_EXPORT Fl_Multiline_Input : public Fl_Input {
public:
  Fl_Multiline_Input(int X,int Y,int W,int H,const char *l = 0);
};

#endif
```

14.100 Fl_Multiline_Output.H

```
//
// Multi line output header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2011 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Multiline_Output widget . */

#ifndef Fl_Multiline_Output_H
#define Fl_Multiline_Output_H

#include "Fl_Output.H"

class FL_EXPORT Fl_Multiline_Output : public Fl_Output {
public:

  Fl_Multiline_Output(int X,int Y,int W,int H,const char *l = 0);
};

#endif
```

14.101 Fl_Native_File_Chooser.H File Reference

Fl_Native_File_Chooser widget.

```
#include <FL/Fl_Export.H>
#include <FL/Fl_File_Chooser.H>
```

Classes

- class Fl_Native_File_Chooser

  This class lets an FLTK application easily and consistently access the operating system's native file chooser.

14.101.1 Detailed Description

Fl_Native_File_Chooser widget.

14.102 Fl_Native_File_Chooser.H

Go to the documentation of this file.

```
//
// FLTK native OS file chooser widget
//
// Copyright 2004 Greg Ercolano.
// Copyright 2005-2024 by Bill Spitzak and others.
//
```

```
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* Implementation note:

 class Fl_Native_File_Chooser  <== public API used by applications

 class Fl_Native_File_Chooser_Driver   <== virtual API that a platform may implement
                                           this API has a do-nothing default implementation

        class Fl_Native_File_Chooser_FLTK_Driver   <== this API implementation is the default FLTK file chooser
                class Fl_GTK_Native_File_Chooser_Driver  <== this API implementation runs a GTK file chooser
                class Fl_Kdialog_Native_File_Chooser_Driver  <== this API implementation runs a KDE file chooser
                                        it is determined at run-time if the GTK dynamic libraries are available
                                        and the KDE file chooser runs under the KDE desktop

        class Fl_Quartz_Native_File_Chooser_Driver  <== this API implementation runs a Mac OS X file chooser

        class Fl_WinAPI_Native_File_Chooser_Driver  <== this API implementation runs a Windows file chooser


 Each platform must implement the constructor of the Fl_Native_File_Chooser class.
 This particular implementation:

 Fl_Native_File_Chooser::Fl_Native_File_Chooser(int val) {
  platform_fnfc = new Fl_Native_File_Chooser_FLTK_Driver(val);
 }

 can be used by any platform.
 No more code is required. The cross-platform Fl_Native_File_Chooser_FLTK.cxx file must be compiled in libfltk,
 and the default FLTK file chooser will be used.

 This other implementation:
 Fl_Native_File_Chooser::Fl_Native_File_Chooser(int val) {
   platform_fnfc = 0;
 }
 can be used by a platform that needs no file chooser.
 */

#ifndef FL_NATIVE_FILE_CHOOSER_H
#define FL_NATIVE_FILE_CHOOSER_H

#include <FL/Fl_Export.H>
#include <FL/Fl_File_Chooser.H>

class Fl_Native_File_Chooser_Driver;

class FL_EXPORT Fl_Native_File_Chooser {
private:
  Fl_Native_File_Chooser_Driver *platform_fnfc;
public:
  enum Type {
    BROWSE_FILE = 0,
    BROWSE_DIRECTORY,
    BROWSE_MULTI_FILE,
    BROWSE_MULTI_DIRECTORY,
    BROWSE_SAVE_FILE,
    BROWSE_SAVE_DIRECTORY
  };
  enum Option {
    NO_OPTIONS     = 0x0000,
    SAVEAS_CONFIRM = 0x0001,
    NEW_FOLDER     = 0x0002,
    PREVIEW        = 0x0004,
    USE_FILTER_EXT = 0x0008
  };
  static const char *file_exists_message;

  Fl_Native_File_Chooser(int val = BROWSE_FILE); // each platorm implements it
  ~Fl_Native_File_Chooser();
  void type(int t);
  int type() const ;
```

```
  void options(int o);
  int options() const;
  int count() const;
  const char *filename() const ;
  const char *filename(int i) const ;
  void directory(const char *val) ;
  const char *directory() const;
  void title(const char *t);
  const char* title() const;
  const char *filter() const ;
  void filter(const char *f);
  int filters() const ;
  void filter_value(int i) ;
  int filter_value() const ;
  void preset_file(const char*f) ;
  const char* preset_file() const;
  const char *errmsg() const ;
  int show() ;
  };

class Fl_Native_File_Chooser_Driver {
protected:
  static void chrcat(char *s, char c);
  static char *strapp(char *s, const char *val);
  static char *strfree(char *val);
  static char *strnew(const char *val);
public:
  Fl_Native_File_Chooser_Driver(int) {}
  virtual ~Fl_Native_File_Chooser_Driver() {}
  virtual void type(int) {}
  virtual int type() const  {return 0;}
  virtual void options(int) {}
  virtual int options() const {return 0;}
  virtual int count() const {return 0;}
  virtual const char *filename() const  {return 0;}
  virtual const char *filename(int) const  {return 0;}
  virtual void directory(const char *)  {}
  virtual const char *directory() const {return 0;}
  virtual void title(const char *) {}
  virtual const char* title() const {return 0;}
  virtual const char *filter() const  {return 0;}
  virtual void filter(const char *) {}
  virtual int filters() const  {return 0;}
  virtual void filter_value(int)  {}
  virtual int filter_value() const  {return 0;}
  virtual void preset_file(const char*)  {}
  virtual const char* preset_file() const {return 0;}
  virtual const char *errmsg() const  {return 0;}
  virtual int show()  {return 1;}
};

class Fl_Native_File_Chooser_FLTK_Driver : public Fl_Native_File_Chooser_Driver {
private:
  void errmsg(const char *msg);
  int type_fl_file(int val);
  int exist_dialog();
  void parse_filter();
protected:
  int   _btype;                 // kind-of browser to show()
  int   _options;               // general options
  int   _nfilters;
  char *_filter;                // user supplied filter
  char *_parsedfilt;            // parsed filter
  int   _filtvalue;             // selected filter
  char *_preset_file;
  char *_prevvalue;             // Returned filename
  char *_directory;
  char *_errmsg;                // error message
  Fl_File_Chooser *_file_chooser;
public:
  Fl_Native_File_Chooser_FLTK_Driver(int val);
  virtual ~Fl_Native_File_Chooser_FLTK_Driver();
  void type(int t) override;
  int type() const override;
  void options(int o) override;
  int options() const override;
  int count() const override;
  const char *filename() const override;
  const char *filename(int i) const override;
  void directory(const char *val) override;
  const char *directory() const override;
  void title(const char *t) override;
  const char* title() const override;
  const char *filter() const override;
  void filter(const char *f) override;
  int filters() const override;
  void filter_value(int i) override;
```

```
  int filter_value() const override;
  void preset_file(const char*f) override;
  const char* preset_file() const override;
  const char *errmsg() const override;
  int show() override;
};

#endif /*FL_NATIVE_FILE_CHOOSER_H*/
```

14.103 Fl_Nice_Slider.H

```
//
// "Nice" slider header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Nice_Slider widget . */

#ifndef Fl_Nice_Slider_H
#define Fl_Nice_Slider_H

#include "Fl_Slider.H"

class FL_EXPORT Fl_Nice_Slider : public Fl_Slider {
public:
  Fl_Nice_Slider(int X,int Y,int W,int H,const char *L=0);
};

#endif
```

14.104 Fl_Object.H

```
//
// Old Fl_Object header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

// This file is provided for back compatibility only.  Please use Fl_Widget
#ifndef Fl_Object
#define Fl_Object Fl_Widget
#endif
#include "Fl_Widget.H"
```

14.105 Fl_Output.H

```
//
// Output header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2022 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
```

```
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Output widget . */

#ifndef Fl_Output_H
#define Fl_Output_H

#include "Fl_Input.H"
class FL_EXPORT Fl_Output : public Fl_Input {
public:
    Fl_Output(int X,int Y,int W,int H, const char *l = 0);
};

#endif
```

14.106 Fl_Overlay_Window.H

```
//
// Overlay window header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Overlay_Window class . */

#ifndef Fl_Overlay_Window_H
#define Fl_Overlay_Window_H

#include "Fl_Double_Window.H"

class FL_EXPORT Fl_Overlay_Window : public Fl_Double_Window {
#ifndef FL_DOXYGEN
  friend class _Fl_Overlay;
  friend class Fl_Window_Driver;
#endif
public:
  virtual void draw_overlay() = 0;
private:
  Fl_Window *overlay_;
public:
  void show() override;
  void hide() override;
  void flush() override;
  void resize(int,int,int,int) override;
  ~Fl_Overlay_Window();
  int can_do_overlay();
  void redraw_overlay();
protected:
  Fl_Overlay_Window(int W, int H, const char *l=0);
  Fl_Overlay_Window(int X, int Y, int W, int H, const char *l=0);
public:
  void show(int a, char **b) {Fl_Double_Window::show(a,b);}
  Fl_Overlay_Window *as_overlay_window() override {return this; }
};

#endif
```

14.107 Fl_Pack.H

```
//
// Pack header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2020 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
```

```
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Pack widget . */

#ifndef Fl_Pack_H
#define Fl_Pack_H

#include <FL/Fl_Group.H>
#include <FL/Fl_Rect.H>

class FL_EXPORT Fl_Pack : public Fl_Group {
  int spacing_;

public:
  enum { // values for type(int)
    VERTICAL = 0,
    HORIZONTAL = 1
  };

protected:
  void draw_filler_(const Fl_Rect& rect);
  void draw() override;

public:
  Fl_Pack(int X, int Y, int W, int H, const char *L = 0);

  int spacing() const {return spacing_;}

  void spacing(int i) {spacing_ = i;}

  uchar horizontal() const {return type();}

  void resize(int X, int Y, int W, int H) override;
  void clear() { Fl_Group::clear(); resizable(NULL); }
};

#endif
```

14.108 Fl_Paged_Device.H File Reference

declaration of class Fl_Paged_Device.

`#include <FL/Fl_Widget_Surface.H>`

Classes

- class Fl_Paged_Device

  Represents page-structured drawing surfaces.
- struct Fl_Paged_Device::page_format

  width, height and name of a page format

Macros

- #define **NO_PAGE_FORMATS** 30 /∗ MSVC6 compilation fix ∗/

  Number of elements in enum Page_Format.

14.108.1 Detailed Description

declaration of class Fl_Paged_Device.

14.109 Fl_Paged_Device.H

Go to the documentation of this file.

```
00001 //
```

```
// Printing support for the Fast Light Tool Kit (FLTK).
//
// Copyright 2010-2024 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef Fl_Paged_Device_H
#define Fl_Paged_Device_H

#include <FL/Fl_Widget_Surface.H>


#define NO_PAGE_FORMATS 30 /* MSVC6 compilation fix */

class FL_EXPORT Fl_Paged_Device : public Fl_Widget_Surface {
protected:
  Fl_Paged_Device() : Fl_Widget_Surface(NULL) {}
public:
  enum Page_Format {
    A0 = 0,
    A1,
    A2,
    A3,
    A4,
    A5,
    A6,
    A7,
    A8,
    A9,
    B0,
    B1,
    B2,
    B3,
    B4,
    B5,
    B6,
    B7,
    B8,
    B9,
    B10,
    C5E,
    DLE,
    EXECUTIVE,
    FOLIO,
    LEDGER,
    LEGAL,
    LETTER,
    TABLOID,
    ENVELOPE,
    MEDIA = 0x1000
  };
  enum Page_Layout {
    PORTRAIT = 0,
    LANDSCAPE = 0x100,
    REVERSED = 0x200,
    ORIENTATION = 0x300
  };

  typedef struct {
    int width;
    int height;
    const char *name;
  } page_format;
  static const page_format page_formats[NO_PAGE_FORMATS];
  virtual ~Fl_Paged_Device() {}
  virtual int begin_job(int pagecount = 0, int *frompage = NULL, int *topage = NULL, char
**perr_message = NULL);
  int start_job(int pagecount = 0, int *frompage = NULL, int *topage = NULL, char **perr_message =
NULL) {
    return begin_job(pagecount, frompage, topage, perr_message);
  }
  virtual int begin_page(void);
  int start_page() {return begin_page();}
  virtual void margins(int *left, int *top, int *right, int *bottom);
  virtual void scale(float scale_x, float scale_y = 0.);
  virtual void rotate(float angle);
  void print_widget(Fl_Widget* widget, int delta_x = 0, int delta_y = 0) {draw(widget, delta_x,
delta_y);}
```

```
  void print_window(Fl_Window *win, int x_off = 0, int y_off = 0) {
    draw_decorated_window(win, x_off, y_off);
  }
  virtual int end_page (void);
  virtual void end_job (void);
};

#endif // Fl_Paged_Device_H
```

14.110 Fl_PDF_File_Surface.H

```
//
// Declaration of class Fl_PDF_File_Surface for the Fast Light Tool Kit (FLTK).
//
// Copyright 2024-2026 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef PDF_FILE_SURFACE_H
#define PDF_FILE_SURFACE_H

#include <FL/Fl_Paged_Device.H>

class FL_EXPORT Fl_PDF_File_Surface : public Fl_Paged_Device {
private:
  const char **out_filename_;
  Fl_Paged_Device *platform_surface_;
  static Fl_Paged_Device *new_platform_pdf_surface_(const char ***);
public:
  static const char * format_dialog_title;
  static const char * format_dialog_page_size;
  static const char * format_dialog_orientation;
  static const char * format_dialog_default;
  Fl_PDF_File_Surface();
  ~Fl_PDF_File_Surface();
  int begin_job(const char* defaultfilename, char **perr = NULL);
  int begin_job(int, int *, int *, char **) override {return 1;}
  int begin_document(const char* pathname,
                     enum Fl_Paged_Device::Page_Format format = Fl_Paged_Device::A4,
                     enum Fl_Paged_Device::Page_Layout layout = Fl_Paged_Device::PORTRAIT,
                     char **perr = NULL);
  int printable_rect(int *w, int *h) override { return platform_surface_->printable_rect(w,h); }
  void margins(int *left, int *top, int *right, int *bottom) override {
    platform_surface_->margins(left,top,right,bottom);
  }
  void origin(int x, int y) override {platform_surface_->origin(x, y);}
  void origin(int *x, int *y) override {platform_surface_->origin(x, y);}
  void scale(float s_x, float s_y = 0) override {platform_surface_->scale(s_x, s_y);}
  void rotate(float angle) override {platform_surface_->rotate(angle);}
  void translate(int x, int y) override {platform_surface_->translate(x, y);}
  void untranslate() override {platform_surface_->untranslate();}
  int begin_page(void) override {return platform_surface_->begin_page();}
  int end_page(void) override {return platform_surface_->end_page();}
  void end_job(void) override {return platform_surface_->end_job();}
  inline const char *pdf_filename() const { return *out_filename_; }
  void set_current() override { if (platform_surface_) platform_surface_->set_current(); }
  bool is_current() override { return surface() == platform_surface_; }
};

#endif // PDF_FILE_SURFACE_H
```

14.111 Fl_Pixmap.H

```
//
// Pixmap header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2026 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
```

```
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Pixmap widget . */

#ifndef Fl_Pixmap_H
#define Fl_Pixmap_H
#  include "Fl_Image.H"

class Fl_Widget;
struct Fl_Menu_Item;

// Older C++ compilers don't support the explicit keyword... :(
#  if defined(__sgi) && !defined(_COMPILER_VERSION)
#    define explicit
#  endif // __sgi && !_COMPILER_VERSION

class FL_EXPORT Fl_Pixmap : public Fl_Image {
  friend class Fl_Graphics_Driver;
  void copy_data();
  void delete_data();
  void set_data(const char * const *p);

protected:
  void measure();

public:

  int alloc_data; // Non-zero if data was allocated

private:
  // for internal use
  fl_uintptr_t id_;
  fl_uintptr_t mask_;
  int cache_w_, cache_h_; // size of pixmap when cached

public:

  explicit Fl_Pixmap(char * const * D) : Fl_Image(-1,0,1), alloc_data(0), id_(0), mask_(0)
{set_data((const char*const*)D); measure();}
  explicit Fl_Pixmap(uchar* const * D) : Fl_Image(-1,0,1), alloc_data(0), id_(0), mask_(0)
{set_data((const char*const*)D); measure();}
  explicit Fl_Pixmap(const char * const * D) : Fl_Image(-1,0,1), alloc_data(0), id_(0), mask_(0)
{set_data((const char*const*)D); measure();}
  explicit Fl_Pixmap(const uchar* const * D) : Fl_Image(-1,0,1), alloc_data(0), id_(0), mask_(0)
{set_data((const char*const*)D); measure();}
  virtual ~Fl_Pixmap();
  Fl_Image *copy(int W, int H) const override;
  Fl_Image *copy() const { return Fl_Image::copy(); }
  void color_average(Fl_Color c, float i) override;
  void desaturate() override;
  void draw(int X, int Y, int W, int H, int cx=0, int cy=0) override;
  void draw(int X, int Y) {draw(X, Y, w(), h(), 0, 0);}
  void label(Fl_Widget*w) override;
  void label(Fl_Menu_Item*m) override;
  void uncache() override;
  int cache_w() const { return cache_w_; }
  int cache_h() const { return cache_h_; }
};

#endif
```

14.112 Fl_Plugin.H

```
//
// A Plugin system for FLTK, implemented in Fl_Preferences.cxx.
//
// Copyright 2002-2023 by Matthias Melcher.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
```

```
   Fl_Plugin class . */

#ifndef Fl_Plugin_H
#define Fl_Plugin_H

#include "Fl_Preferences.H"

#include <string>
#include <vector>


class FL_EXPORT Fl_Plugin  {
  Fl_Preferences::ID id;
public:
  Fl_Plugin(const char *klass, const char *name);
  virtual ~Fl_Plugin();
};


class FL_EXPORT Fl_Plugin_Manager : public Fl_Preferences {
public:
  Fl_Plugin_Manager(const char *klass);
  ~Fl_Plugin_Manager();

  int plugins() { return groups(); }
  Fl_Plugin *plugin(int index);
  Fl_Plugin *plugin(const char *name);
  Fl_Preferences::ID addPlugin(const char *name, Fl_Plugin *plugin);

  static void removePlugin(Fl_Preferences::ID id);
  static int load(const char *filename);
  static int loadAll(const char *dirpath, const char *pattern=0);
  static std::vector<std::string> klass_list();
};


#endif // !Fl_Preferences_H
```

14.113 Fl_PNG_Image.H

```
//
// PNG image header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2023 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_PNG_Image class . */

#ifndef Fl_PNG_Image_H
#define Fl_PNG_Image_H
#  include "Fl_Image.H"
#  include <FL/Fl_Plugin.H>

class FL_EXPORT Fl_PNG_Image : public Fl_RGB_Image {
  friend class Fl_ICO_Image;
public:

  Fl_PNG_Image(const char* filename);
  Fl_PNG_Image (const char *name_png, const unsigned char *buffer, int datasize);
private:
  Fl_PNG_Image(const char *filename, int offset); // used by Fl_ICO_Image
  void load_png_(const char *name_png, int offset, const unsigned char *buffer_png, int datasize);
};

// Support functions to write PNG image files (since 1.4.0)

FL_EXPORT int fl_write_png(const char *filename, Fl_RGB_Image *img);
FL_EXPORT int fl_write_png(const char *filename, const char *pixels, int w, int h, int d=3, int ld=0);
FL_EXPORT int fl_write_png(const char *filename, const unsigned char *pixels, int w, int h, int d=3,
int ld=0);

class Fl_Load_PNG_Plugin : public Fl_Plugin {
private:
  virtual const char *klass() { return "loadpng.fltk.org"; }
```

```
  virtual const char *name() = 0;
protected:
  Fl_Load_PNG_Plugin(const char *pluginName)  : Fl_Plugin(klass(), pluginName) { }
public:
  virtual void *png_load_from_memory(const char *data, int size) = 0;
};

#endif
```

14.114 Fl_PNM_Image.H

```
//
// PNM image header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_PNM_Image class . */

#ifndef Fl_PNM_Image_H
#define Fl_PNM_Image_H
#  include "Fl_Image.H"

class FL_EXPORT Fl_PNM_Image : public Fl_RGB_Image {

  public:

  Fl_PNM_Image(const char* filename);
};

#endif
```

14.115 Fl_Positioner.H

```
//
// Positioner header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2022 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Positioner widget . */

#ifndef Fl_Positioner_H
#define Fl_Positioner_H

#ifndef Fl_Widget_H
#include "Fl_Widget.H"
#endif

class FL_EXPORT Fl_Positioner : public Fl_Widget {

  double xmin, ymin;
  double xmax, ymax;
  double xvalue_, yvalue_;
  double xstep_, ystep_;

protected:

  // these allow subclasses to put the dial in a smaller area:
  void draw(int, int, int, int);
```

```
  int handle(int, int, int, int, int);
  void draw() override;

public:

  int handle(int) override;
  Fl_Positioner(int x,int y,int w,int h, const char *l=0);
  double xvalue() const {return xvalue_;}
  double yvalue() const {return yvalue_;}
  int xvalue(double);
  int yvalue(double);
  int value(double,double);
  void xbounds(double, double);
  double xminimum() const {return xmin;}
  void xminimum(double a) {xbounds(a,xmax);}
  double xmaximum() const {return xmax;}
  void xmaximum(double a) {xbounds(xmin,a);}
  void ybounds(double, double);
  double yminimum() const {return ymin;}
  void yminimum(double a) {ybounds(a, ymax);}
  double ymaximum() const {return ymax;}
  void ymaximum(double a) {ybounds(ymin, a);}
  void xstep(double a) {xstep_ = a;}
  void ystep(double a) {ystep_ = a;}
};

#endif
```

14.116 Fl_PostScript.H File Reference

declaration of classes Fl_PostScript_File_Device and Fl_EPS_File_Surface.

```
#include <FL/Fl_Paged_Device.H>
#include <FL/fl_draw.H>
#include <stdarg.h>
```

Classes

- class Fl_EPS_File_Surface

  Encapsulated PostScript drawing surface.
- class Fl_PostScript_File_Device

  To send graphical output to a PostScript file.

Typedefs

- typedef int(∗ Fl_PostScript_Close_Command) (FILE ∗)

  Signature of functions FLTK may use to close FILE variables after PostScript/EPS output.

14.116.1 Detailed Description

declaration of classes Fl_PostScript_File_Device and Fl_EPS_File_Surface.

14.116.2 Typedef Documentation

14.116.2.1 Fl_PostScript_Close_Command

```
typedef int(* Fl_PostScript_Close_Command) (FILE *)
```

Signature of functions FLTK may use to close FILE variables after PostScript/EPS output.
A non-null return value indicates output error.

See also

Fl_PostScript_File_Device::close_command() and Fl_EPS_File_Surface::Fl_EPS_File_Surface().

14.117 Fl_PostScript.H

Go to the documentation of this file.

```
00001 //
00002 // Support for graphics output to PostScript file for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 2010-2020 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file.  If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 //     https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 //     https://www.fltk.org/bugs.php
00015 //
00016
00021 #ifndef Fl_PostScript_H
00022 #define Fl_PostScript_H
00023
00024 #include <FL/Fl_Paged_Device.H>
00025 #include <FL/fl_draw.H>
00026 #include <stdarg.h>
00027
00032 extern "C" {
00033   typedef int (*Fl_PostScript_Close_Command)(FILE *);
00034 }
00035
00036 class Fl_PostScript_Graphics_Driver;
00037
00078 class FL_EXPORT Fl_PostScript_File_Device : public Fl_Paged_Device {
00079 private:
00080   // memorize the display's current font to restore it when the object ceases being current
00081   Fl_Font display_font_;
00082   Fl_Fontsize display_size_;
00083 protected:
00087   inline Fl_PostScript_Graphics_Driver *driver() { return
      (Fl_PostScript_Graphics_Driver*)Fl_Surface_Device::driver(); }
00088 public:
00090     Fl_PostScript_File_Device();
00092   ~Fl_PostScript_File_Device();
00094   int begin_job(int pagecount, int* from, int* to, char **perr_message) override;
00104  int begin_job(int pagecount = 0, enum Fl_Paged_Device::Page_Format format = Fl_Paged_Device::A4,
00105                enum Fl_Paged_Device::Page_Layout layout = Fl_Paged_Device::PORTRAIT);
00108   int start_job(int pagecount = 0, enum Fl_Paged_Device::Page_Format format = Fl_Paged_Device::A4,
00109                 enum Fl_Paged_Device::Page_Layout layout = Fl_Paged_Device::PORTRAIT) {
00110     return begin_job(pagecount, format, layout);
00111   }
00122   int begin_job(FILE *ps_output, int pagecount = 0, enum Fl_Paged_Device::Page_Format format =
      Fl_Paged_Device::A4,
00123                 enum Fl_Paged_Device::Page_Layout layout = Fl_Paged_Device::PORTRAIT);
00126   int start_job(FILE *ps_output, int pagecount = 0, enum Fl_Paged_Device::Page_Format format =
      Fl_Paged_Device::A4,
00127                 enum Fl_Paged_Device::Page_Layout layout = Fl_Paged_Device::PORTRAIT) {
00128     return begin_job(ps_output, pagecount, format, layout);
00129   }
00130
00131   int begin_page (void) override;
00132   int printable_rect(int *w, int *h) override;
00133   void margins(int *left, int *top, int *right, int *bottom) override;
00134   void origin(int *x, int *y) override;
00135   void origin(int x, int y) override;
00136   void scale (float scale_x, float scale_y = 0.) override;
00137   void rotate(float angle) override;
00138   void translate(int x, int y) override;
00139   void untranslate(void) override;
00140   int end_page (void) override;
00144   void end_job(void) override;
00146   static const char *file_chooser_title;
00148   FILE *file();
00150   void close_command(Fl_PostScript_Close_Command cmd);
00151   void set_current() override;
00152   void end_current() override;
00153 };
00154
00172 class FL_EXPORT Fl_EPS_File_Surface : public Fl_Widget_Surface {
00173 protected:
00175   inline Fl_PostScript_Graphics_Driver *driver() { return
      (Fl_PostScript_Graphics_Driver*)Fl_Surface_Device::driver(); }
00176 public:
00191   Fl_EPS_File_Surface(int width, int height, FILE *eps_output,
00192                       Fl_Color background = FL_WHITE, Fl_PostScript_Close_Command closef = NULL);
00200   ~Fl_EPS_File_Surface();
00201   int printable_rect(int *w, int *h) override;
```

```
  FILE *file();
  void origin(int x, int y) override;
  void origin(int *px, int *py) override;
  void translate(int x, int y) override;
  void untranslate() override;
  int close();
};

#endif // Fl_PostScript_H
```

14.118 Fl_Preferences.H

```
//
// Preferences implementation for the Fast Light Tool Kit (FLTK).
//
// Copyright 2002-2023 by Matthias Melcher.
// Copyright 2023-2026 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Preferences class . */

#ifndef Fl_Preferences_H
#  define Fl_Preferences_H

#  include <stdio.h>
#  include "Fl_Export.H"
#  include "fl_attr.h"

#include <string>

class FL_EXPORT Fl_Preferences {

public:
  enum Root {
    UNKNOWN_ROOT_TYPE = -1,
    SYSTEM            =  0,
    USER,
    MEMORY,
    ROOT_MASK     = 0x00FF,
    CORE          = 0x0100,
    C_LOCALE      = 0x1000,
    CLEAR         = 0x2000,
    SYSTEM_L      = SYSTEM | C_LOCALE,
    USER_L        = USER | C_LOCALE,
    CORE_SYSTEM_L = CORE | SYSTEM_L,
    CORE_USER_L   = CORE | USER_L,
    CORE_SYSTEM   = CORE | SYSTEM,
    CORE_USER     = CORE | USER
  };

  typedef void *ID;

  static const char *new_UUID();

  static const unsigned int NONE = 0x0000;
  static const unsigned int USER_READ_OK = 0x0001;
  static const unsigned int USER_WRITE_OK = 0x0002;
  static const unsigned int USER_OK = USER_READ_OK | USER_WRITE_OK;
  static const unsigned int SYSTEM_READ_OK = 0x0004;
  static const unsigned int SYSTEM_WRITE_OK = 0x0008;
  static const unsigned int SYSTEM_OK = SYSTEM_READ_OK | SYSTEM_WRITE_OK;
  static const unsigned int APP_OK = SYSTEM_OK | USER_OK;
  static const unsigned int CORE_READ_OK = 0x0010;
  static const unsigned int CORE_WRITE_OK = 0x0020;
  static const unsigned int CORE_OK = CORE_READ_OK | CORE_WRITE_OK;
  static const unsigned int ALL_READ_OK = USER_READ_OK | SYSTEM_READ_OK | CORE_READ_OK;
  static const unsigned int ALL_WRITE_OK = USER_WRITE_OK | SYSTEM_WRITE_OK | CORE_WRITE_OK;
  static const unsigned int ALL = ALL_READ_OK | ALL_WRITE_OK;

  static void file_access(unsigned int flags);
  static unsigned int file_access();
  static Root filename( char *buffer, size_t buffer_size, Root root, const char *vendor, const char
*application );

```

```
  Fl_Preferences( Root root, const char *vendor, const char *application );
  Fl_Preferences( const char *path, const char *vendor, const char *application, Root flags );
  Fl_Preferences( Fl_Preferences &parent, const char *group );
  Fl_Preferences( Fl_Preferences *parent, const char *group );
  Fl_Preferences( Fl_Preferences &parent, int groupIndex );
  Fl_Preferences( Fl_Preferences *parent, int groupIndex );
  Fl_Preferences(const Fl_Preferences&);
  Fl_Preferences( ID id );
  virtual ~Fl_Preferences();

  FL_DEPRECATED("since 1.4.0 - use Fl_Preferences(path, vendor, application, flags) instead",
                Fl_Preferences( const char *path, const char *vendor, const char *application ) );

  Root filename( char *buffer, size_t buffer_size);

  ID id() const { return (ID)node; }

  static char remove(ID id_) { return ((Node*)id_)->remove(); }

  const char *name() const { return node->name(); }

  const char *path() const { return node->path(); }

  int groups();
  const char *group( int num_group );
  char group_exists( const char *key );
  char delete_group( const char *group );
  char delete_all_groups();

  int entries();
  const char *entry( int index );
  char entry_exists( const char *key );
  char delete_entry( const char *entry );
  char delete_all_entries();

  char clear();

  char set( const char *entry, int value );
  char set( const char *entry, float value );
  char set( const char *entry, float value, int precision );
  char set( const char *entry, double value );
  char set( const char *entry, double value, int precision );
  char set( const char *entry, const char *value );
  char set( const char *entry, const void *value, int size );
  char set( const char *entry, const std::string &value );

  char get( const char *entry, int &value, int defaultValue );
  char get( const char *entry, float &value,  float defaultValue );
  char get( const char *entry, double &value, double defaultValue );
  char get( const char *entry, char *&value,  const char *defaultValue );
  char get( const char *entry, char *value,   const char *defaultValue, int maxSize );
  char get( const char *entry, void *&value,  const void *defaultValue, int defaultSize );
  char get( const char *entry, void *value,   const void *defaultValue, int defaultSize, int maxSize
);
  char get( const char *entry, void *value,   const void *defaultValue, int defaultSize, int *size );
  char get( const char *entry, std::string &value, const std::string &defaultValue );

  int size( const char *entry );

  char get_userdata_path( char *path, int pathlen );
  char get_userdata_path( std::string& path );

  int flush();

  int dirty();

  static const char *newUUID() { return new_UUID(); }
  char groupExists( const char *key ) { return group_exists(key); }
  char deleteGroup( const char *group ) { return delete_group(group); }
  char deleteAllGroups() { return delete_all_groups(); }
  char entryExists( const char *key ) { return entry_exists(key); }
  char deleteEntry( const char *entry ) { return delete_entry(entry); }
  char deleteAllEntries() { return delete_all_entries(); }
  char getUserdataPath( char *path, int pathlen ) { return get_userdata_path(path, pathlen); }
  class FL_EXPORT Name {

    char *data_;

  public:
    Name( unsigned int n );
    Name( const char *format, ... );

    operator const char *() const { return data_; }
    ~Name();
  };

  struct Entry {
```

```
    char *name, *value;
  };

private:
  Fl_Preferences() : node(0), rootNode(0) { }
  Fl_Preferences &operator=(const Fl_Preferences&);

  static char nameBuffer[128];
  static char uuidBuffer[40];
  static Fl_Preferences *runtimePrefs;
  static unsigned int fileAccess_;

public:  // older Sun compilers need this (public definition of the following classes)
  class RootNode;

  class FL_EXPORT Node {        // a node contains a list to all its entries
                                // and all means to manage the tree structure
    Node *first_child_, *next_;
    union {                     // these two are mutually exclusive
      Node *parent_;            // top_ bit clear
      RootNode *root_node_;     // top_ bit set
    };
    char *path_;
    Entry *entry_;
    int nEntry_, NEntry_;
    unsigned char dirty_:1;
    unsigned char top_:1;
    unsigned char indexed_:1;
    // indexing routines
    Node **index_;
    int nIndex_, NIndex_;
    void createIndex();
    void updateIndex();
    void deleteIndex();
  public:
    static int lastEntrySet;
  public:
    Node( const char *path );
    ~Node();
    // node methods
    int write( FILE *f );
    const char *name() const;
    const char *path() const { return path_; }
    Node *find( const char *path );
    Node *search( const char *path, int offset=0 );
    Node *childNode( int ix );
    Node *addChild( const char *path );
    void setParent( Node *parent );
    Node *parent() const { return top_?0L:parent_; }
    void setRoot(RootNode *r) { root_node_ = r; top_ = 1; }
    RootNode *findRoot() const;
    char remove();
    char dirty() const;
    void clearDirtyFlags();
    void deleteAllChildren();
    // entry methods
    int nChildren() const;
    const char *child( int ix );
    void set( const char *name, const char *value );
    void set( const char *line );
    void add( const char *line );
    const char *get( const char *name ) const;
    int getEntry( const char *name ) const;
    char deleteEntry( const char *name );
    void deleteAllEntries();
    int nEntry() const { return nEntry_; }
    Entry &entry(int i) const { return entry_[i]; }
  };
  friend class Node;

  class FL_EXPORT RootNode {    // the root node manages file paths and basic reading and writing
    Fl_Preferences *prefs_;
    char *filename_;
    char *vendor_, *application_;
    Root root_type_;
  public:
    RootNode( Fl_Preferences *, Root root, const char *vendor, const char *application );
    RootNode( Fl_Preferences *, const char *path, const char *vendor, const char *application, Root
flags );
    RootNode( Fl_Preferences * );
    ~RootNode();
    int read();
    int write();
    char getPath( char *path, int pathlen );
    char *filename() const { return filename_; }
    Root root() const { return root_type_; }
  };
```

```
  friend class RootNode;

protected:
  Node *node;
  RootNode *rootNode;
};

#endif // !Fl_Preferences_H
```

14.119 Fl_Printer.H File Reference

declaration of class Fl_Printer.

`#include <FL/Fl_Paged_Device.H>`

Classes

- class Fl_Printer

  OS-independent print support.

14.119.1 Detailed Description

declaration of class Fl_Printer.

14.120 Fl_Printer.H

Go to the documentation of this file.

```
//
// Printing support for the Fast Light Tool Kit (FLTK).
//
// Copyright 2010-2016 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef Fl_Printer_H
#define Fl_Printer_H

#include <FL/Fl_Paged_Device.H>

class FL_EXPORT Fl_Printer : public Fl_Paged_Device {
private:
  Fl_Paged_Device *printer;
  static Fl_Paged_Device* newPrinterDriver(void);
public:
  Fl_Printer(void);
  int begin_job(int pagecount = 0, int *frompage = NULL, int *topage = NULL, char **perr_message =
    NULL) override;
  int begin_page(void) override;
  int printable_rect(int *w, int *h) override;
  void margins(int *left, int *top, int *right, int *bottom) override;
  void origin(int *x, int *y) override;
  void origin(int x, int y) override;
  void scale(float scale_x, float scale_y = 0.) override;
  void rotate(float angle) override;
  void translate(int x, int y) override;
  void untranslate(void) override;
  int end_page (void) override;
  void end_job (void) override;
  void set_current(void) override;
  bool is_current() override;

  static const char *dialog_title;
  static const char *dialog_printer;
  static const char *dialog_range;
  static const char *dialog_copies;
  static const char *dialog_all;
```

```
  static const char *dialog_pages;
  static const char *dialog_from;
  static const char *dialog_to;
  static const char *dialog_properties;
  static const char *dialog_copyNo;
  static const char *dialog_print_button;
  static const char *dialog_cancel_button;
  static const char *dialog_print_to_file;
  static const char *property_title;
  static const char *property_pagesize;
  static const char *property_mode;
  static const char *property_use;
  static const char *property_save;
  static const char *property_cancel;
  ~Fl_Printer(void);
};

#endif // Fl_Printer_H
```

14.121 Fl_Progress.H

```
//
// Progress bar widget definitions.
//
// Copyright 2000-2010 by Michael Sweet.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Progress widget . */

#ifndef _Fl_Progress_H_
#  define _Fl_Progress_H_

//
// Include necessary headers.
//

#include "Fl_Widget.H"


//
// Progress class...
//
class FL_EXPORT Fl_Progress : public Fl_Widget {

  float value_,
        minimum_,
        maximum_;

  protected:

  void draw() override;

  public:

  Fl_Progress(int x, int y, int w, int h, const char *l = 0);

  void  maximum(float v) { maximum_ = v; redraw(); }
  float maximum() const { return (maximum_); }

  void  minimum(float v) { minimum_ = v; redraw(); }
  float minimum() const { return (minimum_); }

  void  value(float v) { value_ = v; redraw(); }
  float value() const { return (value_); }
};

#endif // !_Fl_Progress_H_
```

14.122 Fl_Radio_Button.H

```
//
```

```
// Radio button header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2014 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Radio_Button widget . */

#ifndef Fl_Radio_Button_H
#define Fl_Radio_Button_H

#include "Fl_Button.H"

class FL_EXPORT Fl_Radio_Button : public Fl_Button {
public:
  Fl_Radio_Button(int X,int Y,int W,int H,const char *L=0);
};

#endif
```

14.123 Fl_Radio_Light_Button.H

```
//
// Radio light button header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2014 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Radio_Light_Button widget . */

#ifndef Fl_Radio_Light_Button_H
#define Fl_Radio_Light_Button_H

#include "Fl_Light_Button.H"

class FL_EXPORT Fl_Radio_Light_Button : public Fl_Light_Button {
public:
  Fl_Radio_Light_Button(int X,int Y,int W,int H,const char *l=0);
};

#endif
```

14.124 Fl_Radio_Round_Button.H

```
//
// Radio round button header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2014 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
```

```
   Fl_Radio_Round_Button widget . */

#ifndef Fl_Radio_Round_Button_H
#define Fl_Radio_Round_Button_H

#include "Fl_Round_Button.H"

class FL_EXPORT Fl_Radio_Round_Button : public Fl_Round_Button {
public:
  Fl_Radio_Round_Button(int X,int Y,int W,int H,const char *L=0);
};

#endif
```

14.125 Fl_Rect.H

```
//
// Fl_Rect header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2023 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef Fl_Rect_H
#define Fl_Rect_H

#include <FL/Fl_Widget.H>       // for c'tor based on Fl_Widget

class FL_EXPORT Fl_Rect {

  int x_;
  int y_;
  int w_;
  int h_;

public:

  Fl_Rect()
          : x_(0), y_(0), w_(0), h_(0) {}

  Fl_Rect(int W, int H)
          : x_(0), y_(0), w_(W), h_(H) {}

  Fl_Rect(int X, int Y, int W, int H)
          : x_(X), y_(Y), w_(W), h_(H) {}

  Fl_Rect(int X, int Y, int W, int H, Fl_Boxtype bt)
          : x_(X), y_(Y), w_(W), h_(H) {
    inset(bt);
  }

  Fl_Rect (const Fl_Widget& widget)
          : x_(widget.x()), y_(widget.y()), w_(widget.w()), h_(widget.h()) {}

  Fl_Rect (const Fl_Widget* const widget)
          : x_(widget->x()), y_(widget->y()), w_(widget->w()), h_(widget->h()) {}

  int x() const { return x_; }
  int y() const { return y_; }
  int w() const { return w_; }
  int h() const { return h_; }

  int r() const { return x_ + w_; }
  int b() const { return y_ + h_; }

  void x(int X) { x_ = X; }
  void y(int Y) { y_ = Y; }
  void w(int W) { w_ = W; }
  void h(int H) { h_ = H; }

  void r(int R) { w_ = R - x_; }
  void b(int B) { h_ = B - y_; }

  void inset(int d) {
    x_ += d;
    y_ += d;
```

```
    w_ -= 2 * d;
    h_ -= 2 * d;
  }

  void inset(Fl_Boxtype bt) {
    x_ += Fl::box_dx(bt);
    y_ += Fl::box_dy(bt);
    w_ -= Fl::box_dw(bt);
    h_ -= Fl::box_dh(bt);
  }

  void inset(int left, int top, int right, int bottom) {
    x_ += left;
    y_ += top;
    w_ -= (left + right);
    h_ -= (top + bottom);
  }

  friend bool operator==(const Fl_Rect& lhs, const Fl_Rect& rhs) {
    return (lhs.x_==rhs.x_) && (lhs.y_==rhs.y_) && (lhs.w_==rhs.w_) && (lhs.h_==rhs.h_);
  }

  friend bool operator!=(const Fl_Rect& lhs, const Fl_Rect& rhs) {
    return !(lhs==rhs);
  }

  bool contains(int X, int Y) const {
    return (X >= x_ && X < r() && Y >= y_ && Y < b());
  }

}; // class Fl_Rect

#endif // Fl_Rect_H
```

14.126 Fl_Repeat_Button.H

```
//
// Repeat button header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Repeat_Button widget . */

#ifndef Fl_Repeat_Button_H
#define Fl_Repeat_Button_H
#include "Fl.H"
#include "Fl_Button.H"

class FL_EXPORT Fl_Repeat_Button : public Fl_Button {
  static void repeat_callback(void *);
public:
  int handle(int) override;
  Fl_Repeat_Button(int X,int Y,int W,int H,const char *l=0);

  void deactivate() {
    Fl::remove_timeout(repeat_callback,this);
    Fl_Button::deactivate();
  }
};

#endif
```

14.127 Fl_Return_Button.H

```
//
// Return button header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2022 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
```

```
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Return_Button widget . */

#ifndef Fl_Return_Button_H
#define Fl_Return_Button_H
#include "Fl_Button.H"

class FL_EXPORT Fl_Return_Button : public Fl_Button {
protected:
  void draw() override;
public:
  int handle(int) override;
  Fl_Return_Button(int X, int Y, int W, int H,const char *l=0);
};

#endif
```

14.128 Fl_RGB_Image.H

```
//
// RGB Image header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef Fl_RGB_Image_H
# define Fl_RGB_Image_H
# include "Fl_Image.H"
#endif // !Fl_RGB_Image_H
```

14.129 Fl_Roller.H

```
//
// Roller header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2023 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Roller widget . */

#ifndef Fl_Roller_H
#define Fl_Roller_H

#ifndef Fl_Valuator_H
#include "Fl_Valuator.H"
#endif

class FL_EXPORT Fl_Roller : public Fl_Valuator {
protected:
  void draw() override;
public:
```

```
  int handle(int) override;
  Fl_Roller(int X,int Y,int W,int H,const char* L=0);
};

#endif
```

14.130 Fl_Round_Button.H

```
//
// Round button header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2022 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef Fl_Round_Button_H
#define Fl_Round_Button_H

#include "Fl_Light_Button.H"

class FL_EXPORT Fl_Round_Button : public Fl_Light_Button {
public:
  Fl_Round_Button(int x,int y,int w,int h,const char *l = 0);
};

#endif
```

14.131 Fl_Round_Clock.H

```
//
// Round clock header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Round_Clock widget . */

#ifndef Fl_Round_Clock_H
#define Fl_Round_Clock_H

#include "Fl_Clock.H"

class FL_EXPORT Fl_Round_Clock : public Fl_Clock {
public:
  Fl_Round_Clock(int X,int Y,int W,int H, const char *L = 0);
};

#endif
```

14.132 Fl_Scheme.H

```
//
// Fl_Scheme header for the Fast Light Tool Kit (FLTK).
//
// Copyright 2022-2025 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
```

```
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef _FL_Fl_Scheme_H_
#define _FL_Fl_Scheme_H_

#include <FL/Fl.H>

class FL_EXPORT Fl_Scheme {

private:

  static const char **names_;           // registered scheme names
  static int num_schemes_;              // number of registered schemes
  static int alloc_size_;               // number of allocated scheme name entries

protected:

  // const char *name_;                 // the scheme's name

  // protected constructor - not yet implemented
  // Fl_Scheme(const char *name);

public:

  // Static methods.

  // Some of these methods will replace the scheme related methods of class Fl,
  // for instance Fl::scheme() and Fl::is_scheme().
  // Backwards compatibility must be kept though.

  static const char **names();

  static int num_schemes() {
    if (!names_) names(); // force initialization
    return num_schemes_;
  }

  // Adding a scheme name must be a public static method in (since) FLTK 1.4.0.
  // This will later be protected or replaced by another method name.

  static int add_scheme_name(const char *name);

  // Set color average parameter of the 'plastic' scheme.
  // See GitHub Issue 464: "RFE: plastic scheme with faithful colors".
  // See also documentation and implementation in src/fl_plastic.cxx.

  static void plastic_color_average(int av);

}; // class Fl_Scheme

#endif // _FL_Fl_Scheme_H_
```

14.133 Fl_Scheme_Choice.H

```
//
// Scheme Choice header for the Fast Light Tool Kit (FLTK).
//
// Copyright 2022-2023 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef FL_Fl_Scheme_Choice_H_
#define FL_Fl_Scheme_Choice_H_

#include <FL/Fl.H>
#include <FL/Fl_Scheme.H>
#include <FL/Fl_Choice.H>

class FL_EXPORT Fl_Scheme_Choice : public Fl_Choice {

```

```
protected:
  static void scheme_cb_(Fl_Widget *w, void *);

public:
  Fl_Scheme_Choice(int X, int Y, int W, int H, const char *L = 0);
  int handle(int event) override;

  // set the current value according to the active scheme
  virtual void init_value();

}; // class Fl_Scheme_Choice

#endif // FL_Fl_Scheme_Choice_H_
```

14.134 Fl_Scroll.H

```
//
// Fl_Scroll header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2026 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Scroll widget . */

#ifndef Fl_Scroll_H
#define Fl_Scroll_H

#include "Fl_Group.H"
#include "Fl_Scrollbar.H"

class FL_EXPORT Fl_Scroll : public Fl_Group {

  int xposition_, yposition_;
  int oldx, oldy;
  int scrollbar_size_;
  static void hscrollbar_cb(Fl_Widget*, void*);
  static void scrollbar_cb(Fl_Widget*, void*);
  static void draw_clip(void*,int,int,int,int);

protected:      //  (STR#1895)

  typedef struct {
    int x, y, w, h;
    int pos;
    int size;
    int first;
    int total;
  } Scrollbar_Data;

  typedef struct {
    int scrollsize;
    Fl_Rect innerbox;
    Fl_Rect innerchild;
    Fl_Rect child;
    int hneeded;
    int vneeded;
    Scrollbar_Data hscroll;
    Scrollbar_Data vscroll;
  } ScrollInfo;
  void recalc_scrollbars(ScrollInfo &si) const;

protected:

  int on_insert(Fl_Widget*, int) override;
  int on_move(int, int) override;
  void fix_scrollbar_order();
  void bbox(int &, int &, int &, int &) const;
  void draw() override;

public:

  Fl_Scrollbar scrollbar;
  Fl_Scrollbar hscrollbar;

```

```
  void resize(int X, int Y, int W, int H) override;
  int handle(int) override;

  Fl_Scroll(int X, int Y, int W, int H, const char *L = 0);
  virtual ~Fl_Scroll();

  enum { // values for type()
    HORIZONTAL = 1,
    VERTICAL = 2,
    BOTH = 3,
    ALWAYS_ON = 4,
    HORIZONTAL_ALWAYS = 5,
    VERTICAL_ALWAYS = 6,
    BOTH_ALWAYS = 7
  };

  int xposition() const {return xposition_;}
  int yposition() const {return yposition_;}
  void scroll_to(int, int);
  void clear();

  /* delete child n (by index) */
  int delete_child(int n) override;

  int scrollbar_size() const {
      return(scrollbar_size_);
  }
  void scrollbar_size(int newSize) {
      if ( newSize != scrollbar_size_ ) redraw();
      scrollbar_size_ = newSize;
  }
};

#endif
```

14.135 Fl_Scrollbar.H

```
//
// Scroll bar header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Scrollbar widget . */

#ifndef Fl_Scrollbar_H
#define Fl_Scrollbar_H

#include "Fl_Slider.H"

class FL_EXPORT Fl_Scrollbar : public Fl_Slider {

  int linesize_;
  int pushed_;
  // error for fractional scrolling
  float scroll_err_ = 0.5f;
  static void timeout_cb(void*);
  void increment_cb();
protected:
  void draw() override;

public:

  Fl_Scrollbar(int X,int Y,int W,int H, const char *L = 0);
  ~Fl_Scrollbar();
  int handle(int) override;

  int value() const {return int(Fl_Slider::value());}

  int value(int p) {return int(Fl_Slider::value((double)p));}

  int value(int pos, int windowSize, int first_line, int total_lines) {
    return scrollvalue(pos, windowSize, first_line, total_lines);
  }
```

```
00092
00096   int linesize() const {return linesize_;}
00097
00103   void linesize(int i) {linesize_ = i;}
00104
00105 };
00106
00107 #endif
```

14.136 Fl_Secret_Input.H

```
00001 //
00002 // Secret input header file for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 1998-2011 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file.  If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 //     https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 //     https://www.fltk.org/bugs.php
00015 //
00016
00017 /* \file
00018    Fl_Secret_Input widget . */
00019
00020 #ifndef Fl_Secret_Input_H
00021 #define Fl_Secret_Input_H
00022
00023 #include "Fl_Input.H"
00024
00033 class FL_EXPORT Fl_Secret_Input : public Fl_Input {
00034 public:
00041   Fl_Secret_Input(int X,int Y,int W,int H,const char *l = 0);
00042   int handle(int) override;
00043 };
00044
00045 #endif
```

14.137 Fl_Select_Browser.H

```
00001 //
00002 // Select browser header file for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 1998-2010 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file.  If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 //     https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 //     https://www.fltk.org/bugs.php
00015 //
00016
00017 /* \file
00018    Fl_Select_Browser widget . */
00019
00020 #ifndef Fl_Select_Browser_H
00021 #define Fl_Select_Browser_H
00022
00023 #include "Fl_Browser.H"
00024
00033 class FL_EXPORT Fl_Select_Browser : public Fl_Browser {
00034 public:
00041   Fl_Select_Browser(int X,int Y,int W,int H,const char *L=0);
00042 };
00043
00044 #endif
```

14.138 Fl_Shared_Image.H File Reference

Fl_Shared_Image class.

```
#include "Fl_Image.H"
```

Classes

- class Fl_Shared_Image::Deleter

  Custom deleter for std::unique_ptr<>.
- class Fl_Shared_Image

  This class supports caching, loading, and drawing of image files.

Typedefs

- typedef Fl_Image *(* Fl_Shared_Handler) (const char *name, uchar *header, int headerlen)

  Test function (typedef) for adding new shared image formats.

Functions

- void fl_register_images ()

  Register the known image formats.

14.138.1 Detailed Description

Fl_Shared_Image class.

14.138.2 Typedef Documentation

14.138.2.1 Fl_Shared_Handler

```
typedef Fl_Image *(* Fl_Shared_Handler) (const char *name, uchar *header, int headerlen)
```

Test function (typedef) for adding new shared image formats.

This defines the function type you can use to add a handler for unknown image formats that can be opened and loaded as an Fl_Shared_Image.

fl_register_images() adds all image formats known to FLTK. Call Fl_Shared_Image::add_handler() to add your own check function to the list of known image formats.

Your function will be passed the filename (`name`), some `header` bytes already read from the image file and the size `headerlen` of the data read. The max value of size is implementation dependent. If your handler function needs to check more bytes you must open the image file yourself.

The provided buffer `header` must not be overwritten.

If your handler function can identify the file type you must open the file and return a valid Fl_Image or derived type, otherwise you must return `NULL`. Example:

```
static Fl_Image *check_my_image(const char *name,
                                uchar *header,
                                int headerlen) {
  // (test image type using header and headerlen)
  if (known) {
    // (load image data from file \p name)
    return new Fl_RGB_Image(data, ...);
  } else
    return 0;
}
// add your handler:
Fl_Shared_Image::add_handler(check_my_image);
```

Parameters

in	*name*	filename to be checked and opened if applicable
in	*header*	portion of the file that has already been read
in	*headerlen*	length of provided `header` data

Returns

valid Fl_Image or NULL.

See also

Fl_Shared_Image::add_handler()

14.138.3 Function Documentation

14.138.3.1 fl_register_images()

```
void fl_register_images ( ) [extern]
```

Register the known image formats.
This function is provided in the fltk_images library and registers all of the "extra" image file formats known to FLTK that are not part of the core FLTK library.
You may add your own image formats with Fl_Shared_Image::add_handler().

14.139 Fl_Shared_Image.H

Go to the documentation of this file.

```
00001 //
00002 // Shared image header file for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 1998-2026 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file.  If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 //     https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 //     https://www.fltk.org/bugs.php
00015 //
00016
00020 #ifndef Fl_Shared_Image_H
00021 #  define Fl_Shared_Image_H
00022
00023 #  include "Fl_Image.H"
00024
00025 #undef SHIM_DEBUG
00026
00071 typedef Fl_Image *(*Fl_Shared_Handler)(const char *name,
00072                                        uchar *header,
00073                                        int headerlen);
00074
00094 class FL_EXPORT Fl_Shared_Image : public Fl_Image {
00095
00096   friend class Fl_JPEG_Image;
00097   friend class Fl_PNG_Image;
00098   friend class Fl_SVG_Image;
00099   friend class Fl_Graphics_Driver;
00100
00101 protected:
00102
00103   static Fl_Shared_Image **images_;     // Shared images
00104   static int    num_images_;            // Number of shared images
00105   static int    alloc_images_;          // Allocated shared images
00106   static Fl_Shared_Handler *handlers_;  // Additional format handlers
00107   static int    num_handlers_;          // Number of format handlers
00108   static int    alloc_handlers_;        // Allocated format handlers
00109
00110   const char    *name_;                 // Name of image file
00111   int           original_;              // Original image?
00112   int           refcount_;              // Number of times this image has been used
00113   Fl_Image      *image_;                // The image that is shared
00114   int           alloc_image_;           // Was the image allocated?
00115
00116   static int    compare(Fl_Shared_Image **i0, Fl_Shared_Image **i1);
00117
00118   // Use get() and release() to load/delete images in memory...
00119   Fl_Shared_Image();
00120   Fl_Shared_Image(const char *n, Fl_Image *img = 0);
00121   virtual ~Fl_Shared_Image();
00122   void add();
00123   void update();
```

```
  Fl_Shared_Image *copy_(int W, int H) const;

public:

  class Deleter {
    public:
      void operator()(Fl_Shared_Image* img) const {
        if (img) img->release();
      }
  };

#ifdef SHIM_DEBUG
  static void print_pool();
#endif

  const char    *name() const { return name_; }

  int           refcount() const { return refcount_; }

  int original() const { return original_; }

  void  release() override;
  virtual void  reload();

  Fl_Shared_Image *as_shared_image() override {
    return this;
  }

  Fl_Image *copy(int W, int H) const override;
  Fl_Image *copy() const;
  Fl_Image *copy();

  void color_average(Fl_Color c, float i) override;
  void desaturate() override;
  void draw(int X, int Y, int W, int H, int cx = 0, int cy = 0) override;
  void draw(int X, int Y) { draw(X, Y, w(), h(), 0, 0); }
  void uncache() override;

  static Fl_Shared_Image *find(const char *name, int W = 0, int H = 0);
  static Fl_Shared_Image *get(const char *name, int W = 0, int H = 0);
  static Fl_Shared_Image *get(Fl_RGB_Image *rgb, int own_it = 1);
  static Fl_Shared_Image **images();
  static int            num_images();
  static void           add_handler(Fl_Shared_Handler f);
  static void           remove_handler(Fl_Shared_Handler f);

  const Fl_Image *image() const { return image_; }

}; // class Fl_Shared_Image

//
// The following function is provided in the fltk_images library and
// registers all of the "extra" image file formats that are not part
// of the core FLTK library...
//

FL_EXPORT extern void fl_register_images();

#endif // !Fl_Shared_Image_H
```

14.140 Fl_Shortcut_Button.H

```
//
// Shortcut Button header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2023 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef Fl_Shortcut_Button_H
#define Fl_Shortcut_Button_H

#include <FL/Fl_Button.H>

class FL_EXPORT Fl_Shortcut_Button : public Fl_Button {
private:
```

```
  bool hot_, pre_hot_, default_set_, handle_default_button_;
  Fl_Shortcut pre_esc_;
  Fl_Shortcut default_shortcut_;
protected:
  Fl_Shortcut shortcut_value;
  void do_end_hot_callback();
  int handle(int) override;
  void draw() override;
public:
  Fl_Shortcut_Button(int X,int Y,int W,int H, const char* l = 0);
  void value(Fl_Shortcut shortcut);
  Fl_Shortcut value();
#if 0
  // Default shortcut settings are disabled until successful review of the UI
  void default_value(Fl_Shortcut shortcut);
  Fl_Shortcut default_value();
  void default_clear();
#endif
};

#endif // Fl_Shortcut_Button_H

```

14.141 fl_show_colormap.H File Reference

The fl_show_colormap() function hides the implementation classes used to provide the popup window and color selection mechanism.

Functions

- Fl_Color fl_show_colormap (Fl_Color oldcol)

  Pops up a window to let the user pick a colormap entry.

14.141.1 Detailed Description

The fl_show_colormap() function hides the implementation classes used to provide the popup window and color selection mechanism.

14.142 fl_show_colormap.H

Go to the documentation of this file.

```
//
// Colormap picker header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef fl_show_colormap_H
#define fl_show_colormap_H

/* doxygen comment here to avoid exposing ColorMenu in fl_show_colormap.cxx
*/

FL_EXPORT Fl_Color fl_show_colormap(Fl_Color oldcol);

#endif
```

14.143 fl_show_input.H

```
//
// Standard input dialog header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
```

```
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#include "fl_ask.H"
```

14.144 Fl_Simple_Counter.H

```
//
// Simple counter header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2022 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Simple_Counter widget . */

#ifndef Fl_Simple_Counter_H
#define Fl_Simple_Counter_H

#include "Fl_Counter.H"
class FL_EXPORT Fl_Simple_Counter : public Fl_Counter {
public:
  Fl_Simple_Counter(int X,int Y,int W,int H, const char *L = 0);
};

#endif
```

14.145 Fl_Single_Window.H

```
//
// Single-buffered window header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2024 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Single_Window class . */

#ifndef Fl_Single_Window_H
#define Fl_Single_Window_H

#include "Fl_Window.H"

class FL_EXPORT Fl_Single_Window : public Fl_Window {
public:
  void show() override;
  void show(int argc, char **argv) { Fl_Window::show(argc, argv);}

  Fl_Single_Window(int W, int H, const char *l=0);

  Fl_Single_Window(int X, int Y, int W, int H, const char *l=0);

  void make_current() { Fl_Window::make_current(); }
```

```
  void flush() override { Fl_Window::flush(); }
};

#endif
```

14.146 Fl_Slider.H

```
//
// Slider header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Slider widget . */

#ifndef Fl_Slider_H
#define Fl_Slider_H

#ifndef Fl_Valuator_H
#include "Fl_Valuator.H"
#endif

#include <FL/Fl_Rect.H>

#define FL_VERT_SLIDER          0
#define FL_HOR_SLIDER           1
#define FL_VERT_FILL_SLIDER     2
#define FL_HOR_FILL_SLIDER      3
#define FL_VERT_NICE_SLIDER     4
#define FL_HOR_NICE_SLIDER      5

class FL_EXPORT Fl_Slider : public Fl_Valuator {

public:

  typedef enum : uchar {
    TICKS_NONE = 0x00,
    TICKS_BELOW = 0x01,
    TICKS_ABOVE = 0x02,
    TICKS_LEFT = 0x01,
    TICKS_RIGHT = 0x02
    // Later: LABEL : label the first, middle, and last tick with its value
    // Later: FRACTIONAL : make a half ticks a bit shorter, and quarter ticks even shorter
    // Later: DECIMAL: make .5 ticks shorter and 0.1 ticks even shorter
    // Later: POINTY_KNOB : make the slider knob pointy on the side where the ticks are
    // Later: SMALLE_KNOB : make the slider knob small, so the ticks are better visible
  } Tick_Position;

  typedef enum : uchar {
    LINEAR_SCALE = 0,
    LOG_SCALE
  } Scale_Type;

private:

  float slider_size_;
  uchar slider_;
  Scale_Type scale_type_ { LINEAR_SCALE };
  Tick_Position ticks_ { TICKS_NONE };
  uchar num_ticks_ { 11 };
  void _Fl_Slider();
  void draw_bg(int, int, int, int, int S=16);

protected:

  // these allow subclasses to put the slider in a smaller area:
  void draw(int, int, int, int);
  int handle(int, int, int, int, int);
  void draw() override;
  void draw_ticks(const Fl_Rect& r, int S);
  double value_to_position(double val) const;
  double position_to_value(double pos) const;
  double increment_lin_log(double v, int n, int range);
```

```

public:

  int handle(int) override;
  Fl_Slider(int X,int Y,int W,int H, const char *L = 0);
  Fl_Slider(uchar t,int X,int Y,int W,int H, const char *L);

  int scrollvalue(int pos,int size,int first,int total);
  void bounds(double a, double b);

  float slider_size() const {return slider_size_;}

  void slider_size(double v);

  Fl_Boxtype slider() const {return (Fl_Boxtype)slider_;}

  void slider(Fl_Boxtype c) {slider_ = (uchar)c;}

  Scale_Type scale() const { return scale_type_; }

  void scale(Scale_Type s) { scale_type_ = s; }

  uchar ticks() const { return ticks_; }

  uchar num_ticks() const { return num_ticks_; }

  void ticks(uchar t, uchar num_ticks=11) { ticks_ = (Tick_Position)t; num_ticks_ = num_ticks; }
};

#endif
```

14.147 Fl_Spinner.H

```
//
// Spinner widget for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2022 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Spinner widget . */

#ifndef Fl_Spinner_H
#define Fl_Spinner_H

#include <FL/Enumerations.H>
#include <FL/Fl_Group.H>
#include <FL/Fl_Input.H>
#include <FL/Fl_Repeat_Button.H>

class FL_EXPORT Fl_Spinner : public Fl_Group {

  double        value_;                 // Current value
  double        minimum_;               // Minimum value
  double        maximum_;               // Maximum value
  double        step_;                  // Amount to add/subtract for up/down
  const char    *format_;               // Format string for input field
  int           wrap_;                  // wrap around at bounds (1/0)

private:

  static void sb_cb(Fl_Widget *w, Fl_Spinner *sb); // internal callback
  void update();                                   // update input field

protected:

  // This class works like Fl_Input but ignores FL_Up and FL_Down key
  // presses so they are handled by its parent, the Fl_Spinner widget.
  // See STR #2989.

  class FL_EXPORT Fl_Spinner_Input : public Fl_Input {
  public:
    Fl_Spinner_Input(int X, int Y, int W, int H)
    : Fl_Input(X, Y, W, H) {}
    int handle(int event) override; // implemented in src/Fl_Spinner.cxx
```

```
  };

  Fl_Spinner_Input input_;              // Input field for the value
  Fl_Repeat_Button
                up_button_,             // Up button
                down_button_;           // Down button

  void draw() override;

public:

  // Constructor
  Fl_Spinner(int X, int Y, int W, int H, const char *L = 0);
  // Event handling
  int handle(int event) override;
  // Resize group and subwidgets
  void resize(int X, int Y, int W, int H) override;

  const char *format() const { return (format_); }

  void format(const char *f) { format_ = f; update(); }

  double maximum() const { return (maximum_); }

  void maximum(double m) { maximum_ = m; }

  double minimum() const { return (minimum_); }

  void minimum(double m) { minimum_ = m; }

  void range(double a, double b) { minimum_ = a; maximum_ = b; }

  // Sets the amount to change the value when the user clicks a button.
  // Docs in src/Fl_Spinner.cxx
  void step(double s);

  double step() const { return (step_); }

  void wrap(int set) { wrap_ = set ? 1 : 0; }

  int wrap() const { return wrap_; }

  Fl_Color textcolor() const { return (input_.textcolor()); }

  void textcolor(Fl_Color c) { input_.textcolor(c); }

  Fl_Font textfont() const { return (input_.textfont()); }

  void textfont(Fl_Font f) { input_.textfont(f); }

  Fl_Fontsize textsize() const { return (input_.textsize()); }

  void textsize(Fl_Fontsize s) { input_.textsize(s); }

  // Sets the numeric representation in the input field.
  // Docs see src/Fl_Spinner.cxx
  void type(uchar v);

  uchar type() const { return (input_.type()); }

  double value() const { return (value_); }

  void value(double v) { value_ = v; update(); }

  void color(Fl_Color v) { input_.color(v); }

  Fl_Color color() const { return(input_.color()); }

  void selection_color(Fl_Color val) { input_.selection_color(val); }

  Fl_Color selection_color() const { return input_.selection_color(); }

  void maximum_size(int m) { if (m > 0) input_.maximum_size(m); }

  int maximum_size() const { return input_.maximum_size(); }
};

#endif // !Fl_Spinner_H
```

14.148 fl_string_functions.h File Reference

Public header for FLTK's platform-agnostic string handling.

```
#include "Fl_Export.H"
#include <stddef.h>
```

Functions

- char * fl_strdup (const char *s)

  Cross platform interface to POSIX function strdup().
- size_t **fl_strlcpy** (char *, const char *, size_t)

14.148.1 Detailed Description

Public header for FLTK's platform-agnostic string handling.

14.149 fl_string_functions.h

Go to the documentation of this file.

```
00001 /*
00002  * Platform agnostic string portability functions for the Fast Light Tool Kit (FLTK).
00003  *
00004  * Copyright 2020-2022 by Bill Spitzak and others.
00005  *
00006  * This library is free software. Distribution and use rights are outlined in
00007  * the file "COPYING" which should have been included with this file.  If this
00008  * file is missing or damaged, see the license at:
00009  *
00010  *     https://www.fltk.org/COPYING.php
00011  *
00012  * Please see the following page on how to report bugs and issues:
00013  *
00014  *     https://www.fltk.org/bugs.php
00015  */
00016
00022 #ifndef _FL_fl_string_functions_h_
00023 #define _FL_fl_string_functions_h_
00024
00025 #include "Fl_Export.H"
00026
00027 #ifdef __cplusplus
00028 extern "C" {
00029 #endif
00030
00031 #include <stddef.h>  // size_t
00032
00038 FL_EXPORT char* fl_strdup(const char *s);
00039
00040 FL_EXPORT size_t fl_strlcpy(char *, const char *, size_t);
00041
00044 #ifdef __cplusplus
00045 }
00046 #endif /* __cplusplus */
00047
00048
00049 #endif /* _FL_fl_string_functions_h_ */
```

14.150 Fl_SVG_File_Surface.H

```
00001 //
00002 // Declaration of Fl_SVG_File_Surface in the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 2020 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file.  If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 //     https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 //     https://www.fltk.org/bugs.php
00015 //
00016
00017 #ifndef Fl_SVG_File_Surface_H
00018 #define Fl_SVG_File_Surface_H
```

```

#include <FL/Fl_Widget_Surface.H>
#include <stdio.h>

class FL_EXPORT Fl_SVG_File_Surface : public Fl_Widget_Surface {
  int width_, height_;
  int (*closef_)(FILE*);
public:
  Fl_SVG_File_Surface(int width, int height, FILE *svg, int (*closef)(FILE*) = NULL);
  ~Fl_SVG_File_Surface();
  FILE *file();
  void origin(int x, int y) override;
  void origin(int *x, int *y) override;
  void translate(int x, int y) override;
  void untranslate() override;
  int printable_rect(int *w, int *h) override;
  int close();
};

#endif /* Fl_SVG_File_Surface_H */
```

14.151 Fl_SVG_Image.H

```
//
// SVG Image header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 2017-2022 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef FL_SVG_IMAGE_H
#define FL_SVG_IMAGE_H

#include <FL/Fl_Image.H>

struct NSVGimage;

class FL_EXPORT Fl_SVG_Image : public Fl_RGB_Image {
private:
  typedef struct {
    NSVGimage* svg_image;
    int ref_count;
  } counted_NSVGimage;
  counted_NSVGimage* counted_svg_image_;
  bool rasterized_;
  int raster_w_, raster_h_;
  bool to_desaturate_;
  Fl_Color average_color_;
  float average_weight_;
  float svg_scaling_(int W, int H);
  void rasterize_(int W, int H);
  void cache_size_(int &width, int &height) override;
  void init_(const char *name, const unsigned char *filedata, size_t length);
  Fl_SVG_Image(const Fl_SVG_Image *source);
public:
  bool proportional;
  Fl_SVG_Image(const char *filename);
  Fl_SVG_Image(const char *sharedname, const char *svg_data);
  Fl_SVG_Image(const char *sharedname, const unsigned char *svg_data, size_t length);
  virtual ~Fl_SVG_Image();
  Fl_Image *copy(int W, int H) const override;
  Fl_Image *copy() const {
    return Fl_Image::copy();
  }
  void resize(int width, int height);
  void desaturate() override;
  void color_average(Fl_Color c, float i) override;
  void draw(int X, int Y, int W, int H, int cx = 0, int cy = 0) override;
  void draw(int X, int Y) { draw(X, Y, w(), h(), 0, 0); }
  Fl_SVG_Image *as_svg_image() override { return this; }
  const Fl_SVG_Image *as_svg_image() const override { return this; }
  void normalize() override;
  void scale(int w, int h, int keep_aspect = 1, int can_expand = 0) override;
};

#endif // FL_SVG_IMAGE_H
```

14.152 Fl_Sys_Menu_Bar.H File Reference

Definition of class Fl_Sys_Menu_Bar.
`#include <FL/Fl_Menu_Bar.H>`

Classes

- class Fl_Sys_Menu_Bar

  A class to create and modify menus that appear on macOS in the menu bar at the top of the screen.

Variables

- Fl_Sys_Menu_Bar * **fl_sys_menu_bar**

  The system menu bar.

14.152.1 Detailed Description

Definition of class Fl_Sys_Menu_Bar.

14.153 Fl_Sys_Menu_Bar.H

Go to the documentation of this file.

```
00001 //
00002 // MacOS system menu bar header file for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 1998-2017 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file.  If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 //     https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 //     https://www.fltk.org/bugs.php
00015 //
00016
00022 #ifndef Fl_Sys_Menu_Bar_H
00023 #define Fl_Sys_Menu_Bar_H
00024
00025 #include <FL/Fl_Menu_Bar.H>
00026
00027 class Fl_Sys_Menu_Bar_Driver;
00028
00095 class FL_EXPORT Fl_Sys_Menu_Bar : public Fl_Menu_Bar {
00096   static Fl_Sys_Menu_Bar_Driver *driver();
00097 protected:
00098   void draw() override;
00099 public:
00101   typedef enum {
00102     no_window_menu = 0,
00103     tabbing_mode_none,
00104     tabbing_mode_automatic,
00105     tabbing_mode_preferred
00106   } window_menu_style_enum;
00107   Fl_Sys_Menu_Bar(int x,int y,int w,int h,const char *l=0);
00108   virtual ~Fl_Sys_Menu_Bar();
00111   const Fl_Menu_Item *menu() const {return Fl_Menu_::menu();}
00112   void menu(const Fl_Menu_Item *m);
00113   void update() override;
00114   void play_menu(const Fl_Menu_Item *) override;
00115   int add(const char* label, int shortcut, Fl_Callback*, void *user_data=0, int flags=0);
00119   int add(const char* label, const char* shortcut, Fl_Callback* cb, void *user_data=0, int flags=0) {
00120     return add(label, fl_old_shortcut(shortcut), cb, user_data, flags);
00121     }
00122   int add(const char* str);
00123   int insert(int index, const char* label, int shortcut, Fl_Callback *cb, void *user_data=0, int
      flags=0);
00127   int insert(int index, const char* label, const char* shortcut, Fl_Callback *cb, void *user_data=0,
      int flags=0) {
00128     return insert(index, label, fl_old_shortcut(shortcut), cb, user_data, flags);
00129     }
00130   void remove(int n);
```

```
  void replace(int index, const char *name);
  void clear();
  int clear_submenu(int index);
  void  mode (int i, int fl);
  int mode(int i) const { return Fl_Menu_::mode(i); }
  bool is_global() const;
  void shortcut (int i, int s);
  void setonly (Fl_Menu_Item *item);
  static void about(Fl_Callback *cb, void *data);

  static window_menu_style_enum window_menu_style();
  static void window_menu_style(window_menu_style_enum style);
  static void create_window_menu();
};

extern Fl_Sys_Menu_Bar *fl_sys_menu_bar;

#endif // Fl_Sys_Menu_Bar_H
```

14.154 Fl_Table.H

```
//
// Fl_Table -- A table widget for the Fast Light Tool Kit (FLTK).
//
// Copyright 2002 by Greg Ercolano.
// Copyright (c) 2004 O'ksi'D
// Copyright 2009-2026 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef _FL_TABLE_H
#define _FL_TABLE_H

#include <FL/Fl_Group.H>
#include <FL/Fl_Scroll.H>

#include <vector>

class FL_EXPORT Fl_Table : public Fl_Group {
public:
  enum TableContext {
    CONTEXT_NONE       = 0,
    CONTEXT_STARTPAGE  = 0x01,
    CONTEXT_ENDPAGE    = 0x02,
    CONTEXT_ROW_HEADER = 0x04,
    CONTEXT_COL_HEADER = 0x08,
    CONTEXT_CELL       = 0x10,
    CONTEXT_TABLE      = 0x20,
    CONTEXT_RC_RESIZE  = 0x40
  };

private:
  int _rows, _cols;     // total rows/cols
  int _row_header_w;    // width of row header
  int _col_header_h;    // height of column header
  int _row_position;    // last row_position set (not necessarily == toprow!)
  int _col_position;    // last col_position set (not necessarily == leftcol!)

  char _row_header;     // row header enabled?
  char _col_header;     // col header enabled?
  char _row_resize;     // row resizing enabled?
  char _col_resize;     // col resizing enabled?
  int _row_resize_min;  // row minimum resizing height (default=1)
  int _col_resize_min;  // col minimum resizing width (default=1)

  // OPTIMIZATION: partial row/column redraw variables
  int _redraw_toprow;
  int _redraw_botrow;
  int _redraw_leftcol;
  int _redraw_rightcol;
  Fl_Color _row_header_color;
  Fl_Color _col_header_color;

  int _auto_drag;
  int _selecting;
  int _scrollbar_size;
```

```
  enum {
    TABCELLNAV = 1<<0
  };
  unsigned int flags_;

  std::vector<int> *_colwidths;         // column widths in pixels
  std::vector<int> *_rowheights;        // row heights in pixels

  // number of columns and rows == size of corresponding vectors
  int col_size() const;                 // size of the column widths vector
  int row_size() const;                 // size of the row heights vector

  Fl_Cursor _last_cursor;               // last mouse cursor before changed to 'resize' cursor

  // EVENT CALLBACK DATA
  TableContext _callback_context;       // event context
  int _callback_row, _callback_col;     // event row/col

  // handle() state variables.
  //    Put here instead of local statics in handle(), so more
  //    than one Fl_Table can exist without crosstalk between them.
  //
  int _resizing_col;                    // column being dragged
  int _resizing_row;                    // row being dragged
  int _dragging_x;                      // starting x position for horiz drag
  int _dragging_y;                      // starting y position for vert drag
  int _last_row;                        // last row we FL_PUSH'ed

  // Redraw single cell
  void _redraw_cell(TableContext context, int R, int C);

  void _start_auto_drag();
  void _stop_auto_drag();
  void _auto_drag_cb();
  static void _auto_drag_cb2(void *d);

protected:
  enum ResizeFlag {
    RESIZE_NONE      = 0,
    RESIZE_COL_LEFT  = 1,
    RESIZE_COL_RIGHT = 2,
    RESIZE_ROW_ABOVE = 3,
    RESIZE_ROW_BELOW = 4
  };

  int table_w;
  int table_h;
  int toprow;
  int botrow;
  int leftcol;
  int rightcol;

  // selection
  int current_row;
  int current_col;
  int select_row;
  int select_col;

  // OPTIMIZATION: Precomputed scroll positions for the toprow/leftcol
  int toprow_scrollpos;
  int leftcol_scrollpos;

  // Data table's inner dimension
  int tix;
  int tiy;
  int tiw;
  int tih;

  // Data table's outer dimension
  int tox;
  int toy;
  int tow;
  int toh;

  // Table widget's inner dimension
  int wix;
  int wiy;
  int wiw;
  int wih;

  Fl_Scroll *table;
  Fl_Scrollbar *vscrollbar;
  Fl_Scrollbar *hscrollbar;

  // Fltk
  int handle(int e) override;        // fltk handle() override

```

```
  // Class maintenance
  void recalc_dimensions();
  void table_resized();                          // table resized; recalc
  void table_scrolled();                         // table scrolled; recalc
  void get_bounds(TableContext context,          // return x/y/w/h bounds for context
                  int &X, int &Y, int &W, int &H);
  void change_cursor(Fl_Cursor newcursor);       // change mouse cursor to some other shape
  TableContext cursor2rowcol(int &R, int &C, ResizeFlag &resizeflag);
  int find_cell(TableContext context,            // find cell's x/y/w/h given r/c
                int R, int C, int &X, int &Y, int &W, int &H);
  int row_col_clamp(TableContext context, int &R, int &C);
  // clamp r/c to known universe

  virtual void draw_cell(TableContext context, int R=0, int C=0,
                         int X=0, int Y=0, int W=0, int H=0)
  { (void)context; (void)R; (void)C; (void)X; (void)Y; (void)W; (void)H;}
// overridden by deriving class

  long row_scroll_position(int row);             // find scroll position of row (in pixels)
  long col_scroll_position(int col);             // find scroll position of col (in pixels)

  int is_fltk_container() const {                // does table contain fltk widgets?
    return( Fl_Group::children() > 3 );          // (ie. more than box and 2 scrollbars?)
  }

  static void scroll_cb(Fl_Widget*,void*);       // h/v scrollbar callback

  void damage_zone(int r1, int c1, int r2, int c2, int r3 = 0, int c3 = 0);

  void redraw_range(int topRow, int botRow, int leftCol, int rightCol) {
    if ( _redraw_toprow == -1 ) {
      // Initialize redraw range
      _redraw_toprow = topRow;
      _redraw_botrow = botRow;
      _redraw_leftcol = leftCol;
      _redraw_rightcol = rightCol;
    } else {
      // Extend redraw range
      if ( topRow < _redraw_toprow ) _redraw_toprow = topRow;
      if ( botRow > _redraw_botrow ) _redraw_botrow = botRow;
      if ( leftCol < _redraw_leftcol ) _redraw_leftcol = leftCol;
      if ( rightCol > _redraw_rightcol ) _redraw_rightcol = rightCol;
    }
    // Indicate partial redraw needed of some cells
    damage(FL_DAMAGE_CHILD);
  }

  // draw() has to be protected per FLTK convention (was public in 1.3.x)
  void draw() override;

public:
  Fl_Table(int X, int Y, int W, int H, const char *l=0);
  ~Fl_Table();

  virtual void clear() {
    rows(0);
    cols(0);
    table->clear();
  }

  // \todo: add topline(), middleline(), bottomline()

  inline void table_box(Fl_Boxtype val) {
    table->box(val);
    table_resized();
  }

  inline Fl_Boxtype table_box( void ) const {
    return table->box();
  }

  virtual void rows(int val);                    // set number of rows

  inline int rows() const {
    return _rows;
  }

  virtual void cols(int val);                    // set number of columns

  inline int cols() const {
    return _cols;
  }

  inline void visible_cells(int& r1, int& r2, int& c1, int& c2) const {
    r1 = toprow;
    r2 = botrow;
    c1 = leftcol;
```

```
    c2 = rightcol;
  }

  int is_interactive_resize() const {
    return(_resizing_row != -1 || _resizing_col != -1);
  }

  inline int row_resize() const {
    return _row_resize;
  }

  void row_resize(int flag) {                    // enable row resizing
    _row_resize = (char)flag;
  }

  inline int col_resize() const {
    return _col_resize;
  }

  void col_resize(int flag) {                    // enable col resizing
    _col_resize = (char)flag;
  }

  inline int col_resize_min() const {            // column minimum resizing width
    return _col_resize_min;
  }

  void col_resize_min(int val) {
    _col_resize_min = ( val < 1 ) ? 1 : val;
  }

  inline int row_resize_min() const {           // column minimum resizing width
    return _row_resize_min;
  }

  void row_resize_min(int val) {
    _row_resize_min = ( val < 1 ) ? 1 : val;
  }

  inline int row_header() const {                // set/get row header enable flag
    return(_row_header);
  }

  void row_header(int flag) {
    _row_header = (char)flag;
    table_resized();
    redraw();
  }

  inline int col_header() const {                // set/get col header enable flag
    return _col_header;
  }

  void col_header(int flag) {
    _col_header = (char)flag;
    table_resized();
    redraw();
  }

  inline void col_header_height(int height) {    // set/get col header height
    _col_header_h = height;
    table_resized();
    redraw();
  }

  inline int col_header_height() const {
    return _col_header_h;
  }

  inline void row_header_width(int width) {      // set/get row header width
    _row_header_w = width;
    table_resized();
    redraw();
  }

  inline int row_header_width() const {
    return _row_header_w;
  }

  inline void row_header_color(Fl_Color val) {   // set/get row header color
    _row_header_color = val;
    redraw();
  }

  inline Fl_Color row_header_color() const {
    return _row_header_color;
  }
```

```
inline void col_header_color(Fl_Color val) {  // set/get col header color
  _col_header_color = val;
  redraw();
}

inline Fl_Color col_header_color() const {
  return _col_header_color;
}

void row_height(int row, int height);         // set row height in pixels

// Returns the current height of the specified row as a value in pixels.
int row_height(int row) const;

void col_width(int col, int width);           // set a column's width in pixels

// Returns the current width of the specified column in pixels.
int col_width(int col) const;

void row_height_all(int height) {             // set all row/col heights
  for ( int r=0; r<rows(); r++ ) {
    row_height(r, height);
  }
}

void col_width_all(int width) {
  for ( int c=0; c<cols(); c++ ) {
    col_width(c, width);
  }
}

void row_position(int row);                   // set/get table's current scroll position
void col_position(int col);

int row_position() const {
  return _row_position;
}

int col_position() const {
  return _col_position;
}

inline void top_row(int row) {                // set/get top row (deprecated)
  row_position(row);
}

inline int top_row() const {
  return row_position();
}
int is_selected(int r, int c) const;                // selected cell
void get_selection(int &row_top, int &col_left, int &row_bot, int &col_right) const;
void set_selection(int row_top, int col_left, int row_bot, int col_right);
int move_cursor(int R, int C, int shiftselect);
int move_cursor(int R, int C);
void resize(int X, int Y, int W, int H) override; // fltk resize() override

// This crashes sortapp() during init.
//  void box(Fl_Boxtype val) {
//    Fl_Group::box(val);
//    if ( table ) {
//      resize(x(), y(), w(), h());
//    }
//  }
//  Fl_Boxtype box(void) const {
//    return(Fl_Group::box());
//  }

// Child group management

void init_sizes() {
  table->init_sizes();
  table->redraw();
}

void add(Fl_Widget& wgt) {
  table->add(wgt);
  if ( table->children() > 2 ) {
    table->show();
  } else {
    table->hide();
  }
}

void add(Fl_Widget* wgt) {
  add(*wgt);
}
```

```
00777
00782   void insert(Fl_Widget& wgt, int n) {
00783     table->insert(wgt,n);
00784   }
00785
00791   void insert(Fl_Widget& wgt, Fl_Widget* w2) {
00792     table->insert(wgt,w2);
00793   }
00794
00798   void remove(Fl_Widget& wgt) {
00799     table->remove(wgt);
00800   }
00801
00802   // (doxygen will substitute Fl_Group's docs here)
00803   void begin() {
00804     table->begin();
00805   }
00806
00807   // (doxygen will substitute Fl_Group's docs here)
00808   void end() {
00809     table->end();
00810     // HACK: Avoid showing Fl_Scroll; seems to erase screen
00811     //       causing unnecessary flicker, even if its box() is FL_NO_BOX.
00812     //
00813     if ( table->children() > 2 ) {
00814       table->show();
00815     } else {
00816       table->hide();
00817     }
00818     Fl_Group::current(Fl_Group::parent());
00819   }
00820
00825   Fl_Widget*const* array() const {
00826     return table->array();
00827   }
00828
00843   Fl_Widget *child(int n) const {
00844     return table->child(n);
00845   }
00846
00855   int children() const {
00856     return table->children()-2;    // -2: skip Fl_Scroll's h/v scrollbar widgets
00857   }
00858
00859   // (doxygen will substitute Fl_Group's docs here)
00860   int find(const Fl_Widget *wgt) const {
00861     return table->find(wgt);
00862   }
00863
00864   // (doxygen will substitute Fl_Group's docs here)
00865   int find(const Fl_Widget &wgt) const {
00866     return table->find(wgt);
00867   }
00868
00869   // CALLBACKS
00870
00876   int callback_row() const {
00877     return _callback_row;
00878   }
00879
00885   int callback_col() const {
00886     return _callback_col;
00887   }
00888
00894   TableContext callback_context() const {
00895     return _callback_context;
00896   }
00897
00905   void do_callback(TableContext context, int row, int col) {
00906     _callback_context = context;
00907     _callback_row = row;
00908     _callback_col = col;
00909     Fl_Widget::do_callback();
00910   }
00911
00912 #ifdef FL_DOXYGEN
00941   void when(Fl_When flags);
00942 #endif
00943
00944 #ifdef FL_DOXYGEN
01022   void callback(Fl_Widget*, void*);
01023 #endif
01024
01034   int scrollbar_size() const {
01035       return _scrollbar_size;
01036   }
01037
```

```
  void scrollbar_size(int newSize) {
      if ( newSize != _scrollbar_size ) redraw();
      _scrollbar_size = newSize;
  }

  void tab_cell_nav(int val) {
    if ( val ) flags_ |=  TABCELLNAV;
    else       flags_ &= ~TABCELLNAV;
  }

  int tab_cell_nav() const {
    return (flags_ & TABCELLNAV ? 1 : 0);
  }
};

#endif /*_FL_TABLE_H*/
```

14.155 Fl_Table_Row.H

```
//
// Fl_Table_Row -- A row oriented table widget for the Fast Light Tool Kit (FLTK).
//
//    A class specializing in a table of rows.
//    Handles row-specific selection behavior.
//
// Copyright 2002 by Greg Ercolano.
// Copyright 2009-2025 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef _FL_TABLE_ROW_H
#define _FL_TABLE_ROW_H

#include <FL/Fl_Table.H>

#include <stdint.h>
#include <vector>

class FL_EXPORT Fl_Table_Row : public Fl_Table {
public:
  enum TableRowSelectMode {
    SELECT_NONE,                // no selection allowed
    SELECT_SINGLE,              // single row selection
    SELECT_MULTI                // multiple row selection (default)
  };
private:

  std::vector<uint8_t> _rowselect; // selection flag for each row

  // handle() state variables.
  //    Put here instead of local statics in handle(), so more
  //    than one instance can exist without crosstalk between.
  //
  int _dragging_select;         // dragging out a selection?
  int _last_row;
  int _last_y;                  // last event's Y position
  int _last_push_x;             // last PUSH event's X position
  int _last_push_y;             // last PUSH event's Y position

  TableRowSelectMode _selectmode;

protected:
  int handle(int event) override;
  int find_cell(TableContext context,           // find cell's x/y/w/h given r/c
                int R, int C, int &X, int &Y, int &W, int &H) {
    return(Fl_Table::find_cell(context, R, C, X, Y, W, H));
  }

public:
  Fl_Table_Row(int X, int Y, int W, int H, const char *l=0) : Fl_Table(X,Y,W,H,l) {
    _dragging_select = 0;
    _last_row        = -1;
    _last_y          = -1;
    _last_push_x     = -1;
    _last_push_y     = -1;
    _selectmode      = SELECT_MULTI;
```

```
  }

  ~Fl_Table_Row() { }

  void rows(int val) override; // set number of rows
  int rows() {                          // get number of rows
    return(Fl_Table::rows());
  }

  void type(TableRowSelectMode val);    // set selection mode

  TableRowSelectMode type() const {     // get selection mode
    return(_selectmode);
  }

  // Checks to see if 'row' is selected. Returns 1 if selected, 0 if not.
  int row_selected(int row);

  // Changes the selection state for 'row', depending on the value of 'flag'.
  int select_row(int row, int flag = 1);

  void select_all_rows(int flag=1);     // all rows to a known state

  void clear() override {
    rows(0);            // implies clearing selection
    cols(0);
    Fl_Table::clear();  // clear the table
  }
};

#endif /*_FL_TABLE_ROW_H*/
```

14.156 Fl_Tabs.H

```
//
// Tab header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2026 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Tabs widget . */

#ifndef Fl_Tabs_H
#define Fl_Tabs_H

#include "Fl_Group.H"

struct Fl_Menu_Item;

class FL_EXPORT Fl_Tabs : public Fl_Group {

  Fl_Widget *push_;

protected:

  int overflow_type;
  int tab_offset;
  int *tab_pos;
  int *tab_width;
  int *tab_flags;
  int tab_count;
  Fl_Align tab_align_;
  int has_overflow_menu;

  void take_focus(Fl_Widget *o);
  int maybe_do_callback(Fl_Widget *o);
  void check_overflow_menu();
  void handle_overflow_menu();
  void draw_overflow_menu_button();

  int on_insert(Fl_Widget*, int) override;
  int on_move(int, int) override;
  void on_remove(int) override;

```

```
  virtual void redraw_tabs();
  virtual int tab_positions();  // allocate and calculate tab positions
  virtual void clear_tab_positions();
  virtual void draw_tab(int x1, int x2, int W, int H, Fl_Widget* o, int flags, int sel);
  virtual int tab_height();
  virtual int hit_close(Fl_Widget *o, int event_x, int event_y);
  virtual int hit_overflow_menu(int event_x, int event_y);
  virtual int hit_tabs_area(int event_x, int event_y);

  void draw() override;

public:

  Fl_Tabs(int X, int Y, int W, int H, const char *L = 0);
  ~Fl_Tabs() override;

  void resize(int, int, int, int) override;
  void show() override;

  int handle(int) override;
  Fl_Widget *value();
  int value(Fl_Widget *);
  int value(int);

  Fl_Widget *push() const { return push_; }
  int push(Fl_Widget *);

  virtual Fl_Widget *which(int event_x, int event_y);
  void client_area(int &rx, int &ry, int &rw, int &rh, int tabh=0);

  void tab_align(Fl_Align a) { tab_align_ = a; }

  Fl_Align tab_align() const { return tab_align_; }

  enum {
    OVERFLOW_COMPRESS = 0,
    OVERFLOW_CLIP,
    OVERFLOW_PULLDOWN,
    OVERFLOW_DRAG
  };

  void handle_overflow(int ov);

};

#endif
```

14.157 Fl_Terminal.H File Reference

Fl_Terminal widget.

```
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Group.H>
#include <FL/Fl_Scrollbar.H>
#include <FL/Fl_Rect.H>
#include <FL/fl_utf8.h>
#include <stdarg.h>
#include <string>
```

Classes

- class Fl_Terminal::CharStyle
- class Fl_Terminal::Cursor
- class Fl_Terminal::EscapeSeq
- class Fl_Terminal

  Terminal widget supporting Unicode/utf-8, ANSI/xterm escape codes with full RGB color control.
- class Fl_Terminal::Margin
- class Fl_Terminal::PartialUtf8Buf
- class Fl_Terminal::RingBuffer
- class Fl_Terminal::Selection
- class Fl_Terminal::Utf8Char

14.157.1 Detailed Description

Fl_Terminal widget.

14.158 Fl_Terminal.H

Go to the documentation of this file.

```
00001 //
00002 // Fl_Terminal - A terminal widget for Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 2022 by Greg Ercolano.
00005 // Copyright 2024-2026 by Bill Spitzak and others.
00006 //
00007 // This library is free software. Distribution and use rights are outlined in
00008 // the file "COPYING" which should have been included with this file.  If this
00009 // file is missing or damaged, see the license at:
00010 //
00011 //     https://www.fltk.org/COPYING.php
00012 //
00013 // Please see the following page on how to report bugs and issues:
00014 //
00015 //     https://www.fltk.org/bugs.php
00016 //
00017
00022 #ifndef Fl_Terminal_H
00023 #define Fl_Terminal_H
00024
00025 #include <FL/Fl.H>
00026 #include <FL/Fl_Window.H>
00027 #include <FL/Fl_Group.H>
00028 #include <FL/Fl_Scrollbar.H>
00029 #include <FL/Fl_Rect.H>
00030 #include <FL/fl_utf8.h>         // fl_utf8_is_continuation()
00031
00032 #include <stdarg.h>             // va_list (MinGW)
00033 #include <string>
00034
00322 class FL_EXPORT Fl_Terminal : public Fl_Group {
00326 public:
00334   enum RedrawStyle {
00335     NO_REDRAW=0,
00336     RATE_LIMITED,
00337     PER_WRITE
00338   };
00339
00348   enum Attrib {
00349     NORMAL      = 0x00,
00350     BOLD        = 0x01,
00351     DIM         = 0x02,
00352     ITALIC      = 0x04,
00353     UNDERLINE   = 0x08,
00354     _RESERVED_1 = 0x10,
00355     INVERSE     = 0x20,
00356     _RESERVED_2 = 0x40,
00357     STRIKEOUT   = 0x80
00358   };
00359
00364   enum CharFlags {
00365     FG_XTERM   = 0x01,
00366     BG_XTERM   = 0x02,
00367     EOL        = 0x04,
00368     RESV_A     = 0x08,
00369     RESV_B     = 0x10,
00370     RESV_C     = 0x20,
00371     RESV_D     = 0x40,
00372     RESV_E     = 0x80,
00373     COLORMASK  = (FG_XTERM | BG_XTERM)
00374   };
00375
00380   enum OutFlags {
00381     OFF        = 0x00,
00382     CR_TO_LF   = 0x01,
00383     LF_TO_CR   = 0x02,
00384     LF_TO_CRLF = 0x04
00385   };
00386
00391   enum ScrollbarStyle {
00392     SCROLLBAR_OFF  = 0x00,
00393     SCROLLBAR_AUTO = 0x01,
00394     SCROLLBAR_ON   = 0x02
00395   };
00396
00402 protected:
00403   // Margin Class ////////////////////////////////////////////
```

```
//
// Class to manage the terminal's margins
//
class FL_EXPORT Margin {
  int left_, right_, top_, bottom_;
public:
  Margin(void) { left_ = right_ = top_ = bottom_ = 3; }
  int left(void)   const { return left_;   }
  int right(void)  const { return right_;  }
  int top(void)    const { return top_;    }
  int bottom(void) const { return bottom_; }
  void left(int val)     { left_   = val;  }
  void right(int val)    { right_  = val;  }
  void top(int val)      { top_    = val;  }
  void bottom(int val)   { bottom_ = val;  }
};

// CharStyle Class ////////////////////////////////////////////
//
// Class to manage the terminal's character style
// This includes the font, color, and some cached internal
// info for optimized drawing speed.
//
class FL_EXPORT CharStyle {
  uchar       attrib_;           // bold, underline..
  uchar       charflags_;        // CharFlags (xterm color management)
  Fl_Color    fgcolor_;          // foreground color for text
  Fl_Color    bgcolor_;          // background color for text
  Fl_Color    defaultfgcolor_;   // default fg color used by ESC[0m
  Fl_Color    defaultbgcolor_;   // default bg color used by ESC[0m
  Fl_Font     fontface_;         // font face
  Fl_Fontsize fontsize_;         // font size
  int         fontheight_;       // font height (in pixels)
  int         fontdescent_;      // font descent (pixels below font baseline)
  int         charwidth_;        // width of a fixed width ASCII character
public:
  CharStyle(bool fontsize_defer);
  uchar attrib(void) const            { return attrib_; }
  uchar charflags(void) const         { return charflags_; }
  // Colors - All access to colors are by Fl_Color only.
  //          There are three ways to SET colors: Fl_Color, rgb, xterm(uchar)
  //
  Fl_Color fltk_fg_color(uchar ci);
  Fl_Color fltk_bg_color(uchar ci);
  Fl_Color fgcolor(void) const;
  Fl_Color bgcolor(void) const;
  Fl_Color defaultfgcolor(void) const { return defaultfgcolor_; }
  Fl_Color defaultbgcolor(void) const { return defaultbgcolor_; }
  Fl_Font fontface(void) const        { return fontface_; }
  Fl_Fontsize fontsize(void) const    { return fontsize_; }
  int  fontheight(void) const         { return fontheight_; }
  int  fontdescent(void) const        { return fontdescent_; }
  int  charwidth(void) const          { return charwidth_; }
  uchar colorbits_only(uchar inflags) const;
  void attrib(uchar val)              { attrib_    = val; }
  void charflags(uchar val)           { charflags_ = val; }
  void set_charflag(uchar val)        { charflags_ |= val; }
  void clr_charflag(uchar val)        { charflags_ &= ~val; }
  // Non-xterm colors
  void fgcolor(int r,int g,int b)     { fgcolor_ = (r«24) | (g«16) | (b«8); clr_charflag(FG_XTERM); }
  void bgcolor(int r,int g,int b)     { bgcolor_ = (r«24) | (g«16) | (b«8); clr_charflag(BG_XTERM); }
  void fgcolor(Fl_Color val)          { fgcolor_ = val; clr_charflag(FG_XTERM); }
  void bgcolor(Fl_Color val)          { bgcolor_ = val; clr_charflag(BG_XTERM); }
  // Xterm colors
  void fgcolor_xterm(Fl_Color val)    { fgcolor_ = val; set_charflag(FG_XTERM); }
  void bgcolor_xterm(Fl_Color val)    { bgcolor_ = val; set_charflag(BG_XTERM); }
  void fgcolor_xterm(uchar val);
  void bgcolor_xterm(uchar val);
  //
  void defaultfgcolor(Fl_Color val)   { defaultfgcolor_ = val; }
  void defaultbgcolor(Fl_Color val)   { defaultbgcolor_ = val; }
  void fontface(Fl_Font val)          { fontface_ = val; update(); }
  void fontsize(Fl_Fontsize val)      { fontsize_ = val; update(); }
  void update(void);
  void update_fake(void);
  // SGR MODES: Set Graphics Rendition
  void sgr_reset(void) {                        // e.g. ESC[0m
    attrib(Fl_Terminal::NORMAL);
    if (charflags() & FG_XTERM) fgcolor_xterm(defaultfgcolor_);
    else                        fgcolor(defaultfgcolor_);
    if (charflags() & BG_XTERM) bgcolor_xterm(defaultbgcolor_);
```

```
      else                        bgcolor(defaultbgcolor_);
    }
    int  onoff(bool flag, Attrib a) const { return (flag ? (attrib_ | a) : (attrib_ & ~a)); }
    void sgr_bold(bool val)      { attrib_ = (uchar)onoff(val, Fl_Terminal::BOLD);      } // e.g. ESC[1m
    void sgr_dim(bool val)       { attrib_ = (uchar)onoff(val, Fl_Terminal::DIM);       } // e.g. ESC[2m
    void sgr_italic(bool val)    { attrib_ = (uchar)onoff(val, Fl_Terminal::ITALIC);    } // e.g. ESC[3m
    void sgr_underline(bool val) { attrib_ = (uchar)onoff(val, Fl_Terminal::UNDERLINE); } // e.g. ESC[4m
    void sgr_dbl_under(bool val) { attrib_ = (uchar)onoff(val, Fl_Terminal::UNDERLINE); } // e.g. ESC[21m (TODO!)
    void sgr_blink(bool val)     { (void)val; /* NOT IMPLEMENTED */                     } // e.g. ESC[5m
    void sgr_inverse(bool val)   { attrib_ = (uchar)onoff(val, Fl_Terminal::INVERSE);   } // e.g. ESC[7m
    void sgr_strike(bool val)    { attrib_ = (uchar)onoff(val, Fl_Terminal::STRIKEOUT); } // e.g. ESC[9m
  };

protected:
  // Cursor Class ////////////////////////////////////////////////////////
  //
  // Class to manage the terminal's cursor position, color, etc.
  //
  class FL_EXPORT Cursor {
    int      col_;      // cursor's current col (x) position on display
    int      row_;      // cursor's current row (y) position on display
    int      h_;        // cursor's height (affected by font size)
    Fl_Color fgcolor_;  // cursor's fg color (color of text, if any)
    Fl_Color bgcolor_;  // cursor's bg color
  public:
    Cursor(void) {
      col_     = 0;
      row_     = 0;
      h_       = 10;
      fgcolor_ = 0xfffff000;   // wht
      bgcolor_ = 0x00d00000;   // grn
    }
    int col(void) const          { return col_; }
    int row(void) const          { return row_; }
    int h(void) const            { return h_; }
    Fl_Color fgcolor(void) const { return fgcolor_; }
    Fl_Color bgcolor(void) const { return bgcolor_; }
    void col(int val)            { col_ = val >= 0 ? val : 0; }
    void row(int val)            { row_ = val >= 0 ? val : 0; }
    void h(int val)              { h_   = val; }
    void fgcolor(Fl_Color val)   { fgcolor_ = val; }
    void bgcolor(Fl_Color val)   { bgcolor_ = val; }
    int  left(void)              { col_ = (col_>0) ? (col_-1) : 0; return col_; }
    int  right(void)             { return ++col_; }
    int  up(void)                { row_ = (row_>0) ? (row_-1) : 0; return row_; }
    int  down(void)              { return ++row_; }
    bool is_rowcol(int drow,int dcol) const;
    void scroll(int nrows);
    void home(void) { row_ = 0; col_ = 0; }
  };

  // Utf8Char Class //////////////////////////////////////////////////////
  //
  //    Class to manage the terminal's individual UTF-8 characters.
  //    Includes fg/bg color, attributes (BOLD, UNDERLINE..)
  //
  class FL_EXPORT Utf8Char {
    static const int max_utf8_ = 4; // RFC 3629 paraphrased: In UTF-8, chars are encoded with 1 to 4 octets
    char     text_[max_utf8_];      // memory for actual ASCII or UTF-8 byte contents
    uchar    len_;                  // length of bytes in text_[] buffer; 1 for ASCII, >1 for UTF-8
    uchar    attrib_;               // attribute bits for this char (bold, underline..)
    uchar    charflags_;            // CharFlags (xterm colors management)
    Fl_Color fgcolor_;              // fltk fg color (supports 8color or 24bit color set w/ESC[38;2;<r>;<g>;<b>m)
    Fl_Color bgcolor_;              // fltk bg color (supports 8color or 24bit color set w/ESC[48;2;<r>;<g>;<b>m)
    // Private methods
    void text_utf8_(const char *text, int len);
    Fl_Color attr_color_(Fl_Color col, const Fl_Widget *grp) const;
  public:
    // Public methods
    Utf8Char(void);                             // ctor
    Utf8Char(const Utf8Char& o);                // copy ctor
    ~Utf8Char(void);                            // dtor
    Utf8Char& operator=(const Utf8Char& o);     // assignment
    inline int max_utf8() const { return max_utf8_; }
    void text_utf8(const char *text, int len, const CharStyle& style);
    void text_ascii(char c, const CharStyle& style);
```

```
    void fl_font_set(const CharStyle& style) const;

    // Return the UTF-8 text string for this character.
    //   Use length() to get number of bytes in string, which will be 1 for ASCII chars.
    //
    const char* text_utf8(void) const { return text_; }
    // Return the attribute for this char
    uchar attrib(void) const { return attrib_; }
    uchar charflags(void) const { return charflags_; }
    Fl_Color fgcolor(void) const;
    Fl_Color bgcolor(void) const;
    // Return the length of this character in bytes (UTF-8 can be multibyte..)
    int length(void) const { return int(len_); }
    double pwidth(void) const;
    int pwidth_int(void) const;
    // Clear the character to a 'space'
    void clear(const CharStyle& style) { text_utf8(" ", 1, style); charflags_ = 0; attrib_ = 0; }
    bool is_char(char c) const { return *text_ == c; }
    void show_char(void) const { ::printf("%.*s", len_, text_); }
    void show_char_info(void) const { ::fprintf(stderr, "UTF-8('%.*s', len=%d)\n", len_, text_, len_);
  }
    Fl_Color attr_fg_color(const Fl_Widget *grp) const;
    Fl_Color attr_bg_color(const Fl_Widget *grp) const;
  };

  // RingBuffer Class ////////////////////////////////////////////////////
  //
  // Manages ring with indexed row/col and "history" vs. "display" concepts.
  //
  class FL_EXPORT RingBuffer {
    Utf8Char *ring_chars_;    // the ring UTF-8 char buffer
    int ring_rows_;           // #rows in ring total
    int ring_cols_;           // #columns in ring/hist/disp
    int nchars_;              // #chars in ring (ring_rows*ring_cols)
    int hist_rows_;           // #rows in history
    int hist_use_;            // #rows in use by history
    int disp_rows_;           // #rows in display
    int offset_;              // index offset (used for 'scrolling')

private:
    void new_copy(int drows, int dcols, int hrows, const CharStyle& style);
    //DEBUG    void write_row(FILE *fp, Utf8Char *u8c, int cols) const {
    //DEBUG      cols = (cols != 0) ? cols : ring_cols();
    //DEBUG      for ( int col=0; col<cols; col++, u8c++ ) {
    //DEBUG        ::fprintf(fp, "%.*s", u8c->length(), u8c->text_utf8());
    //DEBUG      }
    //DEBUG    }
  public:
    void clear(void);
    void clear_hist(void);
    RingBuffer(void);
    RingBuffer(int drows, int dcols, int hrows);
    ~RingBuffer(void);

    // Methods to access ring
    //
    //    The 'offset' concept allows the 'history' and 'display'
    //    to be scrolled indefinitely. The 'offset' is applied
    //    to all row accesses, and is wrapped within the buffer.
    //
    //    For 'raw' access to the ring (without the offset concept),
    //    use the ring_chars() method, and walk from 0 - ring_rows()-1.
    //
    //          _____________
    //         |             | <- hist_srow()  <- ring_srow()
    //         |   H i s t   |
    //         |             |
    //         |_____________| <- hist_erow()
    //         |             | <- disp_srow()
    //         |   D i s p   |
    //         |             |
    //         |_____________| <- disp_erow()  <- ring_erow()
    //
    //          \___________/
    //           ring_cols()
    //           hist_cols()
    //           disp_cols()
    //
    inline int ring_rows(void) const { return ring_rows_; }
    inline int ring_cols(void) const { return ring_cols_; }
    inline int ring_srow(void) const { return(0); }
    inline int ring_erow(void) const { return(ring_rows_ - 1); }
    inline int hist_rows(void) const { return hist_rows_; }
    inline int hist_cols(void) const { return ring_cols_; }
    inline int hist_srow(void) const { return((offset_ + 0) % ring_rows_); }
    inline int hist_erow(void) const { return((offset_ + hist_rows_ - 1) % ring_rows_); }
    inline int disp_rows(void) const { return disp_rows_; }
```

```
    inline int disp_cols(void) const { return ring_cols_; }
    inline int disp_srow(void) const { return((offset_ + hist_rows_) % ring_rows_); }
    inline int disp_erow(void) const { return((offset_ + hist_rows_ + disp_rows_ - 1) % ring_rows_); }
    inline int offset(void)    const { return offset_; }
    void offset_adjust(int rows);
    void hist_rows(int val) { hist_rows_ = val; }
    void disp_rows(int val) { disp_rows_ = val; }

    // History use
    inline int  hist_use(void) const        { return hist_use_; }
    inline void hist_use(int val)           { hist_use_ = val; }
    inline int  hist_use_srow(void) const   { return((offset_ + hist_rows_ - hist_use_) % ring_rows_);
  }
    inline Utf8Char *ring_chars(void)       { return ring_chars_; }  // access ring buffer directly
    inline Utf8Char *ring_chars(void) const { return ring_chars_; }  // access ring buffer directly

    bool is_hist_ring_row(int grow) const;
    bool is_disp_ring_row(int grow) const;
    //DEBUG void show_ring_info(void) const;
    void move_disp_row(int src_row, int dst_row);
    void clear_disp_rows(int sdrow, int edrow, const CharStyle& style);
    void scroll(int rows, const CharStyle& style);

    const Utf8Char* u8c_ring_row(int row) const;
    const Utf8Char* u8c_hist_row(int hrow) const;
    const Utf8Char* u8c_hist_use_row(int hurow) const;
    const Utf8Char* u8c_disp_row(int drow) const;
    // Non-const versions of the above methods
    //   Using "Effective C++" ugly-as-hell syntax technique.
    //
    Utf8Char* u8c_ring_row(int row);
    Utf8Char* u8c_hist_row(int hrow);
    Utf8Char* u8c_hist_use_row(int hurow);
    Utf8Char* u8c_disp_row(int drow);

    void create(int drows, int dcols, int hrows);
    void resize(int drows, int dcols, int hrows, const CharStyle& style);

    void change_disp_rows(int drows, const CharStyle& style);
    void change_disp_cols(int dcols, const CharStyle& style);
  };

  // Selection Class ////////////////////////////////////////////////////
  //
  // Class to manage mouse selection
  //
  class FL_EXPORT Selection {
    Fl_Terminal *terminal_;
    int srow_, scol_, erow_, ecol_;     // selection start/end. NOTE: start *might* be > end
    int push_row_, push_col_;           // global row/col for last FL_PUSH
    bool push_char_right_;
    Fl_Color selectionbgcolor_;
    Fl_Color selectionfgcolor_;
    int state_ ;                        // 0=none, 1=started, 2=extended, 3=done
    bool is_selection_;                 // false: no selection
  public:
    Selection(Fl_Terminal *terminal);
    int srow(void) const { return srow_; }
    int scol(void) const { return scol_; }
    int erow(void) const { return erow_; }
    int ecol(void) const { return ecol_; }
    void push_clear() { push_row_ = push_col_ = -1; push_char_right_ = false; }
    void push_rowcol(int row,int col,bool char_right) {
      push_row_ = row; push_col_ = col; push_char_right_ = char_right; }
    void start_push() { start(push_row_, push_col_, push_char_right_); }
    bool dragged_off(int row,int col,bool char_right) const {
      return (push_row_ != row) || (push_col_+push_char_right_ != col+char_right); }
    void selectionfgcolor(Fl_Color val) { selectionfgcolor_ = val; }
    void selectionbgcolor(Fl_Color val) { selectionbgcolor_ = val; }
    Fl_Color selectionfgcolor(void) const { return selectionfgcolor_; }
    Fl_Color selectionbgcolor(void) const { return selectionbgcolor_; }
    bool is_selection(void) const { return is_selection_; }
    bool get_selection(int &srow,int &scol,int &erow,int &ecol) const;
    bool start(int row, int col, bool char_right);
    bool extend(int row, int col, bool char_right);
    void end(void);
    void select(int srow, int scol, int erow, int ecol);
    bool clear(void);
    int  state(void) const { return state_; }
    void scroll(int nrows);
  };

  // EscapeSeq Class ////////////////////////////////////////////////////
  //
  // Class to handle parsing ESC sequences
  //
  // Holds all state information for parsing esc sequences,
```

```
  // so sequences can span multiple block read(2) operations, etc.
  // Handling of parsed sequences is NOT handled in this class,
  // just the parsing of the sequences and managing generic integers.
  //
  class FL_EXPORT EscapeSeq {
  public:
    // EscapeSeq Constants
    // Maximums
    static const int maxbuff   = 80;  // character buffer
    static const int maxvals   = 20;  // integer value buffer
    // Return codes
    static const int success   = 0;   // operation succeeded
    static const int fail      = -1;  // operation failed
    static const int completed = 1;   // multi-step operation completed successfully
  private:
    char  esc_mode_;                  // escape parsing mode state
    char  csi_;                       // This is an ESC[.. sequence (Ctrl Seq Introducer)
    char  buff_[maxbuff];             // escape sequence being parsed
    char *buffp_;                     // parsing ptr into buff[]
    char *buffendp_;                  // end of buff[] (ptr to last valid buff char)
    char *valbuffp_;                  // pointer to first char in buff of integer being parsed
    int   vals_[maxvals];             // value array for parsing #'s in ESC[#;#;#..
    int   vali_;                      // parsing index into vals_[], 0 if none
    int   save_row_, save_col_;       // used by ESC[s/u for save/restore

    int append_buff(char c);
    int append_val(void);

  public:
    EscapeSeq(void);
    void reset(void);
    char esc_mode(void) const;
    void esc_mode(char val);
    int  total_vals(void) const;
    int  val(int i) const;
    int  defvalmax(int dval, int max) const;
    bool parse_in_progress(void) const;
    bool is_csi(void) const;
    int  parse(char c);
    void save_cursor(int row, int col);
    void restore_cursor(int &row, int &col);
  };

  // Partial UTF-8 Buffer Class ////////////////////////////////////////////
  //
  // Class to manage buffering partial UTF-8 characters between write calls.
  //
  class FL_EXPORT PartialUtf8Buf {
    char buf_[10];      // buffer partial UTF-8 encoded char
    int  buflen_;       // length of buffered UTF-8 encoded char
    int  clen_;         // final byte length of a UTF-8 char
  public:
    void clear(void) { buflen_ = clen_ = 0; }    // clear the buffer
    PartialUtf8Buf(void) { clear(); }            // Ctor
    // Is byte 'c' in the middle of a UTF-8 encoded byte sequence?
    bool is_continuation(char c) { return fl_utf8_is_continuation(c); }  // issue #1541
    // Access buffer
    const char* buf(void) const { return buf_; }
    // Access buffer length
    int buflen(void)      const { return buflen_; }
    // Append bytes of a partial UTF-8 string to the buffer.
    //
    // Returns:
    //   - true if done OK. Use is_complete() to see if a complete char received.
    //   - false if buffer overrun occurred, class is clear()ed.
    //
    // An appropriate response to 'false' would be to print the
    // "unknown character" and skip all subsequent UTF-8 continuation chars.
    //
    bool append(const char* p, int len) {
      if (len <= 0) return true;                      // ignore silly requests: say we did but dont
      if (buflen_ + len >= (int)sizeof(buf_))         // overrun check
        { clear(); return false; }                    // clear self, return false
      if (!buflen_) clen_ = fl_utf8len(*p);           // first byte? save char len for later
      while (len>0) { buf_[buflen_++] = *p++; len--; } // append byte to buffer
      return true;
    }
    bool is_complete(void) const { return (buflen_ && (buflen_ == clen_)); }
  };

public:
  Fl_Scrollbar  *scrollbar;         // vertical scrollbar (value: rows above the display)
  Fl_Scrollbar  *hscrollbar;        // horizontal scrollbar
private:
  // Special utf8 symbols
  std::string    error_char_;       // utf8 string shown for invalid utf8, bad ANSI, etc
  bool           fontsize_defer_;   // flag defers font calcs until first draw() (issue 837)
```

```
  int            scrollbar_size_;   // local preference for scrollbar size
  ScrollbarStyle hscrollbar_style_;
  CharStyle     *current_style_;    // current font, attrib, color..
  OutFlags       oflags_;           // output translation flags (CR_TO_LF, LF_TO_CR, LF_TO_CRLF)

  // A ring buffer is used for the terminal's history (hist) and display (disp) buffer.
  //    See README-Fl_Terminal.txt, section "RING BUFFER DESCRIPTION" for diagrams/info.
  //
  // Ring buffer
  RingBuffer     ring_;             // terminal history/display ring buffer
  Cursor         cursor_;           // terminal cursor (position, color, etc)
  Margin         margin_;           // terminal margins (top,left,bottom,right)
  Selection      select_;           // mouse selection
  EscapeSeq      escseq;            // Escape sequence parsing (ESC[ xterm/vt100)
  bool           show_unknown_;     // if true, show unknown chars as '¿' (default off)
  bool           ansi_;             // if true, parse ansi codes (default on)
  char          *tabstops_;         // array of tab stops (0|1)   \__ TODO: This should probably
  int            tabstops_size_;    // size of tabstops[] array   /         be a class "TabStops".
  Fl_Rect        scrn_;             // terminal screen xywh inside box(), margins, and scrollbar
  int            autoscroll_dir_;   // 0=autoscroll timer off, 3=scrolling up, 4=scrolling down
  int            autoscroll_amt_;   // #pixels above or below edge, used for autoscroll speed
  RedrawStyle    redraw_style_;     // NO_REDRAW, RATE_LIMITED, PER_WRITE
  float          redraw_rate_;      // maximum redraw rate in seconds, default=0.10 (10 per sec)
  bool           redraw_modified_;  // display modified; used by redraw_timer_cb() to rate limit redraws
  bool           redraw_timer_;     // if true, redraw timer is running
  PartialUtf8Buf pub_;              // handles Partial Utf8 Buffer (pub)

protected:
  // Ring buffer management
  const Utf8Char* u8c_ring_row(int grow) const;
  const Utf8Char* u8c_hist_row(int hrow) const;
  const Utf8Char* u8c_hist_use_row(int hrow) const;
  const Utf8Char* u8c_disp_row(int drow) const;
  // Non-const versions of the above.
  //     "Effective C++" says: implement non-const method to cast away const
  //
  Utf8Char* u8c_ring_row(int grow);
  Utf8Char* u8c_hist_row(int hrow);
  Utf8Char* u8c_hist_use_row(int hurow);
  Utf8Char* u8c_disp_row(int drow);
  Utf8Char* u8c_cursor(void);
private:
  void create_ring(int drows, int dcols, int hrows);
  void init_(int X,int Y,int W,int H,const char*L,int rows,int cols,int hist,bool fontsize_defer);
  // Tabstops
  void init_tabstops(int newsize);
  void default_tabstops(void);
  void clear_all_tabstops(void);
  void set_tabstop(void);
  void clear_tabstop(void);
  // Updates
  void update_screen_xywh(void);
  void update_screen(bool font_changed);
  void set_scrollbar_params(Fl_Scrollbar* scroll, int min, int max);
  void update_scrollbar(void);
  // Resize
  void resize_display_rows(int drows);
  void resize_display_columns(int dcols);
  void refit_disp_to_screen(void);
  // Callbacks
  static void scrollbar_cb(Fl_Widget*, void*);    // scrollbar manipulation
  static void autoscroll_timer_cb(void*);         // mouse drag autoscroll
  void        autoscroll_timer_cb2(void);
  static void redraw_timer_cb(void*);             // redraw rate limiting timer
  void        redraw_timer_cb2(void);

  // Screen management
protected:
  CharStyle& current_style(void) const;
  void current_style(const CharStyle& sty);
private:
  int  x_to_glob_col(int X, int grow, int &gcol, bool &gcr) const;
  int  xy_to_glob_rowcol(int X, int Y, int &grow, int &gcol, bool &gcr) const;
protected:
  int  w_to_col(int W) const;
  int  h_to_row(int H) const;
  // API: Display clear operations
  void clear_sod(void);
  void clear_eod(void);
  void clear_eol(void);
  void clear_sol(void);
  void clear_line(int row);
  void clear_line(void);
  const Utf8Char* walk_selection(const Utf8Char *u8c, int &row, int &col) const;
  bool get_selection(int &srow,int &scol,int &erow,int &ecol) const;
  bool is_selection(void) const;
```

```
  bool is_inside_selection(int row,int col) const;
private:
  bool is_hist_ring_row(int grow) const;
  bool is_disp_ring_row(int grow) const;
public:
  int  selection_text_len(void) const;
  const char* selection_text(void) const;
protected:
  void clear_mouse_selection(void);
  bool selection_extend(int X,int Y);
  void select_word(int grow, int gcol);
  void select_line(int grow);
  void scroll(int rows);
  void insert_rows(int count);
  void delete_rows(int count);
  void insert_char_eol(char c, int drow, int dcol, int rep);
  void insert_char(char c, int rep);
  void delete_chars(int drow, int dcol, int rep);
  void delete_chars(int rep);
public:
 // API: Terminal operations
  void clear(void);
  void clear(Fl_Color val);
  void clear_screen(bool scroll_to_hist=true);      // ESC [ 2 J
  void clear_screen_home(bool scroll_to_hist=true); // ESC [ H ESC [ 2 J
  void clear_history(void);                         // ESC [ 3 J
  void reset_terminal(void);                        // ESC c
  void cursor_home(void);                           // ESC [ 0 H
 // API: Cursor
  void cursorfgcolor(Fl_Color val);
  void cursorbgcolor(Fl_Color val);
  Fl_Color cursorfgcolor(void) const;
  Fl_Color cursorbgcolor(void) const;
protected:
  void cursor_row(int row);
  void cursor_col(int col);
public:
  int  cursor_row(void) const;
  int  cursor_col(void) const;
protected:
  void cursor_up(int count=1, bool do_scroll=false);
  void cursor_down(int count=1, bool do_scroll=false);
  void cursor_left(int count=1);
  void cursor_right(int count=1, bool do_scroll=false);
  void cursor_eol(void);
  void cursor_sol(void);
  void cursor_cr(void);
  void cursor_crlf(int count=1);
  void cursor_tab_right(int count=1);
  void cursor_tab_left(int count=1);
  void save_cursor(void);
  void restore_cursor(void);
  // Output translation
public:
  void output_translate(Fl_Terminal::OutFlags val);
  Fl_Terminal::OutFlags output_translate(void) const;
private:
  void handle_lf(void);
  void handle_cr(void);
  void handle_esc(void);
  // Printing
  void handle_ctrl(char c);
  bool is_printable(char c);
  bool is_ctrl(char c);
  void handle_SGR(void);
  void handle_DECRARA(void);
  void handle_escseq(char c);
  // --
  void display_modified(void);
  void display_modified_clear(void);
  void clear_char_at_disp(int drow, int dcol);
protected:
  const Utf8Char* utf8_char_at_disp(int drow, int dcol) const;
  const Utf8Char* utf8_char_at_glob(int grow, int gcol) const;
private:
  void repeat_char(char c, int rep);
  void utf8_cache_clear(void);
  void utf8_cache_flush(void);
  // API: Character display output
public:
  void plot_char(const char *text, int len, int drow, int dcol);
  void plot_char(char c, int drow, int dcol);
  void print_char(const char *text, int len=-1);
  void print_char(char c);
  // API: String display output
  void append_utf8(const char *buf, int len=-1);
  void append_ascii(const char *s);
```

```
  void append(const char *s, int len=-1);
protected:
  int handle_unknown_char(void);
  int handle_unknown_char(int drow, int dcol);
  // Drawing
  void draw_row_bg(int grow, int X, int Y) const;
  void draw_row(int grow, int Y) const;
  void draw_buff(int Y) const;
private:
  void handle_selection_autoscroll(void);
  int  handle_selection(int e);
public:
  // FLTK: draw(), resize(), handle()
  void draw(void) override;
  void resize(int X,int Y,int W,int H) override;
  int  handle(int e) override;
  const char* text(bool lines_below_cursor=false) const;

protected:
  // Internal short names
  //   Don't make these public, but allow internals and
  //   derived classes to maintain brevity.
  //
  inline int ring_rows(void) const { return ring_.ring_rows(); }
  inline int ring_cols(void) const { return ring_.ring_cols(); }
  inline int ring_srow(void) const { return ring_.ring_srow(); }
  inline int ring_erow(void) const { return ring_.ring_erow(); }
  inline int hist_rows(void) const { return ring_.hist_rows(); }
  inline int hist_cols(void) const { return ring_.hist_cols(); }
  inline int hist_srow(void) const { return ring_.hist_srow(); }
  inline int hist_erow(void) const { return ring_.hist_erow(); }
  inline int hist_use(void)  const { return ring_.hist_use();  }
  inline int hist_use_srow(void) const { return ring_.hist_use_srow(); }
  inline int disp_rows(void) const { return ring_.disp_rows(); }
  inline int disp_cols(void) const { return ring_.disp_cols(); }
  inline int disp_srow(void) const { return ring_.disp_srow(); }
  inline int disp_erow(void) const { return ring_.disp_erow(); }
  inline int offset(void) const    { return ring_.offset(); }

  // TODO: CLEAN UP WHAT'S PUBLIC, AND WHAT SHOULD BE 'PROTECTED' AND 'PRIVATE'
  //       Some of the public stuff should, quite simply, "not be".

  // API: Terminal features
public:
  // API: Scrollbar
  int   scrollbar_size(void) const;
  void  scrollbar_size(int val);
  int   scrollbar_actual_size(void) const;
  void  hscrollbar_style(ScrollbarStyle val);
  ScrollbarStyle hscrollbar_style(void) const;
  // API: History
  int   history_rows(void) const;
  void  history_rows(int val);
  int   history_use(void) const;
  // API: Display
  int   display_rows(void) const;
  void  display_rows(int val);
  int   display_columns(void) const;
  void  display_columns(int val);
  // API: Box
  void  box(Fl_Boxtype val) { Fl_Group::box(val); update_screen(false); }
  Fl_Boxtype box(void) const { return Fl_Group::box(); }
  // API: Margins
  int   margin_left(void)   const  { return margin_.left();   }
  int   margin_right(void)  const  { return margin_.right();  }
  int   margin_top(void)    const  { return margin_.top();    }
  int   margin_bottom(void) const  { return margin_.bottom(); }
  void  margin_left(int val);
  void  margin_right(int val);
  void  margin_top(int val);
  void  margin_bottom(int val);
  // API: Text font/size/color
  void  textfont(Fl_Font val);
  void  textsize(Fl_Fontsize val);
  void  textcolor(Fl_Color val);
  void  color(Fl_Color val);
  void  textfgcolor(Fl_Color val);
  void  textbgcolor(Fl_Color val);
  void  textfgcolor_default(Fl_Color val);
  void  textbgcolor_default(Fl_Color val);
  Fl_Font     textfont(void)      const { return current_style_->fontface(); }
  Fl_Fontsize textsize(void)      const { return current_style_->fontsize(); }
  Fl_Color    color(void)         const { return Fl_Group::color(); }
  Fl_Color    textcolor(void)     const { return textfgcolor_default(); }
  Fl_Color    textfgcolor(void)   const { return current_style_->fgcolor(); }
  Fl_Color    textbgcolor(void)   const { return current_style_->bgcolor(); }
  Fl_Color    textfgcolor_default(void)   const { return current_style_->defaultfgcolor(); }
```

```
  Fl_Color    textbgcolor_default(void)   const { return current_style_->defaultbgcolor(); }
  void        textfgcolor_xterm(uchar val);
  void        textbgcolor_xterm(uchar val);
  void selectionfgcolor(Fl_Color val) { select_.selectionfgcolor(val); }
  void selectionbgcolor(Fl_Color val) { select_.selectionbgcolor(val); }
  Fl_Color selectionfgcolor(void) const { return select_.selectionfgcolor(); }
  Fl_Color selectionbgcolor(void) const { return select_.selectionbgcolor(); }
  // API: Text attrib
  void textattrib(uchar val);
  uchar textattrib() const;
  // API: Redraw style/rate
  RedrawStyle redraw_style(void) const;
  void  redraw_style(RedrawStyle val);
private:
  bool  is_redraw_style(RedrawStyle val) const { return redraw_style_ == val; }
public:
  float redraw_rate(void) const;
  void  redraw_rate(float val);
  // API: Show unknown/invalid utf8/ANSI sequences with an error character (¿).
  bool  show_unknown(void) const;
  void  show_unknown(bool val);
  void error_char(const char* val) { error_char_ = val ? val : ""; }
  const char* error_char(void) const { return error_char_.c_str(); }
  // API: ANSI sequences
  bool  ansi(void) const;
  void  ansi(bool val);
  // Fl_Simple_Terminal API compatibility
  int   history_lines(void) const;
  void  history_lines(int val);
  // API: printf()
  void  printf(const char *fmt, ...);
  void  vprintf(const char *fmt, va_list ap);
  // Ctor
  Fl_Terminal(int X,int Y,int W,int H,const char*L=0);
  Fl_Terminal(int X,int Y,int W,int H,const char*L,int rows,int cols,int hist);
  // Dtor
  ~Fl_Terminal(void);
  // Debugging features
//DEBUG  void show_ring_info() const { ring_.show_ring_info(); }
//DEBUG  void write_row(FILE *fp, Utf8Char *u8c, int cols) const;
//DEBUG  void show_buffers(RingBuffer *a, RingBuffer *b=0) const;
};
#endif
```

14.159 Fl_Text_Buffer.H

```
//
// Header file for Fl_Text_Buffer class.
//
// Copyright 2001-2026 by Bill Spitzak and others.
// Original code Copyright Mark Edel.  Permission to distribute under
// the LGPL for the FLTK library granted by Mark Edel.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
 Fl_Text_Buffer, Fl_Text_Selection widget . */

#ifndef FL_TEXT_BUFFER_H
#define FL_TEXT_BUFFER_H

#include <stdarg.h>     /* va_list */
#include <string>
#include "fl_attr.h"    /* Doxygen can't find <FL/fl_attr.h> */

#undef ASSERT_UTF8

#ifdef ASSERT_UTF8
# include <assert.h>
# define IS_UTF8_ALIGNED(a) if (a && *a) assert(fl_utf8len(*(a))>0);
# define IS_UTF8_ALIGNED2(a, b) if (b>=0 && b<a->length()) assert(fl_utf8len(a->byte_at(b))>0);
#else
# define IS_UTF8_ALIGNED(a)
# define IS_UTF8_ALIGNED2(a, b)
#endif

```

```

/*
 "character size" is the size of a UTF-8 character in bytes
 "character width" is the width of a Unicode character in pixels
 "column" was originally defined as a character offset from the left margin.
 It was identical to the byte offset. In UTF-8, we have neither a byte offset
 nor truly fixed width fonts (*). Column could be a pixel value multiplied with
 an average character width (which is a bearable approximation).

 * in Unicode, there are no fixed width fonts! Even if the ASCII characters may
   happen to be all the same width in pixels, Chinese characters surely are not.
   There are plenty of exceptions, like ligatures, that make special handling of
   "fixed" character widths a nightmare. I decided to remove all references to
   fixed fonts and see "columns" as a multiple of the average width of a
   character in the main font.
     - Matthias
 */


/* Maximum length in characters of a tab or control character expansion
   of a single buffer character */
#define FL_TEXT_MAX_EXP_CHAR_LEN 20

#include "Fl_Export.H"

class Fl_Text_Undo_Action_List;
class Fl_Text_Undo_Action;

class FL_EXPORT Fl_Text_Selection {
  friend class Fl_Text_Buffer;

public:

  // Sets the selection range and selected().
  void set(int startpos, int endpos);

  // Updates a selection after text was modified.
  void update(int pos, int nDeleted, int nInserted);

  int start() const { return mSelected ? mStart : 0; }

  int end() const { return mSelected ? mEnd : 0; }

  bool selected() const { return mSelected; }

  void selected(bool b) { mSelected = b; }

  int length() const { return mSelected ? mEnd - mStart : 0; }

  // Returns true if position \p pos is in this Fl_Text_Selection.
  int includes(int pos) const;

  // Returns true if selected() and the positions of this selection.
  int selected(int *startpos, int *endpos) const;
  FL_DEPRECATED("since 1.4.0 - use selected(startpos, endpos) instead",
  int position(int *startpos, int *endpos) const) { return selected(startpos, endpos); }

protected:

  int mStart;
  int mEnd;
  bool mSelected;
};


typedef void (*Fl_Text_Modify_Cb)(int pos, int nInserted, int nDeleted,
                                  int nRestyled, const char* deletedText,
                                  void* cbArg);


typedef void (*Fl_Text_Predelete_Cb)(int pos, int nDeleted, void* cbArg);


class FL_EXPORT Fl_Text_Buffer {
public:

  Fl_Text_Buffer(int requestedSize = 0, int preferredGapSize = 1024);

  ~Fl_Text_Buffer();

  int length() const { return mLength; }

  char* text() const;

  std::string text_str() const;

  void text(const char* text);
```

```
char* text_range(int start, int end) const;

unsigned int char_at(int pos) const;

char byte_at(int pos) const;

const char *address(int pos) const
{ return (pos < mGapStart) ? mBuf+pos : mBuf+pos+mGapEnd-mGapStart; }

char *address(int pos)
{ return (pos < mGapStart) ? mBuf+pos : mBuf+pos+mGapEnd-mGapStart; }

void insert(int pos, const char* text, int insertedLength = -1);

void append(const char* t, int addedLength = -1) { insert(length(), t, addedLength); }

void vprintf(const char *fmt, va_list ap);
void printf(const char* fmt, ...);

void remove(int start, int end);

void replace(int start, int end, const char *text, int insertedLength = -1);

void copy(Fl_Text_Buffer* fromBuf, int fromStart, int fromEnd, int toPos);

int undo(int *cp=0);

bool can_undo() const;

int redo(int *cp=0);

bool can_redo() const;

void canUndo(char flag=1);

int insertfile(const char *file, int pos, int buflen = 128*1024);

int appendfile(const char *file, int buflen = 128*1024)
{ return insertfile(file, length(), buflen); }

int loadfile(const char *file, int buflen = 128*1024)
{ select(0, length()); remove_selection(); return appendfile(file, buflen); }

int outputfile(const char *file, int start, int end, int buflen = 128*1024);

int savefile(const char *file, int buflen = 128*1024)
{ return outputfile(file, 0, length(), buflen); }

int tab_distance() const { return mTabDist; }

void tab_distance(int tabDist);

void select(int start, int end);

int selected() const { return mPrimary.selected(); }

void unselect();

int selection_position(int* start, int* end) const;

char* selection_text() const;

void remove_selection();

void replace_selection(const char* text);

void secondary_select(int start, int end);

int secondary_selected() const { return mSecondary.selected(); }

void secondary_unselect();

int secondary_selection_position(int* start, int* end) const;

char* secondary_selection_text() const;

void remove_secondary_selection();

void replace_secondary_selection(const char* text);

void highlight(int start, int end);

int highlight() const { return mHighlight.selected(); }

void unhighlight();
```

```
  int highlight_position(int* start, int* end) const;

  char* highlight_text() const;

  void add_modify_callback(Fl_Text_Modify_Cb bufModifiedCB, void* cbArg);

  void remove_modify_callback(Fl_Text_Modify_Cb bufModifiedCB, void* cbArg);

  void call_modify_callbacks() { call_modify_callbacks(0, 0, 0, 0, 0); }

  void add_predelete_callback(Fl_Text_Predelete_Cb bufPredelCB, void* cbArg);

  void remove_predelete_callback(Fl_Text_Predelete_Cb predelCB, void* cbArg);

  void call_predelete_callbacks() { call_predelete_callbacks(0, 0); }

  char* line_text(int pos) const;

  int line_start(int pos) const;

  int line_end(int pos) const;

  int word_start(int pos) const;

  int word_end(int pos) const;

  int count_displayed_characters(int lineStartPos, int targetPos) const;

  int skip_displayed_characters(int lineStartPos, int nChars) const;

  int count_lines(int startPos, int endPos) const;

  int estimate_lines(int startPos, int endPos, int lineLen) const;

  int skip_lines(int startPos, int nLines) const;

  int rewind_lines(int startPos, int nLines) const;

  int findchar_forward(int startPos, unsigned searchChar, int* foundPos) const;

  int findchar_backward(int startPos, unsigned int searchChar, int* foundPos) const;

  int search_forward(int startPos, const char* searchString, int* foundPos,
                     int matchCase = 0) const;

  int search_backward(int startPos, const char* searchString, int* foundPos,
                      int matchCase = 0) const;

  const Fl_Text_Selection* primary_selection() const { return &mPrimary; }

  Fl_Text_Selection* primary_selection() { return &mPrimary; }

  const Fl_Text_Selection* secondary_selection() const { return &mSecondary; }

  const Fl_Text_Selection* highlight_selection() const { return &mHighlight; }

  int prev_char(int pos) const;
  int prev_char_clipped(int pos) const;

  int next_char(int pos) const;
  int next_char_clipped(int pos) const;

  int utf8_align(int) const;

  int input_file_was_transcoded;

  static const char* file_encoding_warning_message;

  void (*transcoding_warning_action)(Fl_Text_Buffer*);
  bool is_word_separator(int pos) const;

protected:

  void call_modify_callbacks(int pos, int nDeleted, int nInserted,
                             int nRestyled, const char* deletedText) const;

  void call_predelete_callbacks(int pos, int nDeleted) const;

  int insert_(int pos, const char* text, int insertedLength = -1);

  void remove_(int start, int end);

  void redisplay_selection(Fl_Text_Selection* oldSelection,
                           Fl_Text_Selection* newSelection) const;

  void move_gap(int pos);

```

```
  void reallocate_with_gap(int newGapStart, int newGapLen);

  char* selection_text_(const Fl_Text_Selection* sel) const;

  void remove_selection_(Fl_Text_Selection* sel);

  void replace_selection_(Fl_Text_Selection* sel, const char* text);

  void update_selections(int pos, int nDeleted, int nInserted);

  int apply_undo(Fl_Text_Undo_Action* action, int* cursorPos);

  Fl_Text_Selection mPrimary;
  Fl_Text_Selection mSecondary;
  Fl_Text_Selection mHighlight;
  int mLength;
  char* mBuf;
  int mGapStart;
  int mGapEnd;
  // The hardware tab distance used by all displays for this buffer,
  // and used in computing offsets for rectangular selection operations.
  int mTabDist;
  int mNModifyProcs;
  Fl_Text_Modify_Cb *mModifyProcs;
  void** mCbArgs;
  int mNPredeleteProcs;
  Fl_Text_Predelete_Cb *mPredeleteProcs;
  void **mPredeleteCbArgs;
  int mCursorPosHint;
  char mCanUndo;
  int mPreferredGapSize;
  Fl_Text_Undo_Action* mUndo;
  Fl_Text_Undo_Action_List* mUndoList;
  Fl_Text_Undo_Action_List* mRedoList;
};

#endif
```

14.160 Fl_Text_Display.H

```
//
// Header file for Fl_Text_Display class.
//
// Copyright 2001-2026 by Bill Spitzak and others.
// Original code Copyright Mark Edel.  Permission to distribute under
// the LGPL for the FLTK library granted by Mark Edel.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
 Fl_Text_Display widget . */

#ifndef FL_TEXT_DISPLAY_H
#define FL_TEXT_DISPLAY_H

#include <FL/Fl.H>              // Fl::scrollbar_size()
#include "fl_draw.H"
#include "Fl_Group.H"
#include "Fl_Widget.H"
#include "Fl_Scrollbar.H"
#include "Fl_Text_Buffer.H"

class FL_EXPORT Fl_Text_Display: public Fl_Group {

public:

  enum {
    NORMAL_CURSOR,
    CARET_CURSOR,
    DIM_CURSOR,
    BLOCK_CURSOR,
    HEAVY_CURSOR,
    SIMPLE_CURSOR
  };

  enum {
```

```
    CURSOR_POS,
    CHARACTER_POS
  };

  enum {
    DRAG_NONE = -2,
    DRAG_START_DND = -1,
    DRAG_CHAR = 0,
    DRAG_WORD = 1,
    DRAG_LINE = 2
  };

  enum {
    WRAP_NONE,
    WRAP_AT_COLUMN,
    WRAP_AT_PIXEL,
    WRAP_AT_BOUNDS
  };

  friend int fl_text_drag_prepare(int pos, int key, Fl_Text_Display* d);
  friend void fl_text_drag_me(int pos, Fl_Text_Display* d);

  typedef void (*Unfinished_Style_Cb)(int, void *);

  struct Style_Table_Entry {
    Fl_Color    color;
    Fl_Font     font;
    Fl_Fontsize size;
    unsigned    attr;
    Fl_Color    bgcolor;
  };

  enum {
    ATTR_BGCOLOR        = 0x0001,
    ATTR_BGCOLOR_EXT_   = 0x0002,
    ATTR_BGCOLOR_EXT    = 0x0003,
    ATTR_UNDERLINE      = 0x0004,
    ATTR_GRAMMAR        = 0x0008,
    ATTR_SPELLING       = 0x000C,
    ATTR_STRIKE_THROUGH = 0x0010,
    ATTR_LINES_MASK     = 0x001C
  };

  Fl_Text_Display(int X, int Y, int W, int H, const char *l = 0);
  ~Fl_Text_Display();

  int handle(int e) override;

  void buffer(Fl_Text_Buffer* buf);

  void buffer(Fl_Text_Buffer& buf) { buffer(&buf); }

  Fl_Text_Buffer* buffer() const { return mBuffer; }
  Fl_Text_Buffer* style_buffer() const { return mStyleBuffer; }

  void redisplay_range(int start, int end);
  void scroll(int topLineNum, int horizOffset);
  void insert(const char* text);
  void overstrike(const char* text);
  void insert_position(int newPos);

  int insert_position() const { return mCursorPos; }
  int position_to_xy(int pos, int* x, int* y) const;

  int in_selection(int x, int y) const;
  void show_insert_position();

  int move_right();
  int move_left();
  int move_up();
  int move_down();
  int count_lines(int start, int end, bool start_pos_is_line_start) const;
  int line_start(int pos) const;
  int line_end(int startPos, bool startPosIsLineStart) const;
  int skip_lines(int startPos, int nLines, bool startPosIsLineStart);
  int rewind_lines(int startPos, int nLines);
  void next_word(void);
  void previous_word(void);

  void show_cursor(int b = 1);

  void hide_cursor() { show_cursor(0); }

  void cursor_style(int style);
  int cursor_style() const { return mCursorStyle; }

  Fl_Color cursor_color() const {return mCursor_color;}
```

```
00248
00253   void cursor_color(Fl_Color n) {mCursor_color = n;}
00254
00260   FL_DEPRECATED("since 1.4.0 - use scrollbar_size() instead",
00261   int scrollbar_width() const ) {
00262       return scrollbar_width_ ? scrollbar_width_ : Fl::scrollbar_size();
00263   }
00264
00270   FL_DEPRECATED("since 1.4.0 - use scrollbar_size() instead",
00271   void scrollbar_width(int width) ) {
00272       Fl::scrollbar_size(width);
00273       scrollbar_width_ = 0;
00274   }
00275
00285   int scrollbar_size() const {
00286       return(scrollbar_width_);
00287   }
00288
00308   void scrollbar_size(int newSize) {
00309       scrollbar_width_ = newSize;
00310   }
00311
00316   Fl_Align scrollbar_align() const { return scrollbar_align_; }
00317
00322   void scrollbar_align(Fl_Align a) { scrollbar_align_ = a; }
00323
00329   int word_start(int pos) const { return buffer()->word_start(pos); }
00330
00336   int word_end(int pos) const { return buffer()->word_end(pos); }
00337
00338
00339   void highlight_data(Fl_Text_Buffer *styleBuffer,
00340                       const Style_Table_Entry *styleTable,
00341                       int nStyles, char unfinishedStyle,
00342                       Unfinished_Style_Cb unfinishedHighlightCB,
00343                       void *cbArg);
00344
00345   int position_style(int lineStartPos, int lineLen, int lineIndex) const;
00346
00352   int shortcut() const {return shortcut_;}
00353
00359   void shortcut(int s) {shortcut_ = s;}
00360
00365   Fl_Font textfont() const {return textfont_;}
00366
00371   void textfont(Fl_Font s) {textfont_ = s; mColumnScale = 0; }
00372
00377   Fl_Fontsize textsize() const {return textsize_;}
00378
00383   void textsize(Fl_Fontsize s) {textsize_ = s; mColumnScale = 0; }
00384
00389   Fl_Color textcolor() const {return textcolor_;}
00390
00395   void textcolor(Fl_Color n) {textcolor_ = n;}
00396
00401   void grammar_underline_color(Fl_Color color) { grammar_underline_color_ = color; }
00402
00407   Fl_Color grammar_underline_color() const { return grammar_underline_color_;}
00408
00413   void spelling_underline_color(Fl_Color color) { spelling_underline_color_ = color; }
00414
00419   Fl_Color spelling_underline_color() const { return spelling_underline_color_;}
00420
00425   void secondary_selection_color(Fl_Color color) { secondary_selection_color_ = color; }
00426
00431   Fl_Color secondary_selection_color() const { return secondary_selection_color_;}
00432
00433   int wrapped_column(int row, int column) const;
00434   int wrapped_row(int row) const;
00435   void wrap_mode(int wrap, int wrap_margin);
00436
00437   virtual void recalc_display();
00438   virtual void display_needs_recalc();
00439   void resize(int X, int Y, int W, int H) override;
00440
00446   double x_to_col(double x) const;
00447
00454   double col_to_x(double col) const;
00455
00456   void linenumber_width(int width);
00457   int linenumber_width() const;
00458   void linenumber_font(Fl_Font val);
00459   Fl_Font linenumber_font() const;
00460   void linenumber_size(Fl_Fontsize val);
00461   Fl_Fontsize linenumber_size() const;
00462   void linenumber_fgcolor(Fl_Color val);
00463   Fl_Color linenumber_fgcolor() const;
```

```
  void linenumber_bgcolor(Fl_Color val);
  Fl_Color linenumber_bgcolor() const;
  void linenumber_align(Fl_Align val);
  Fl_Align linenumber_align() const;
  void linenumber_format(const char* val);
  const char* linenumber_format() const;

protected:
  // Most (all?) of this stuff should only be called from resize() or
  // draw().
  // Anything with "vline" indicates thats it deals with currently
  // visible lines.

  void draw() override;
  void draw_text(int X, int Y, int W, int H);
  void draw_range(int start, int end);
  void draw_cursor(int, int);

  void draw_string(int style, int x, int y, int toX, const char *string,
                   int nChars) const;

  void draw_vline(int visLineNum, int leftClip, int rightClip,
                  int leftCharIndex, int rightCharIndex);

  int find_x(const char *s, int len, int style, int x) const;

  enum {
    DRAW_LINE,
    FIND_INDEX,
    FIND_INDEX_FROM_ZERO,
    GET_WIDTH,
    FIND_CURSOR_INDEX   // STR #2788
  };

  int handle_vline(int mode,
                   int lineStart, int lineLen, int leftChar, int rightChar,
                   int topClip, int bottomClip,
                   int leftClip, int rightClip) const;

  int handle_rmb(int readonly);

  void draw_line_numbers(bool clearAll);

  void clear_rect(int style, int x, int y, int width, int height) const;
  void display_insert();

  void offset_line_starts(int newTopLineNum);

  void calc_line_starts(int startLine, int endLine);

  void update_line_starts(int pos, int charsInserted, int charsDeleted,
                          int linesInserted, int linesDeleted, int *scrolled);

  void calc_last_char();

  int position_to_line( int pos, int* lineNum ) const;
  double string_width(const char* string, int length, int style) const;

  static void scroll_timer_cb(void*);

  static void buffer_predelete_cb(int pos, int nDeleted, void* cbArg);
  static void buffer_modified_cb(int pos, int nInserted, int nDeleted,
                                 int nRestyled, const char* deletedText,
                                 void* cbArg);

  static void h_scrollbar_cb(Fl_Scrollbar* w, Fl_Text_Display* d);
  static void v_scrollbar_cb( Fl_Scrollbar* w, Fl_Text_Display* d);
  void update_v_scrollbar();
  void update_h_scrollbar();
  int measure_vline(int visLineNum) const;
  int longest_vline() const;
  int empty_vlines() const;
  int vline_length(int visLineNum) const;
  int xy_to_position(int x, int y, int PosType = CHARACTER_POS) const;

  void xy_to_rowcol(int x, int y, int* row, int* column,
                    int PosType = CHARACTER_POS) const;
  void maintain_absolute_top_line_number(int state);
public:
  int get_absolute_top_line_number() const;
  int scroll_row() const { return mTopLineNum; }
  int scroll_col() const { return mHorizOffset; }
protected:
  void absolute_top_line_number(int oldFirstChar);
  int maintaining_absolute_top_line_number() const;
  void reset_absolute_top_line_number();
  int position_to_linecol(int pos, int* lineNum, int* column) const;
```

```
int scroll_(int topLineNum, int horizOffset);

void extend_range_for_styles(int* start, int* end);

void find_wrap_range(const char *deletedText, int pos, int nInserted,
                     int nDeleted, int *modRangeStart, int *modRangeEnd,
                     int *linesInserted, int *linesDeleted);
void measure_deleted_lines(int pos, int nDeleted);
void wrapped_line_counter(Fl_Text_Buffer *buf, int startPos, int maxPos,
                          int maxLines, bool startPosIsLineStart,
                          int styleBufOffset, int *retPos, int *retLines,
                          int *retLineStart, int *retLineEnd,
                          bool countLastLineMissingNewLine = true) const;
void find_line_end(int pos, bool start_pos_is_line_start, int *lineEnd,
                   int *nextLineStart) const;
double measure_proportional_character(const char *s, int colNum, int pos) const;
int wrap_uses_character(int lineEndPos) const;

int damage_range1_start, damage_range1_end;
int damage_range2_start, damage_range2_end;
int mCursorPos;
int mCursorOn;
int mCursorOldY;              /* Y pos. of cursor for blanking */
int mCursorToHint;            /* Tells the buffer modified callback
                               where to move the cursor, to reduce
                               the number of redraw calls */
int mCursorStyle;             /* One of enum cursorStyles above */
int mCursorPreferredXPos;     /* Pixel position for vert. cursor movement */
int mNVisibleLines;           /* # of visible (displayed) lines. This is
                                 also the size of the mLineStarts[] array. */
int mNBufferLines;            /* # of newlines in the buffer, or number of
                                 wraps if line wrapping is enabled. Note that
                                 partial lines at the end of the buffer are
                                 not counted, so you may want to add 1. */
Fl_Text_Buffer* mBuffer;      /* Contains text to be displayed */
Fl_Text_Buffer* mStyleBuffer; /* Optional parallel buffer containing
                               color and font information */
int mFirstChar, mLastChar;    /* Buffer positions of first and last
                               displayed character (lastChar points
                               either to a newline or one character
                               beyond the end of the buffer) */
int mContinuousWrap;          /* Wrap long lines when displaying */
int mWrapMarginPix;           /* Margin in # of pixels for
                               wrapping in continuousWrap mode */
int* mLineStarts;             /* Array of the size mNVisibleLines.
                                 This array only keeps track of lines
                                 within the display area. Each entry
                                 contains the starting character offset
                                 (from the beginning of the text buffer)
                                 for each /visible/ line.
                                 If wrap enabled, points to the beginning
                                 of each wrap. So a long line wrapping
                                 into 3 separate lines in the display
                                 will take up 3 separate array entries. */
int mTopLineNum;              /* Line number of top displayed line
                               of file (first line of file is 1) */
int mAbsTopLineNum;           /* In continuous wrap mode, the line
                                number of the top line if the text
                                were not wrapped (note that this is
                                only maintained as needed). */
int mNeedAbsTopLineNum;       /* Externally settable flag to continue
                               maintaining absTopLineNum even if
                               it isn't needed for line # display */
int mHorizOffset;             /* Horizontal scroll pos. in pixels */
int mTopLineNumHint;          /* Line number of top displayed line
                               of file (first line of file is 1) */
int mHorizOffsetHint;         /* Horizontal scroll pos. in pixels */
int mNStyles;                 /* Number of entries in styleTable */
const Style_Table_Entry *mStyleTable; /* Table of fonts and colors for
                                         coloring/syntax-highlighting */
char mUnfinishedStyle;        /* Style buffer entry which triggers
                               on-the-fly reparsing of region */
Unfinished_Style_Cb mUnfinishedHighlightCB; /* Callback to parse "unfinished" */
/* regions */
void* mHighlightCBArg;        /* Arg to unfinishedHighlightCB */

int mMaxsize;

int mSuppressResync;          /* Suppress resynchronization of line
                               starts during buffer updates */
int mNLinesDeleted;           /* Number of lines deleted during
                               buffer modification (only used
                               when resynchronization is suppressed) */
int mModifyingTabDistance;    /* Whether tab distance is being modified XXX: UNUSED */

mutable double mColumnScale; /* Width in pixels of an average character. This
                               value is calculated as needed (lazy eval); it
```

```
                                needs to be mutable so that it can be calculated
                                within a method marked as "const" */

  bool display_needs_recalc_;  /* Set to true when the display needs
                                to be recalculated. */

  Fl_Color mCursor_color;

  Fl_Scrollbar* mHScrollBar;
  Fl_Scrollbar* mVScrollBar;
  int scrollbar_width_;         // size of scrollbar trough (behavior changed in 1.4)
  Fl_Align scrollbar_align_;
  int dragPos, dragType, dragging;
  int display_insert_position_hint;
  struct { int x, y, w, h; } text_area;

  int shortcut_;

  Fl_Font textfont_;
  Fl_Fontsize textsize_;
  Fl_Color textcolor_;
  Fl_Color grammar_underline_color_;
  Fl_Color spelling_underline_color_;
  Fl_Color secondary_selection_color_;

  // Line number margin and width
  int mLineNumLeft, mLineNumWidth;

  // Line number font/colors
  Fl_Font     linenumber_font_;
  Fl_Fontsize linenumber_size_;
  Fl_Color    linenumber_fgcolor_;
  Fl_Color    linenumber_bgcolor_;
  Fl_Align    linenumber_align_;
  const char* linenumber_format_;
};

#endif
```

14.161 Fl_Text_Editor.H

```
//
// Header file for Fl_Text_Editor class.
//
// Copyright 2001-2026 by Bill Spitzak and others.
// Original code Copyright Mark Edel.  Permission to distribute under
// the LGPL for the FLTK library granted by Mark Edel.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Text_Editor widget . */


#ifndef FL_TEXT_EDITOR_H
#define FL_TEXT_EDITOR_H

#include "Fl_Text_Display.H"

// key will match in any state
#define FL_TEXT_EDITOR_ANY_STATE  (-1L)

class FL_EXPORT Fl_Text_Editor : public Fl_Text_Display {
  public:
    typedef int (*Key_Func)(int key, Fl_Text_Editor* editor);

    struct Key_Binding {
      int          key;
      int          state;
      Key_Func     function;
      Key_Binding* next;
    };

    Fl_Text_Editor(int X, int Y, int W, int H, const char* l = 0);
    ~Fl_Text_Editor() { remove_all_key_bindings(); }
    int handle(int e) override;
```

```
    void insert_mode(int b) { insert_mode_ = b; }
    int insert_mode() const { return insert_mode_; }
    void tab_nav(int val);
    int tab_nav() const;
    void add_key_binding(int key, int state, Key_Func f, Key_Binding** list);
    void add_key_binding(int key, int state, Key_Func f)
      { add_key_binding(key, state, f, &key_bindings); }
    void remove_key_binding(int key, int state, Key_Binding** list);
    void remove_key_binding(int key, int state)
      { remove_key_binding(key, state, &key_bindings); }
    void remove_all_key_bindings(Key_Binding** list);
    void remove_all_key_bindings() { remove_all_key_bindings(&key_bindings); }
    void add_default_key_bindings(Key_Binding** list);
    Key_Func bound_key_function(int key, int state, Key_Binding* list) const;
    Key_Func bound_key_function(int key, int state) const
      { return bound_key_function(key, state, key_bindings); }
    void default_key_function(Key_Func f) { default_key_function_ = f; }

    // functions for the built in default bindings
    static int kf_default(int c, Fl_Text_Editor* e);
    static int kf_ignore(int c, Fl_Text_Editor* e);
    static int kf_backspace(int c, Fl_Text_Editor* e);
    static int kf_enter(int c, Fl_Text_Editor* e);
    static int kf_move(int c, Fl_Text_Editor* e);
    static int kf_shift_move(int c, Fl_Text_Editor* e);
    static int kf_ctrl_move(int c, Fl_Text_Editor* e);
    static int kf_c_s_move(int c, Fl_Text_Editor* e);
    static int kf_meta_move(int c, Fl_Text_Editor* e);
    static int kf_m_s_move(int c, Fl_Text_Editor* e);
    static int kf_home(int, Fl_Text_Editor* e);
    static int kf_end(int c, Fl_Text_Editor* e);
    static int kf_left(int c, Fl_Text_Editor* e);
    static int kf_up(int c, Fl_Text_Editor* e);
    static int kf_right(int c, Fl_Text_Editor* e);
    static int kf_down(int c, Fl_Text_Editor* e);
    static int kf_page_up(int c, Fl_Text_Editor* e);
    static int kf_page_down(int c, Fl_Text_Editor* e);
    static int kf_insert(int c, Fl_Text_Editor* e);
    static int kf_delete(int c, Fl_Text_Editor* e);
    static int kf_copy(int c, Fl_Text_Editor* e);
    static int kf_cut(int c, Fl_Text_Editor* e);
    static int kf_paste(int c, Fl_Text_Editor* e);
    static int kf_select_all(int c, Fl_Text_Editor* e);
    static int kf_undo(int c, Fl_Text_Editor* e);
    static int kf_redo(int c, Fl_Text_Editor* e);

  protected:
    int handle_key();
    void maybe_do_callback(Fl_Callback_Reason reason = FL_REASON_CHANGED);

#ifndef FL_DOXYGEN
    int insert_mode_;
    Key_Binding* key_bindings;
#endif

    static Key_Binding* global_key_bindings;

#ifndef FL_DOXYGEN
    Key_Func default_key_function_;
#endif
};

#endif
```

14.162 Fl_Tile.H

```
//
// Tile header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2026 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file. If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef Fl_Tile_H
#define Fl_Tile_H
```

```

#include "Fl_Group.H"

/*
  The Fl_Tile class lets you resize its children by dragging
  the border between them.
*/

class FL_EXPORT Fl_Tile : public Fl_Group {
public:
  int handle(int event) override;
  Fl_Tile(int X, int Y, int W, int H, const char *L=0);
  ~Fl_Tile() override;
  void resize(int X, int Y, int W, int H) override;
  virtual void move_intersection(int oldx, int oldy, int newx, int newy);
  virtual void drag_intersection(int oldx, int oldy, int newx, int newy);
  FL_DEPRECATED("since 1.4.0 - use move_intersection(p) instead",
  void position(int oldx, int oldy, int newx, int newy)) { move_intersection(oldx, oldy, newx, newy);
    }
  void position(int x, int y) { Fl_Group::position(x, y); }
  void size_range(int index, int minw, int minh, int maxw=0x7FFFFFFF, int maxh=0x7FFFFFFF);
  void size_range(Fl_Widget *w , int minw, int minh, int maxw=0x7FFFFFFF, int maxh=0x7FFFFFFF);
  void init_size_range(int default_min_w = -1, int default_min_h = -1);

protected:
  int cursor_;
  Fl_Cursor *cursors_;

  Fl_Cursor cursor(int n) const {
    return cursors_[n];
  }

  void set_cursor(int n);       // set one of n (0..3) cursors

  typedef struct { int minw, minh, maxw, maxh; } Size_Range;

  Size_Range *size_range_;
  int size_range_size_, size_range_capacity_;
  int default_min_w_, default_min_h_;
  void request_shrink_l(int old_l, int &new_l, Fl_Rect *final_size);
  void request_shrink_r(int old_r, int &new_r, Fl_Rect *final_size);
  void request_shrink_t(int old_t, int &new_t, Fl_Rect *final_size);
  void request_shrink_b(int old_b, int &new_b, Fl_Rect *final_size);
  void request_grow_l(int old_l, int &new_l, Fl_Rect *final_size);
  void request_grow_r(int old_r, int &new_r, Fl_Rect *final_size);
  void request_grow_t(int old_t, int &new_t, Fl_Rect *final_size);
  void request_grow_b(int old_b, int &new_b, Fl_Rect *final_size);

  int on_insert(Fl_Widget*, int) override;
  int on_move(int, int) override;
  void on_remove(int) override;
};

#endif
```

14.163 Fl_Tiled_Image.H

```
//
// Tiled image header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2026 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Tiled_Image widget . */

#ifndef Fl_Tiled_Image_H
#  define Fl_Tiled_Image_H

#  include "Fl_Image.H"


class FL_EXPORT Fl_Tiled_Image : public Fl_Image {
protected:

```

```
  Fl_Image      *image_;                // The image that is tiled
  int           alloc_image_;           // Did we allocate this image?

public:
  Fl_Tiled_Image(Fl_Image *i, int W = 0, int H = 0);
  virtual ~Fl_Tiled_Image();

  Fl_Image *copy(int W, int H) const override;
  Fl_Image *copy() const {
    return Fl_Image::copy();
  }
  void color_average(Fl_Color c, float i) override;
  void desaturate() override;
  void draw(int X, int Y, int W, int H, int cx = 0, int cy = 0) override;
  void draw(int X, int Y) { draw(X, Y, w(), h(), 0, 0); }
  Fl_Image *image() const { return image_; }
};

#endif // !Fl_Tiled_Image_H
```

14.164 Fl_Timer.H

```
//
// Timer header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Timer widget . */

#ifndef Fl_Timer_H
#define Fl_Timer_H

#ifndef Fl_Widget_H
#include "Fl_Widget.H"
#endif

// values for type():
#define FL_NORMAL_TIMER         0
#define FL_VALUE_TIMER          1
#define FL_HIDDEN_TIMER         2

class FL_EXPORT Fl_Timer : public Fl_Widget {
  static void stepcb(void *);
  void step();
  char on, direction_;
  double delay, total;
  long lastsec,lastusec;
protected:
  void draw() override;
public:
  int handle(int) override;
  Fl_Timer(uchar t,int x,int y,int w,int h, const char *l);
  ~Fl_Timer();
  void value(double);
  double value() const {return delay>0.0?delay:0.0;}
  char direction() const {return direction_;}
  void direction(char d) {direction_ = d;}
  char suspended() const {return !on;}
  void suspended(char d);
};

#endif

```

14.165 Fl_Toggle_Button.H

```
//
// Toggle button header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
```

```
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Toggle_Button widget . */

#ifndef Fl_Toggle_Button_H
#define Fl_Toggle_Button_H

#include "Fl_Button.H"

class FL_EXPORT Fl_Toggle_Button : public Fl_Button {
public:
  Fl_Toggle_Button(int X,int Y,int W,int H,const char *l=0);
};

#endif
```

14.166 Fl_Toggle_Light_Button.H

```
//
// Toggle light button header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

// provided for back-compatibility only

#ifndef Fl_Toggle_Light_Button
#include "Fl_Light_Button.H"
#define Fl_Toggle_Light_Button Fl_Light_Button
#endif
```

14.167 Fl_Toggle_Round_Button.H

```
//
// Toggle round button header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

// provided for back-compatibility only

#ifndef Fl_Toggle_Round_Button
#include "Fl_Round_Button.H"
#define Fl_Toggle_Round_Button Fl_Round_Button
#endif
```

14.168 Fl_Tooltip.H

```
//
// Tooltip header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2011 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Tooltip widget . */

#ifndef Fl_Tooltip_H
#define Fl_Tooltip_H

#include <FL/Fl.H>
#include <FL/Fl_Widget.H>

class FL_EXPORT Fl_Tooltip {
  friend class Fl_TooltipBox;
public:
  static float delay() { return delay_; }
  static void delay(float f) { delay_ = f; }
  static float hidedelay() { return hidedelay_; }
  static void hidedelay(float f) { hidedelay_ = f; }
  static float hoverdelay() { return hoverdelay_; }
  static void hoverdelay(float f) { hoverdelay_ = f; }
  static int enabled() { return Fl::option(Fl::OPTION_SHOW_TOOLTIPS); }
  static void enable(int b = 1) { Fl::option(Fl::OPTION_SHOW_TOOLTIPS, (b!=0));}
  static void disable() { enable(0); }
  static void (*enter)(Fl_Widget* w);
  static void enter_area(Fl_Widget* w, int X, int Y, int W, int H, const char* tip);
  static void (*exit)(Fl_Widget *w);
  static Fl_Widget* current() {return widget_;}
  static void current(Fl_Widget*);

  static Fl_Font font() { return font_; }
  static void font(Fl_Font i) { font_ = i; }
  static Fl_Fontsize size() { return (size_ == -1 ? FL_NORMAL_SIZE : size_); }
  static void size(Fl_Fontsize s) { size_ = s; }
  static Fl_Color color() { return color_; }
  static void color(Fl_Color c) { color_ = c; }
  static Fl_Color textcolor() { return textcolor_; }
  static void textcolor(Fl_Color c) { textcolor_ = c; }
  static int margin_width() { return margin_width_; }
  static void margin_width(int v) { margin_width_ = v; }
  static int margin_height() { return margin_height_; }
  static void margin_height(int v) { margin_height_ = v; }
  static int wrap_width() { return wrap_width_; }
  static void wrap_width(int v) { wrap_width_ = v; }
  static Fl_Window* current_window(void);
  static int override_text(const char *new_text);

  // These should not be public, but Fl_Widget::tooltip() needs them...
  // fabien: made it private with only a friend function access
private:
  friend void Fl_Widget::tooltip(const char *);
  friend void Fl_Widget::copy_tooltip(const char *);
  static void enter_(Fl_Widget* w);
  static void exit_(Fl_Widget *w);
  static void set_enter_exit_once_();
  static void tooltip_timeout_(void*);

private:
  static float delay_;
  static float hidedelay_;
  static float hoverdelay_;
  static Fl_Color color_;
  static Fl_Color textcolor_;
  static Fl_Font font_;
  static Fl_Fontsize size_;
  static Fl_Widget* widget_;
  static int margin_width_;
  static int margin_height_;
  static int wrap_width_;
  static const int draw_symbols_; // 1 = draw @-symbols in tooltips, 0 = no
  static char *override_text_;
};
```

```
00126
00127 #endif
```

14.169 Fl_Tree.H File Reference

This file contains the definitions of the Fl_Tree class.

```
#include <FL/Fl.H>
#include <FL/Fl_Group.H>
#include <FL/Fl_Scrollbar.H>
#include <FL/fl_draw.H>
#include <FL/Fl_Tree_Item.H>
#include <FL/Fl_Tree_Prefs.H>
```

Classes

- class Fl_Tree

  Tree widget.

Enumerations

- enum Fl_Tree_Reason {
  FL_TREE_REASON_NONE = FL_REASON_UNKNOWN , FL_TREE_REASON_SELECTED = FL_↩
  REASON_SELECTED , FL_TREE_REASON_DESELECTED = FL_REASON_DESELECTED , FL_TREE_REASON_RESELE
  = FL_REASON_RESELECTED ,
  FL_TREE_REASON_OPENED = FL_REASON_OPENED , FL_TREE_REASON_CLOSED = FL_↩
  REASON_CLOSED , FL_TREE_REASON_DRAGGED = FL_REASON_DRAGGED }

  The reason the callback was invoked.

14.169.1 Detailed Description

This file contains the definitions of the Fl_Tree class.

14.169.2 Enumeration Type Documentation

14.169.2.1 Fl_Tree_Reason

```
enum Fl_Tree_Reason
```

The reason the callback was invoked.

Enumerator

FL_TREE_REASON_NONE	unknown reason
FL_TREE_REASON_SELECTED	an item was selected
FL_TREE_REASON_DESELECTED	an item was de-selected
FL_TREE_REASON_RESELECTED	an item was re-selected (double-clicked). See Fl_Tree_Item_Reselect_Mode to enable this.
FL_TREE_REASON_OPENED	an item was opened
FL_TREE_REASON_CLOSED	an item was closed
FL_TREE_REASON_DRAGGED	an item was dragged into a new place

14.170 Fl_Tree.H

Go to the documentation of this file.

```
00001 //
00002 // Fl_Tree.H -- This file is part of the Fl_Tree widget for FLTK
```

```
//
// Copyright 2009-2010 by Greg Ercolano.
// Copyright 2011-2026 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef FL_TREE_H
#define FL_TREE_H

#include <FL/Fl.H>
#include <FL/Fl_Group.H>
#include <FL/Fl_Scrollbar.H>
#include <FL/fl_draw.H>

#include <FL/Fl_Tree_Item.H>
#include <FL/Fl_Tree_Prefs.H>


enum Fl_Tree_Reason {
  FL_TREE_REASON_NONE       = FL_REASON_UNKNOWN,
  FL_TREE_REASON_SELECTED   = FL_REASON_SELECTED,
  FL_TREE_REASON_DESELECTED = FL_REASON_DESELECTED,
  FL_TREE_REASON_RESELECTED = FL_REASON_RESELECTED,
  FL_TREE_REASON_OPENED     = FL_REASON_OPENED,
  FL_TREE_REASON_CLOSED     = FL_REASON_CLOSED,
  FL_TREE_REASON_DRAGGED    = FL_REASON_DRAGGED
};

class FL_EXPORT Fl_Tree : public Fl_Group {
  friend class Fl_Tree_Item;
  Fl_Tree_Item  *_root;                        // can be null!
  Fl_Tree_Item  *_item_focus;                  // item that has focus box
  Fl_Tree_Item  *_callback_item;               // item invoked during callback (can be NULL)
  Fl_Tree_Reason _callback_reason;             // reason for the callback
  Fl_Tree_Prefs  _prefs;                       // all the tree's settings
  int            _scrollbar_size;              // size of scrollbar trough
  Fl_Tree_Item  *_lastselect;                  // last selected item
  char           _lastpushed;                  // FL_PUSH occurred on: 0=nothing, 1=open/close, 2=usericon, 3=label
  int            _auto_resize_children;        // if true: resize children when the Fl_Tree container is resized

  void           fix_scrollbar_order();        // internal: rearrange scrollbars in list of children

protected:
  Fl_Scrollbar *_vscroll;
  Fl_Scrollbar *_hscroll;
  int _tox,_toy,_tow,_toh;
  int _tix,_tiy,_tiw,_tih;

  int _tree_w;
  int _tree_h;
  void item_clicked(Fl_Tree_Item* val);
  void do_callback_for_item(Fl_Tree_Item* item, Fl_Tree_Reason reason);

  // next_visible_item() and extend_selection() moved to 'public' in ABI 1.3.3
  // undocmented draw_tree() dropped -- draw() does all the work now

  // draw() has to be protected per FLTK convention (was public in 1.3.x)
  void draw() override;

public:
  Fl_Tree(int X, int Y, int W, int H, const char *L=0);
  ~Fl_Tree();
  int handle(int e) override;
  void show_self();
  void resize(int,int,int,int) override;

  void auto_resize_children(int mode) {
    _auto_resize_children = (mode) ? 1 : 0;
  }

  int auto_resize_children() {
    return _auto_resize_children;
  }

  // root methods
  void root_label(const char *new_label);
```

```
Fl_Tree_Item* root();
void root(Fl_Tree_Item *newitem);
const Fl_Tree_Prefs& prefs() const { return _prefs; }

// Item creation/removal methods
Fl_Tree_Item *add(const char *path, Fl_Tree_Item *newitem=0);
Fl_Tree_Item* add(Fl_Tree_Item *parent_item, const char *name);
Fl_Tree_Item *insert_above(Fl_Tree_Item *above, const char *name);
Fl_Tree_Item* insert(Fl_Tree_Item *item, const char *name, int pos);
int remove(Fl_Tree_Item *item);
void clear();
void clear_children(Fl_Tree_Item *item);

// Item lookup methods
Fl_Tree_Item *find_item(const char *path);
const Fl_Tree_Item *find_item(const char *path) const;
int item_pathname(char *pathname, int pathnamelen, const Fl_Tree_Item *item) const;
const Fl_Tree_Item* find_clicked(int yonly=0) const;
Fl_Tree_Item* find_clicked(int yonly=0);
Fl_Tree_Item *item_clicked();
Fl_Tree_Item *first();
Fl_Tree_Item *first_visible();                // deprecated in ABI 10303
Fl_Tree_Item *first_visible_item();
Fl_Tree_Item *next(Fl_Tree_Item *item=0);
Fl_Tree_Item *prev(Fl_Tree_Item *item=0);
Fl_Tree_Item *last();
Fl_Tree_Item *last_visible();                 // deprecated in ABI 10303
Fl_Tree_Item *last_visible_item();
Fl_Tree_Item *next_visible_item(Fl_Tree_Item *start, int dir);        // made public in 1.3.3 ABI
Fl_Tree_Item *first_selected_item();
Fl_Tree_Item *last_selected_item();
Fl_Tree_Item *next_item(Fl_Tree_Item *item, int dir=FL_Down, bool visible=false);
Fl_Tree_Item *next_selected_item(Fl_Tree_Item *item=0, int dir=FL_Down);
int get_selected_items(Fl_Tree_Item_Array &items);

// Item open/close methods
int open(Fl_Tree_Item *item, int docallback=1);
int open(const char *path, int docallback=1);
void open_toggle(Fl_Tree_Item *item, int docallback=1);
int close(Fl_Tree_Item *item, int docallback=1);
int close(const char *path, int docallback=1);
int is_open(Fl_Tree_Item *item) const;
int is_open(const char *path) const;
int is_close(Fl_Tree_Item *item) const;
int is_close(const char *path) const;

// Item selection methods
int select(Fl_Tree_Item *item, int docallback=1);
int select(const char *path, int docallback=1);
void select_toggle(Fl_Tree_Item *item, int docallback=1);
int deselect(Fl_Tree_Item *item, int docallback=1);
int deselect(const char *path, int docallback=1);
int deselect_all(Fl_Tree_Item *item=0, int docallback=1);
int select_only(Fl_Tree_Item *selitem, int docallback=1);
int select_all(Fl_Tree_Item *item=0, int docallback=1);
int extend_selection_dir(Fl_Tree_Item *from,
                         Fl_Tree_Item *to,
                         int dir,
                         int val,
                         bool visible);
int extend_selection(Fl_Tree_Item *from,
                     Fl_Tree_Item *to,
                     int val=1,
                     bool visible=false);
void set_item_focus(Fl_Tree_Item *item);
Fl_Tree_Item *get_item_focus() const;
int is_selected(Fl_Tree_Item *item) const;
int is_selected(const char *path);

// Item attribute related methods
Fl_Font     item_labelfont() const;
void        item_labelfont(Fl_Font val);
Fl_Fontsize item_labelsize() const;
void        item_labelsize(Fl_Fontsize val);
Fl_Color    item_labelfgcolor(void) const;
void        item_labelfgcolor(Fl_Color val);
Fl_Color    item_labelbgcolor(void) const;
void        item_labelbgcolor(Fl_Color val);
Fl_Color connectorcolor() const;
void connectorcolor(Fl_Color val);
int marginleft() const;
void marginleft(int val);
int margintop() const;
void margintop(int val);
int marginbottom() const;
void marginbottom(int val);
int linespacing() const;
```

```
  void linespacing(int val);
  int openchild_marginbottom() const;
  void openchild_marginbottom(int val);
  int usericonmarginleft() const;
  void usericonmarginleft(int val);
  int labelmarginleft() const;
  void labelmarginleft(int val);
  int widgetmarginleft() const;
  void widgetmarginleft(int val);
  int connectorwidth() const;
  void connectorwidth(int val);
  Fl_Image* usericon() const;
  void usericon(Fl_Image *val);
  Fl_Image* openicon() const;
  void openicon(Fl_Image *val);
  Fl_Image* closeicon() const;
  void closeicon(Fl_Image *val);
  int showcollapse() const;
  void showcollapse(int val);
  int showroot() const;
  void showroot(int val);
  Fl_Tree_Connector connectorstyle() const;
  void connectorstyle(Fl_Tree_Connector val);
  Fl_Tree_Sort sortorder() const;
  void sortorder(Fl_Tree_Sort val);
  Fl_Boxtype selectbox() const;
  void selectbox(Fl_Boxtype val);
  Fl_Tree_Select selectmode() const;
  void selectmode(Fl_Tree_Select val);
  Fl_Tree_Item_Reselect_Mode item_reselect_mode() const;
  void item_reselect_mode(Fl_Tree_Item_Reselect_Mode mode);
  Fl_Tree_Item_Draw_Mode item_draw_mode() const;
  void item_draw_mode(Fl_Tree_Item_Draw_Mode mode);
  void item_draw_mode(int mode);
  void calc_dimensions();
  void calc_tree();
  void recalc_tree();
  int displayed(Fl_Tree_Item *item);
  void show_item(Fl_Tree_Item *item, int yoff);
  void show_item(Fl_Tree_Item *item);
  void show_item_top(Fl_Tree_Item *item);
  void show_item_middle(Fl_Tree_Item *item);
  void show_item_bottom(Fl_Tree_Item *item);
  void display(Fl_Tree_Item *item);
  int  vposition() const;
  void vposition(int pos);
  int  hposition() const;
  void hposition(int pos);

  int is_scrollbar(Fl_Widget *w);
  int scrollbar_size() const;
  void scrollbar_size(int size);
  int is_vscroll_visible() const;
  int is_hscroll_visible() const;

  // callback related
  void callback_item(Fl_Tree_Item* item);
  Fl_Tree_Item* callback_item();
  void callback_reason(Fl_Tree_Reason reason);
  Fl_Tree_Reason callback_reason() const;

  void load(class Fl_Preferences&);
};

#endif /*FL_TREE_H*/
```

14.171 Fl_Tree_Item.H File Reference

This file contains the definitions for Fl_Tree_Item.

```
#include <FL/Fl.H>
#include <FL/Fl_Widget.H>
#include <FL/Fl_Image.H>
#include <FL/fl_draw.H>
#include <FL/Fl_Tree_Item_Array.H>
#include <FL/Fl_Tree_Prefs.H>
```

Classes

- class Fl_Tree_Item

  Tree widget item.

14.171.1 Detailed Description

This file contains the definitions for Fl_Tree_Item.

14.172 Fl_Tree_Item.H

Go to the documentation of this file.

```
00001 //
00002
00003 #ifndef FL_TREE_ITEM_H
00004 #define FL_TREE_ITEM_H
00005
00006 #include <FL/Fl.H>
00007 #include <FL/Fl_Widget.H>
00008 #include <FL/Fl_Image.H>
00009 #include <FL/fl_draw.H>
00010
00011 #include <FL/Fl_Tree_Item_Array.H>
00012 #include <FL/Fl_Tree_Prefs.H>
00013
00015 // FL/Fl_Tree_Item.H
00017 //
00018 // Fl_Tree -- This file is part of the Fl_Tree widget for FLTK
00019 // Copyright (C) 2009-2010 by Greg Ercolano.
00020 //
00021 // This library is free software. Distribution and use rights are outlined in
00022 // the file "COPYING" which should have been included with this file.  If this
00023 // file is missing or damaged, see the license at:
00024 //
00025 //     https://www.fltk.org/COPYING.php
00026 //
00027 // Please see the following page on how to report bugs and issues:
00028 //
00029 //     https://www.fltk.org/bugs.php
00030 //
00031
00036
00064 class Fl_Tree;
00065 class FL_EXPORT Fl_Tree_Item {
00066   Fl_Tree                *_tree;                // parent tree
00067   const char             *_label;               // label (memory managed)
00068   Fl_Font                _labelfont;           // label's font face
00069   Fl_Fontsize            _labelsize;           // label's font size
00070   Fl_Color               _labelfgcolor;        // label's fg color
00071   Fl_Color               _labelbgcolor;        // label's bg color (0xffffffff is 'transparent')
00073   enum Fl_Tree_Item_Flags {
00074     OPEN                = 1«0,
00075     VISIBLE             = 1«1,
00076     ACTIVE              = 1«2,
00077     SELECTED            = 1«3
00078   };
00079   unsigned short _flags;                // misc flags
00080   int                    _xywh[4];             // xywh of this widget (if visible)
00081   int                    _collapse_xywh[4];    // xywh of collapse icon (if visible)
00082   int                    _label_xywh[4];       // xywh of label
00083   Fl_Widget              *_widget;             // item's label widget (optional)
00084   Fl_Image               *_usericon;           // item's user-specific icon (optional)
00085   Fl_Image               *_userdeicon;         // deactivated usericon
00086   Fl_Tree_Item_Array     _children;            // array of child items
00087   Fl_Tree_Item           *_parent;             // parent item (=0 if root)
00088   void                   *_userdata;           // user data that can be associated with an item
00089   Fl_Tree_Item           *_prev_sibling;       // previous sibling (same level)
00090   Fl_Tree_Item           *_next_sibling;       // next sibling (same level)
00091   // Protected methods
00092 protected:
00093   void _Init(const Fl_Tree_Prefs &prefs, Fl_Tree *tree);
00094   void show_widgets();
00095   void hide_widgets();
00096   virtual void draw_vertical_connector(int x, int y1, int y2, const Fl_Tree_Prefs &prefs);
00097   virtual void draw_horizontal_connector(int x1, int x2, int y, const Fl_Tree_Prefs &prefs);
00098   void recalc_tree();
00099   int calc_item_height(const Fl_Tree_Prefs &prefs) const;
00100   Fl_Color drawfgcolor() const;
00101   Fl_Color drawbgcolor() const;
00102
00103 public:
```

```
Fl_Tree_Item(const Fl_Tree_Prefs &prefs);     // CTOR -- backwards compatible
Fl_Tree_Item(Fl_Tree *tree);                  // CTOR -- ABI 1.3.3+
virtual ~Fl_Tree_Item();                      // DTOR -- ABI 1.3.3+
Fl_Tree_Item(const Fl_Tree_Item *o);          // COPY CTOR
int x() const { return(_xywh[0]); }
int y() const { return(_xywh[1]); }
int w() const { return(_xywh[2]); }
int h() const { return(_xywh[3]); }
int label_x() const { return(_label_xywh[0]); }
int label_y() const { return(_label_xywh[1]); }
int label_w() const { return(_label_xywh[2]); }
int label_h() const { return(_label_xywh[3]); }
virtual int draw_item_content(int render);
void draw(int X, int &Y, int W, Fl_Tree_Item *itemfocus,
          int &tree_item_xmax, int lastchild=1, int render=1);
void show_self(const char *indent = "") const;
void label(const char *val);
const char *label() const;

inline void user_data( void* data ) { _userdata = data; }

inline void* user_data() const { return _userdata; }

void labelfont(Fl_Font val) {
  _labelfont = val;
  recalc_tree();              // may change tree geometry
}
Fl_Font labelfont() const {
  return(_labelfont);
}
void labelsize(Fl_Fontsize val) {
  _labelsize = val;
  recalc_tree();              // may change tree geometry
}
Fl_Fontsize labelsize() const {
  return(_labelsize);
}
void labelfgcolor(Fl_Color val) {
  _labelfgcolor = val;
}
Fl_Color labelfgcolor() const {
  return(_labelfgcolor);
}
void labelcolor(Fl_Color val) {
   labelfgcolor(val);
}
Fl_Color labelcolor() const {
  return labelfgcolor();
}
void labelbgcolor(Fl_Color val) {
  _labelbgcolor = val;
}
Fl_Color labelbgcolor() const {
  return(_labelbgcolor);
}
void widget(Fl_Widget *val) {
  _widget = val;
  recalc_tree();              // may change tree geometry
}
Fl_Widget *widget() const {
  return(_widget);
}
int children() const {
  return(_children.total());
}
Fl_Tree_Item *child(int index) {
  return(_children[index]);
}
const Fl_Tree_Item *child(int t) const;
int has_children() const {
  return(children());
}
int find_child(const char *name);
int find_child(Fl_Tree_Item *item);
int remove_child(Fl_Tree_Item *item);
int remove_child(const char *new_label);
void clear_children();
void swap_children(int ax, int bx);
int swap_children(Fl_Tree_Item *a, Fl_Tree_Item *b);
const Fl_Tree_Item *find_child_item(const char *name) const;
      Fl_Tree_Item *find_child_item(const char *name);
const Fl_Tree_Item *find_child_item(char **arr) const;
      Fl_Tree_Item *find_child_item(char **arr);
const Fl_Tree_Item *find_item(char **arr) const;
      Fl_Tree_Item *find_item(char **arr);
// Adding items
Fl_Tree_Item *add(const Fl_Tree_Prefs &prefs,
```

```
                    const char *new_label,
                    Fl_Tree_Item *newitem);
  Fl_Tree_Item *add(const Fl_Tree_Prefs &prefs,
                    const char *new_label);
  Fl_Tree_Item *add(const Fl_Tree_Prefs &prefs,
                    char **arr,
                    Fl_Tree_Item *newitem);
  Fl_Tree_Item *add(const Fl_Tree_Prefs &prefs,
                    char **arr);
  Fl_Tree_Item *replace(Fl_Tree_Item *new_item);
  Fl_Tree_Item *replace_child(Fl_Tree_Item *olditem, Fl_Tree_Item *newitem);
  Fl_Tree_Item *insert(const Fl_Tree_Prefs &prefs, const char *new_label, int pos=0);
  Fl_Tree_Item *insert_above(const Fl_Tree_Prefs &prefs, const char *new_label);
  Fl_Tree_Item* deparent(int index);
  int reparent(Fl_Tree_Item *newchild, int index);
  int move(int to, int from);
  int move(Fl_Tree_Item *item, int op=0, int pos=0);
  int move_above(Fl_Tree_Item *item);
  int move_below(Fl_Tree_Item *item);
  int move_into(Fl_Tree_Item *item, int pos=0);
  int depth() const;
  Fl_Tree_Item *prev();
  Fl_Tree_Item *next();
  Fl_Tree_Item *next_sibling();
  Fl_Tree_Item *prev_sibling();
  void update_prev_next(int index);
  Fl_Tree_Item *next_displayed(Fl_Tree_Prefs &prefs);   // deprecated
  Fl_Tree_Item *prev_displayed(Fl_Tree_Prefs &prefs);   // deprecated
  Fl_Tree_Item *next_visible(Fl_Tree_Prefs &prefs);
  Fl_Tree_Item *prev_visible(Fl_Tree_Prefs &prefs);

  Fl_Tree_Item *parent() {
    return(_parent);
  }
  const Fl_Tree_Item *parent() const {
    return(_parent);
  }
  void parent(Fl_Tree_Item *val) {
    _parent = val;
  }
  const Fl_Tree_Prefs& prefs() const;
  const Fl_Tree *tree() const {
    return(_tree);
  }
  Fl_Tree *tree() {
    return(_tree);
  }
  // State
  void open();
  void close();
  int is_open() const {
    return(is_flag(OPEN));
  }
  int is_close() const {
    return(is_flag(OPEN)?0:1);
  }
  void open_toggle() {
    is_open()?close():open();   // handles calling recalc_tree()
  }
  void select(int val=1) {
    set_flag(SELECTED, val);
  }
  void select_toggle() {
    if ( is_selected() ) {
      deselect();       // deselect if selected
    } else {
      select();         // select if deselected
    }
  }
  int select_all() {
    int count = 0;
    if ( ! is_selected() ) {
      select();
      ++count;
    }
    for ( int t=0; t<children(); t++ ) {
      count += child(t)->select_all();
    }
    return(count);
  }
  void deselect() {
    set_flag(SELECTED, 0);
  }
  int deselect_all() {
    int count = 0;
    if ( is_selected() ) {
      deselect();
```

```
      ++count;
    }
    for ( int t=0; t<children(); t++ ) {
      count += child(t)->deselect_all();
    }
    return(count);
  }
  char is_selected() const {
    return (char)is_flag(SELECTED);
  }
  void activate(int val=1) {
    set_flag(ACTIVE,val);
    if ( _widget && val != (int)_widget->active() ) {
      if ( val ) {
        _widget->activate();
      } else {
        _widget->deactivate();
      }
      _widget->redraw();
    }
  }
  void deactivate() {
    activate(0);
  }
  char is_activated() const {
    return (char)is_flag(ACTIVE);
  }
  char is_active() const {
    return(is_activated());
  }
  int visible() const {
    return(is_visible());
  }
  int is_visible() const {
    return(is_flag(VISIBLE));
  }
  int visible_r() const {
    return(is_visible_r());
  }
  int is_visible_r() const;

  void usericon(Fl_Image *val) {
    _usericon = val;
    recalc_tree();              // may change tree geometry
  }
  Fl_Image *usericon() const {
    return(_usericon);
  }
  void userdeicon(Fl_Image* val) {
    _userdeicon = val;
  }
  Fl_Image* userdeicon() const {
    return _userdeicon;
  }
  // Events
  const Fl_Tree_Item* find_clicked(const Fl_Tree_Prefs &prefs, int yonly=0) const;
  Fl_Tree_Item* find_clicked(const Fl_Tree_Prefs &prefs, int yonly=0);
  int event_on_item(const Fl_Tree_Prefs &prefs) const;
  int event_on_collapse_icon(const Fl_Tree_Prefs &prefs) const;
  int event_on_user_icon(const Fl_Tree_Prefs &prefs) const;
  int event_on_label(const Fl_Tree_Prefs &prefs) const;
  int is_root() const {
    return(_parent==0?1:0);
  }

  // Protected methods
  // TODO: move these to top 'protected:' section
protected:
  inline void set_flag(unsigned short flag,int val) {
    if ( flag==OPEN || flag==VISIBLE ) {
      recalc_tree();            // may change tree geometry
    }
    if ( val ) _flags |= flag; else _flags &= ~flag;
  }
  inline int is_flag(unsigned short val) const {
    return(_flags & val ? 1 : 0);
  }

};

#endif /*FL_TREE_ITEM_H*/
```

14.173 Fl_Tree_Item_Array.H File Reference

This file defines a class that manages an array of Fl_Tree_Item pointers.

```
#include <FL/Fl.H>
#include "Fl_Export.H"
```

Classes

- class Fl_Tree_Item_Array

  Manages an array of Fl_Tree_Item pointers.

14.173.1 Detailed Description

This file defines a class that manages an array of Fl_Tree_Item pointers.

14.174 Fl_Tree_Item_Array.H

Go to the documentation of this file.

```
00001 //
00002
00003 #ifndef _FL_TREE_ITEM_ARRAY_H
00004 #define _FL_TREE_ITEM_ARRAY_H
00005
00006 #include <FL/Fl.H>
00007 #include "Fl_Export.H"
00008
00009 class FL_EXPORT Fl_Tree_Item;   // forward decl must *precede* first doxygen comment block
00010                                 // or doxygen will not document our class..
00011
00013 // FL/Fl_Tree_Item_Array.H
00015 //
00016 // Fl_Tree -- This file is part of the Fl_Tree widget for FLTK
00017 // Copyright (C) 2009-2010 by Greg Ercolano.
00018 //
00019 // This library is free software. Distribution and use rights are outlined in
00020 // the file "COPYING" which should have been included with this file.  If this
00021 // file is missing or damaged, see the license at:
00022 //
00023 //     https://www.fltk.org/COPYING.php
00024 //
00025 // Please see the following page on how to report bugs and issues:
00026 //
00027 //     https://www.fltk.org/bugs.php
00028 //
00029
00034
00044
00045 class FL_EXPORT Fl_Tree_Item_Array {
00046   Fl_Tree_Item **_items;        // items array
00047   int _total;                   // #items in array
00048   int _size;                    // #items *allocated* for array
00049   int _chunksize;               // #items to enlarge mem allocation
00050   enum {
00051     MANAGE_ITEM = 1
00052   };
00053   char _flags;                  // flags to control behavior
00054   void enlarge(int count);
00055 public:
00056   Fl_Tree_Item_Array(int new_chunksize = 10);           // CTOR
00057   ~Fl_Tree_Item_Array();                                // DTOR
00058   Fl_Tree_Item_Array(const Fl_Tree_Item_Array *o);      // COPY CTOR
00060   Fl_Tree_Item *operator[](int i) {
00061     return(_items[i]);
00062   }
00064   const Fl_Tree_Item *operator[](int i) const {
00065     return(_items[i]);
00066   }
00068   int total() const {
00069     return(_total);
00070   }
00072   void swap(int ax, int bx);
00073   int move(int to, int from);
00074   int deparent(int pos);
00075   int reparent(Fl_Tree_Item *item, Fl_Tree_Item *newparent, int pos);
00076   void clear();
00077   void add(Fl_Tree_Item *val);
```

```
  void insert(int pos, Fl_Tree_Item *new_item);
  void replace(int pos, Fl_Tree_Item *new_item);
  void remove(int index);
  int  remove(Fl_Tree_Item *item);
  void manage_item_destroy(int val) {
    if ( val ) _flags |= MANAGE_ITEM; else _flags &= ~MANAGE_ITEM;
  }
  int manage_item_destroy() const {
    return _flags & MANAGE_ITEM ? 1 : 0;
  }
};

#endif /*_FL_TREE_ITEM_ARRAY_H*/
```

14.175 Fl_Tree_Prefs.H File Reference

This file contains the definitions for Fl_Tree's preferences.

`#include <FL/Fl.H>`

Classes

- class Fl_Tree_Prefs

  Tree widget's preferences.

Typedefs

- typedef void() **Fl_Tree_Item_Draw_Callback**(Fl_Tree_Item *, void *)

Enumerations

- enum Fl_Tree_Connector { FL_TREE_CONNECTOR_NONE =0 , FL_TREE_CONNECTOR_DOTTED =1 , FL_TREE_CONNECTOR_SOLID =2 }

  Defines the style of connection lines between items.
- enum Fl_Tree_Item_Draw_Mode { FL_TREE_ITEM_DRAW_DEFAULT =0 , FL_TREE_ITEM_DRAW_LABEL_AND_WIDGET =1 , FL_TREE_ITEM_HEIGHT_FROM_WIDGET =2 }

  Bit flags that control how item's labels and widget()s are drawn in the tree via item_draw_mode().
- enum Fl_Tree_Item_Reselect_Mode { FL_TREE_SELECTABLE_ONCE =0 , FL_TREE_SELECTABLE_ALWAYS }

  Defines the ways an item can be (re) selected via item_reselect_mode().
- enum Fl_Tree_Select { FL_TREE_SELECT_NONE =0 , FL_TREE_SELECT_SINGLE =1 , FL_TREE_SELECT_MULTI =2 , FL_TREE_SELECT_SINGLE_DRAGGABLE =3 }

  Tree selection style.
- enum Fl_Tree_Sort { FL_TREE_SORT_NONE =0 , FL_TREE_SORT_ASCENDING =1 , FL_TREE_SORT_DESCENDING =2 }

  Sort order options for items added to the tree.

14.175.1 Detailed Description

This file contains the definitions for Fl_Tree's preferences.

```
   Fl_Tree_Prefs
        :
   .....:.......
   :           :
Fl_Tree        :
   |_____ Fl_Tree_Item
```

14.175.2 Enumeration Type Documentation

14.175.2.1 Fl_Tree_Connector

`enum Fl_Tree_Connector`

Defines the style of connection lines between items.

Enumerator

FL_TREE_CONNECTOR_NONE	Use no lines connecting items.
FL_TREE_CONNECTOR_DOTTED	Use dotted lines connecting items (default)
FL_TREE_CONNECTOR_SOLID	Use solid lines connecting items.

14.175.2.2 Fl_Tree_Item_Draw_Mode

`enum Fl_Tree_Item_Draw_Mode`

Bit flags that control how item's labels and widget()s are drawn in the tree via item_draw_mode().

Enumerator

FL_TREE_ITEM_DRAW_DEFAULT	If widget() defined, draw in place of label, and widget() tracks item height (default)
FL_TREE_ITEM_DRAW_LABEL_AND_WIDGET	If widget() defined, include label to the left of the widget.
FL_TREE_ITEM_HEIGHT_FROM_WIDGET	If widget() defined, widget()'s height controls item's height.

14.175.2.3 Fl_Tree_Item_Reselect_Mode

`enum Fl_Tree_Item_Reselect_Mode`

Defines the ways an item can be (re) selected via item_reselect_mode().

Enumerator

FL_TREE_SELECTABLE_ONCE	Item can only be selected once (default)
FL_TREE_SELECTABLE_ALWAYS	Enables FL_TREE_REASON_RESELECTED events for callbacks.

14.175.2.4 Fl_Tree_Select

`enum Fl_Tree_Select`

Tree selection style.

Enumerator

FL_TREE_SELECT_NONE	Nothing selected when items are clicked.
FL_TREE_SELECT_SINGLE	Single item selected when item is clicked (default)
FL_TREE_SELECT_MULTI	Multiple items can be selected by clicking with SHIFT, CTRL or mouse drags.
FL_TREE_SELECT_SINGLE_DRAGGABLE	Single items may be selected, and they may be reordered by mouse drag.

14.175.2.5 Fl_Tree_Sort

`enum Fl_Tree_Sort`

Sort order options for items added to the tree.

Enumerator

FL_TREE_SORT_NONE	No sorting; items are added in the order defined (default).
FL_TREE_SORT_ASCENDING	Add items in ascending sort order.
FL_TREE_SORT_DESCENDING	Add items in descending sort order.

14.176 Fl_Tree_Prefs.H

Go to the documentation of this file.

```
00001 //
00002
00003 #ifndef FL_TREE_PREFS_H
00004 #define FL_TREE_PREFS_H
00005
00006 #include <FL/Fl.H>      // needed for ABI version features (via Enumerations.H)
00007
00009 // FL/Fl_Tree_Prefs.H
00011 //
00012 // Fl_Tree_Prefs -- This file is part of the Fl_Tree widget for FLTK
00013 // Copyright (C) 2009-2010 by Greg Ercolano.
00014 //
00015 // This library is free software. Distribution and use rights are outlined in
00016 // the file "COPYING" which should have been included with this file.  If this
00017 // file is missing or damaged, see the license at:
00018 //
00019 //     https://www.fltk.org/COPYING.php
00020 //
00021 // Please see the following page on how to report bugs and issues:
00022 //
00023 //     https://www.fltk.org/bugs.php
00024 //
00025
00040
00043
00047 enum Fl_Tree_Sort {
00048   FL_TREE_SORT_NONE=0,
00049   FL_TREE_SORT_ASCENDING=1,
00050   FL_TREE_SORT_DESCENDING=2
00051 };
00052
00056 enum Fl_Tree_Connector {
00057   FL_TREE_CONNECTOR_NONE=0,
00058   FL_TREE_CONNECTOR_DOTTED=1,
00059   FL_TREE_CONNECTOR_SOLID=2
00060 };
00061
00065 enum Fl_Tree_Select {
00066   FL_TREE_SELECT_NONE=0,
00067   FL_TREE_SELECT_SINGLE=1,
00068   FL_TREE_SELECT_MULTI=2,
00070   FL_TREE_SELECT_SINGLE_DRAGGABLE=3
00072 };
00073
00078 enum Fl_Tree_Item_Reselect_Mode {
00079   FL_TREE_SELECTABLE_ONCE=0,
00080   FL_TREE_SELECTABLE_ALWAYS
00081 };
00082
00087 enum Fl_Tree_Item_Draw_Mode {
00088   FL_TREE_ITEM_DRAW_DEFAULT=0,
00090   FL_TREE_ITEM_DRAW_LABEL_AND_WIDGET=1,
00091   FL_TREE_ITEM_HEIGHT_FROM_WIDGET=2
00092 };
00093
00094 class Fl_Tree_Item;
00095 typedef void (Fl_Tree_Item_Draw_Callback)(Fl_Tree_Item*, void*);
00096
00105 class FL_EXPORT Fl_Tree_Prefs {
00106   Fl_Font _labelfont;                   // label's font face
00107   Fl_Fontsize _labelsize;               // label's font size
00108   int _margintop;                       // --
00109   int _marginleft;                      //   |- tree's controllable margins
00110   int _marginbottom;                    // --
00111   int _openchild_marginbottom;          // extra space below an open child tree
00112   int _usericonmarginleft;              // space to left of user icon (if any)
00113   int _labelmarginleft;                 // space to left of label
00114   int _widgetmarginleft;                // space to left of widget
00115   int _connectorwidth;                  // connector width (right of open/close icon)
00116   int _linespacing;                     // vertical space between lines
00117   // Colors
00118   Fl_Color _labelfgcolor;               // label's foreground color
00119   Fl_Color _labelbgcolor;               // label's background color
00120   Fl_Color _connectorcolor;             // connector dotted line color
00121   Fl_Tree_Connector _connectorstyle;    // connector line style
00122   Fl_Image *_openimage;                 // the 'open' icon [+]
00123   Fl_Image *_closeimage;                // the 'close' icon [-]
00124   Fl_Image *_userimage;                 // user's own icon
00125   Fl_Image *_opendeimage;               // deactivated 'open' icon
00126   Fl_Image *_closedeimage;              // deactivated 'close' icon
00127   Fl_Image *_userdeimage;               // deactivated user icon
00128   char _showcollapse;                   // 1=show collapse icons, 0=don't
00129   char _showroot;                       // show the root item as part of the tree
```

```
  Fl_Tree_Sort   _sortorder;            // none, ascening, descending, etc.
  Fl_Boxtype     _selectbox;            // selection box type
  Fl_Tree_Select _selectmode;           // selection mode
  Fl_Tree_Item_Reselect_Mode _itemreselectmode; // controls item selection callback() behavior
  Fl_Tree_Item_Draw_Mode     _itemdrawmode;     // controls how items draw label + widget()
  Fl_Tree_Item_Draw_Callback *_itemdrawcallback;        // callback to handle drawing items (0=none)
  void                       *_itemdrawuserdata;        // data for drawing items (0=none)
public:
  Fl_Tree_Prefs();
  ~Fl_Tree_Prefs();

  // Labels
  inline Fl_Font item_labelfont() const { return(_labelfont); }
  inline void item_labelfont(Fl_Font val) { _labelfont = val; }
  inline Fl_Fontsize item_labelsize() const { return(_labelsize); }
  inline void item_labelsize(Fl_Fontsize val) { _labelsize = val; }
  inline Fl_Color item_labelfgcolor() const { return(_labelfgcolor); }
  inline void item_labelfgcolor(Fl_Color val) { _labelfgcolor = val; }
  inline Fl_Color item_labelbgcolor() const {
    return _labelbgcolor;
  }
  inline void item_labelbgcolor(Fl_Color val) {
    _labelbgcolor = val;
  }

  // Obsolete names - for 1.3.0 backwards compat
  inline Fl_Font labelfont() const { return(_labelfont); }
  inline void labelfont(Fl_Font val) { _labelfont = val; }
  inline Fl_Fontsize labelsize() const { return(_labelsize); }
  inline void labelsize(Fl_Fontsize val) { _labelsize = val; }
  inline Fl_Color labelfgcolor() const { return(_labelfgcolor); }
  inline void labelfgcolor(Fl_Color val) { _labelfgcolor = val; }
  inline Fl_Color labelbgcolor() const { return(item_labelbgcolor()); }
  inline void labelbgcolor(Fl_Color val) { item_labelbgcolor(val); }

  // Margins
  inline int marginleft() const {
    return(_marginleft);
  }
  inline void marginleft(int val) {
    _marginleft = val;
  }
  inline int margintop() const {
    return(_margintop);
  }
  inline void margintop(int val) {
    _margintop = val;
  }
  inline int marginbottom() const {
    return(_marginbottom);
  }
  inline void marginbottom(int val) {
    _marginbottom = val;
  }
  inline int openchild_marginbottom() const {
    return(_openchild_marginbottom);
  }
  inline void openchild_marginbottom(int val) {
    _openchild_marginbottom = val;
  }
  inline int usericonmarginleft() const {
    return(_usericonmarginleft);
  }
  inline void usericonmarginleft(int val) {
    _usericonmarginleft = val;
  }
  inline int labelmarginleft() const {
    return(_labelmarginleft);
  }
  inline void labelmarginleft(int val) {
    _labelmarginleft = val;
  }
  inline int widgetmarginleft() const {
    return(_widgetmarginleft);
  }
  inline void widgetmarginleft(int val) {
    _widgetmarginleft = val;
  }
  inline int linespacing() const {
    return(_linespacing);
  }
  inline void linespacing(int val) {
    _linespacing = val;
  }

  // Colors and Styles
  inline Fl_Color connectorcolor() const {
```

```
    return(_connectorcolor);
  }
  inline void connectorcolor(Fl_Color val) {
    _connectorcolor = val;
  }
  inline Fl_Tree_Connector connectorstyle() const {
    return(_connectorstyle);
  }
  inline void connectorstyle(Fl_Tree_Connector val) {
    _connectorstyle = val;
  }
  inline void connectorstyle(int val) {
    _connectorstyle = Fl_Tree_Connector(val);
  }
  inline int connectorwidth() const {
    return(_connectorwidth);
  }
  inline void connectorwidth(int val) {
    _connectorwidth = val;
  }

  // Icons
  inline Fl_Image *openicon() const {
    return(_openimage);
  }
  inline int openicon_w() const { return _openimage ? _openimage->w() : 11; }
  inline int openicon_h() const { return _openimage ? _openimage->h() : 11; }
  void openicon(Fl_Image *val);
  inline Fl_Image *closeicon() const {
    return(_closeimage);
  }
  inline int closeicon_w() const { return _closeimage ? _closeimage->w() : 11; }
  inline int closeicon_h() const { return _closeimage ? _closeimage->h() : 11; }
  void closeicon(Fl_Image *val);
  inline Fl_Image *usericon() const {
    return(_userimage);
  }
  inline void usericon(Fl_Image *val) {
    _userimage = val;
    // Update deactivated version of icon..
    if ( _userdeimage ) delete _userdeimage;
    if ( _userimage ) {
      _userdeimage = _userimage->copy();
      _userdeimage->inactive();
    } else {
      _userdeimage = 0;
    }
  }

  inline Fl_Image *opendeicon() const {
    return _opendeimage;
  }
  inline Fl_Image *closedeicon() const {
    return _closedeimage;
  }
  inline Fl_Image *userdeicon() const {
    return _userdeimage;
  }

  // Options
  inline char showcollapse() const {
    return(_showcollapse);
  }
  inline void showcollapse(int val) {
    _showcollapse = (char)val;
  }
  inline Fl_Tree_Sort sortorder() const {
    return(_sortorder);
  }
  inline void sortorder(Fl_Tree_Sort val) {
    _sortorder = val;
  }
  inline Fl_Boxtype selectbox() const {
    return(_selectbox);
  }
  inline void selectbox(Fl_Boxtype val) {
    _selectbox = val;
  }
  inline int showroot() const {
    return(int(_showroot));
  }
  inline void showroot(int val) {
    _showroot = char(val);
  }
  inline Fl_Tree_Select selectmode() const {
    return(_selectmode);
  }
```

```
  inline void selectmode(Fl_Tree_Select val) {
    _selectmode = val;
  }
  Fl_Tree_Item_Reselect_Mode item_reselect_mode() const {
    return _itemreselectmode;
  }
  void item_reselect_mode(Fl_Tree_Item_Reselect_Mode mode) {
    _itemreselectmode = mode;
  }
  inline Fl_Tree_Item_Draw_Mode item_draw_mode() const {
    return(_itemdrawmode);
  }
  inline void item_draw_mode(Fl_Tree_Item_Draw_Mode val) {
    _itemdrawmode = val;
  }
  void item_draw_callback(Fl_Tree_Item_Draw_Callback *cb, void *data=0) {
    _itemdrawcallback = cb;
    _itemdrawuserdata = data;
  }
  Fl_Tree_Item_Draw_Callback* item_draw_callback() const {
    return(_itemdrawcallback);
  }
  void* item_draw_user_data() const {
    return(_itemdrawuserdata);
  }
  void do_item_draw_callback(Fl_Tree_Item *o) const {
    _itemdrawcallback(o, _itemdrawuserdata);
  }
};

#endif /*FL_TREE_PREFS_H*/
```

14.177 fl_types.h File Reference

This file contains simple "C"-style type definitions.

`#include "fl_attr.h"`

Typedefs

Miscellaneous

- typedef unsigned int Fl_Shortcut

  16-bit Unicode character + 8-bit indicator for keyboard flags.
- typedef unsigned char **uchar**

  unsigned char
- typedef unsigned long **ulong**

  unsigned long

14.177.1 Detailed Description

This file contains simple "C"-style type definitions.

14.177.2 Typedef Documentation

14.177.2.1 Fl_Shortcut

`typedef unsigned int Fl_Shortcut`

16-bit Unicode character + 8-bit indicator for keyboard flags.

Note

This **should** be 24-bit Unicode character + 8-bit indicator for keyboard flags. The upper 8 bits are currently unused but reserved.

Due to compatibility issues this type and all FLTK **shortcuts** can only be used with 16-bit Unicode characters (`U+0000 .. U+FFFF`) and not with the full range of unicode characters (`U+0000 .. U+10FFFF`). This is caused by the bit flags `FL_SHIFT`, `FL_CTRL`, `FL_ALT`, and `FL_META` being all in the range `0x010000 .. 0x400000`.

Todo Discuss and decide whether we can "shift" these special keyboard flags to the upper byte to enable full 21-bit Unicode characters (`U+0000 .. U+10FFFF`) plus the keyboard indicator bits as this was originally intended. This would be possible if we could rely on **all** programs being coded with symbolic names and not hard coded bit values.

14.178 fl_types.h

Go to the documentation of this file.

```
00001 /*
00002  * Simple "C"-style types for the Fast Light Tool Kit (FLTK).
00003  *
00004  * Copyright 1998-2020 by Bill Spitzak and others.
00005  *
00006  * This library is free software. Distribution and use rights are outlined in
00007  * the file "COPYING" which should have been included with this file.  If this
00008  * file is missing or damaged, see the license at:
00009  *
00010  *     https://www.fltk.org/COPYING.php
00011  *
00012  * Please see the following page on how to report bugs and issues:
00013  *
00014  *     https://www.fltk.org/bugs.php
00015  */
00016
00021 #ifndef FL_TYPES_H
00022 #define FL_TYPES_H
00023
00024 #include "fl_attr.h"
00025
/* group:  Miscellaneous */  00028
00030 typedef unsigned char uchar;
00032 typedef unsigned long ulong;
00033
00055 typedef unsigned int Fl_Shortcut;
00056
/* group:  Miscellaneous */  00058
00059 #endif
```

14.179 fl_utf8.h File Reference

header for Unicode and UTF-8 character handling

```
#include "Fl_Export.H"
#include "fl_types.h"
#include <stdio.h>
#include <sys/stat.h>
```

Functions

- int fl_access (const char *f, int mode)

  Cross-platform function to test a files access() with a UTF-8 encoded name or value.
- int fl_chdir (const char *path)

  Cross-platform function to change the current working directory, given as a UTF-8 encoded string.
- int fl_chmod (const char *f, int mode)

  Cross-platform function to set a files mode() with a UTF-8 encoded name or value.
- int fl_close_fd (int fd)

  Cross-platform function to close a file descriptor.
- int fl_execvp (const char *file, char *const *argv)

  Calls `execvp` on all platforms.
- FILE * fl_fopen (const char *f, const char *mode)

  Cross-platform function to open files with a UTF-8 encoded name.
- char * fl_getcwd (char *buf, int len)

  Cross-platform function to get the current working directory as a UTF-8 encoded value.
- char * fl_getenv (const char *v)

Cross-platform function to get environment variables with a UTF-8 encoded name or value.

- char fl_make_path (const char *path)

  Cross-platform function to recursively create a path in the file system.
- void fl_make_path_for_file (const char *path)

  Cross-platform function to create a path for the file in the file system.
- int fl_mkdir (const char *f, int mode)

  Cross-platform function to create a directory with a UTF-8 encoded name.
- unsigned int fl_nonspacing (unsigned int ucs)

  Returns true if the Unicode character `ucs` is non-spacing.
- int fl_open (const char *fname, int oflags,...)

  Cross-platform function to open files with a UTF-8 encoded name.
- int fl_open_ext (const char *fname, int binary, int oflags,...)

  Cross-platform function to open files with a UTF-8 encoded name.
- int fl_putenv (const char *var)

  Cross-platform function to write environment variables with a UTF-8 encoded name or value.
- int fl_rename (const char *f, const char *n)

  Cross-platform function to rename a filesystem object using UTF-8 encoded names.
- int fl_rmdir (const char *f)

  Cross-platform function to remove a directory with a UTF-8 encoded name.
- int fl_stat (const char *f, struct stat *b)

  Cross-platform function to stat() a file using a UTF-8 encoded name or value.
- int fl_system (const char *cmd)

  Cross-platform function to run a system command with a UTF-8 encoded string.
- int fl_tolower (unsigned int ucs)

  Returns the Unicode lower case value of `ucs`.
- int fl_toupper (unsigned int ucs)

  Returns the Unicode upper case value of `ucs`.
- unsigned fl_ucs_to_Utf16 (const unsigned ucs, unsigned short *dst, const unsigned dstlen)

  Convert a single 32-bit Unicode codepoint into an array of 16-bit characters.
- int fl_unlink (const char *fname)

  Cross-platform function to unlink() (that is, delete) a file using a UTF-8 encoded filename.
- char * fl_utf2mbcs (const char *s)

  Converts UTF-8 string `s` to a local multi-byte character string.
- const char * fl_utf8_clean (const char *str, int *len)

  Test and optionally re-encode a UTF-8 string (character sequence).
- int **fl_utf8_is_continuation** (char byte)

  Evaluates true if 'byte' is in the middle of a UTF-8 character (0x80..0xBF).
- const char * fl_utf8_next_composed_char (const char *from, const char *end)

  Returns pointer to beginning of character after given location in UTF8 string accounting for emoji sequences.
- const char * fl_utf8_previous_composed_char (const char *from, const char *begin)

  Returns pointer to beginning of character before given location in UTF8 string accounting for emoji sequences.
- const char * fl_utf8back (const char *p, const char *start, const char *end)

  Move `p` backward until it points to the start of a UTF-8 character.
- int fl_utf8bytes (unsigned ucs)

  Returns number of bytes that utf8encode() will use to encode the character `ucs`.
- unsigned fl_utf8decode (const char *p, const char *end, int *len)

  Decode a single UTF-8 encoded character starting at p.
- int fl_utf8encode (unsigned ucs, char *buf)

  Write the UTF-8 encoding of ucs into buf and return the number of bytes written.
- unsigned fl_utf8from_mb (char *dst, unsigned dstlen, const char *src, unsigned srclen)

  Convert a filename from the locale-specific multibyte encoding used by Windows to UTF-8 as used by FLTK.

- unsigned fl_utf8froma (char *dst, unsigned dstlen, const char *src, unsigned srclen)

  Convert an ISO-8859-1 (ie normal c-string) byte stream to UTF-8.
- unsigned fl_utf8fromwc (char *dst, unsigned dstlen, const wchar_t *src, unsigned srclen)

  Turn "wide characters" as returned by some system calls (especially on Windows) into UTF-8.
- const char * fl_utf8fwd (const char *p, const char *start, const char *end)

  Move `p` forward until it points to the start of a UTF-8 character.
- int fl_utf8len (char c)

  Returns the byte length of the UTF-8 sequence, or -1 if `c` is not a valid leading byte.
- int fl_utf8len1 (char c)

  Returns the byte length of the UTF-8 sequence with first byte `c`, or 1 if `c` is not valid.
- int fl_utf8locale ()

  Return true if the "locale" seems to indicate that UTF-8 encoding is used.
- int fl_utf8strlen (const char *text, int len)

  Return the length in bytes of a UTF-8 string.
- int fl_utf8test (const char *src, unsigned srclen)

  Examines the first `srclen` bytes in `src` and returns a verdict on whether it is UTF-8 or not.
- unsigned fl_utf8to_mb (const char *src, unsigned srclen, char *dst, unsigned dstlen)

  Convert the UTF-8 used by FLTK to the locale-specific encoding used for filenames (and sometimes used for data in files).
- unsigned fl_utf8toa (const char *src, unsigned srclen, char *dst, unsigned dstlen)

  Convert a UTF-8 sequence into an array of 1-byte characters.
- unsigned fl_utf8toUtf16 (const char *src, unsigned srclen, unsigned short *dst, unsigned dstlen)

  Convert a UTF-8 sequence into an array of 16-bit characters.
- unsigned fl_utf8towc (const char *src, unsigned srclen, wchar_t *dst, unsigned dstlen)

  Converts a UTF-8 string into a wide character string.
- int fl_utf_nb_char (const unsigned char *buf, int len)

  Returns the number of Unicode chars in the UTF-8 string.
- int fl_utf_strcasecmp (const char *s1, const char *s2)

  UTF-8 aware strcasecmp - converts to Unicode and tests.
- int fl_utf_strncasecmp (const char *s1, const char *s2, int n)

  UTF-8 aware strncasecmp - converts to lower case Unicode and tests.
- int fl_utf_tolower (const unsigned char *str, int len, char *buf)

  Converts the string `str` to its lower case equivalent into buf.
- int fl_utf_toupper (const unsigned char *str, int len, char *buf)

  Converts the string `str` to its upper case equivalent into buf.
- int fl_wcwidth (const char *src)

  extended wrapper around fl_wcwidth_(unsigned int ucs) function.
- int fl_wcwidth_ (unsigned int ucs)

  Wrapper to adapt Markus Kuhn's implementation of wcwidth() for FLTK.

14.179.1 Detailed Description

header for Unicode and UTF-8 character handling

14.180 fl_utf8.h

Go to the documentation of this file.

```
/*
 * Unicode to UTF-8 conversion functions for the Fast Light Tool Kit (FLTK).
 *
 * Original author: Jean-Marc Lienher ( http://oksid.ch )
 * Copyright 2000-2010 by O'ksi'D.
 *
 * Copyright 2016-2026 by Bill Spitzak and others.
 *
 * This library is free software. Distribution and use rights are outlined in
 * the file "COPYING" which should have been included with this file.  If this
 * file is missing or damaged, see the license at:
 *
 *     https://www.fltk.org/COPYING.php
 *
 * Please see the following page on how to report bugs and issues:
 *
 *     https://www.fltk.org/bugs.php
 */

/* Merged in some functionality from the fltk-2 version. IMM.
 * The following code is an attempt to merge the functions incorporated in FLTK2
 * with the functions provided in OksiD's fltk-1.1.6-utf8 port
 */

#ifndef _HAVE_FL_UTF8_HDR_
#define _HAVE_FL_UTF8_HDR_

#include "Fl_Export.H"
#include "fl_types.h"
#include <stdio.h>      /* FILE *fl_fopen() */
#include <sys/stat.h>   /* struct stat */

#ifdef __cplusplus
extern "C" {
#endif

/* F2: comes from FLTK2 */
/* OD: comes from OksiD */

FL_EXPORT int fl_utf8bytes(unsigned ucs);

/* OD: returns the byte length of the first UTF-8 char sequence (returns -1 if not valid) */
FL_EXPORT int fl_utf8len(char c);

/* OD: returns the byte length of the first UTF-8 char sequence (returns +1 if not valid) */
FL_EXPORT int fl_utf8len1(char c);

/* OD: returns the byte length of a UTF-8 text */
FL_EXPORT int fl_utf8strlen(const char *text, int len);

/* OD: returns the number of Unicode chars in the UTF-8 string */
FL_EXPORT int fl_utf_nb_char(const unsigned char *buf, int len);

/* F2: Convert the next UTF-8 char-sequence into a Unicode value (and say how many bytes were used) */
FL_EXPORT unsigned fl_utf8decode(const char* p, const char* end, int* len);

/* F2: Encode a Unicode value into a UTF-8 sequence, return the number of bytes used */
FL_EXPORT int fl_utf8encode(unsigned ucs, char* buf);

/* F2: Move forward to the next valid UTF-8 sequence start betwen start and end */
FL_EXPORT const char* fl_utf8fwd(const char* p, const char* start, const char* end);

/* F2: Move backward to the previous valid UTF-8 sequence start */
FL_EXPORT const char* fl_utf8back(const char* p, const char* start, const char* end);

/* XX: Convert a single 32-bit Unicode value into UTF16 */
FL_EXPORT unsigned fl_ucs_to_Utf16(const unsigned ucs, unsigned short *dst, const unsigned dstlen);

/* F2: Convert a UTF-8 string into UTF16 */
FL_EXPORT unsigned fl_utf8toUtf16(const char* src, unsigned srclen, unsigned short* dst, unsigned
    dstlen);

/* F2: Convert a UTF-8 string into a wide character string - makes UTF16 on win32, "UCS4" elsewhere */
FL_EXPORT unsigned fl_utf8towc(const char *src, unsigned srclen, wchar_t *dst, unsigned dstlen);

/* F2: Convert a wide character string to UTF-8 - takes in UTF16 on win32, "UCS4" elsewhere */
FL_EXPORT unsigned fl_utf8fromwc(char *dst, unsigned dstlen, const wchar_t *src, unsigned srclen);

/* F2: Convert a UTF-8 string into ASCII, eliding untranslatable glyphs */
FL_EXPORT unsigned fl_utf8toa (const char *src, unsigned srclen, char *dst, unsigned dstlen);

/* F2: Convert 8859-1 string to UTF-8 */
FL_EXPORT unsigned fl_utf8froma (char *dst, unsigned dstlen, const char *src, unsigned srclen);
```

```

/* F2: Returns true if the current O/S locale is UTF-8 */
FL_EXPORT int fl_utf8locale(void);

/* Test if byte is in the middle of a UTF-8 character */
FL_EXPORT int fl_utf8_is_continuation(char byte);

/* F2: Examine the first len characters of src, to determine if the input text is UTF-8 or not
 * NOTE: The value returned is not simply boolean - it contains information about the probable
 * type of the src text. */
FL_EXPORT int fl_utf8test(const char *src, unsigned len);

/* XX: return width of "raw" ucs character in columns.
 * for internal use only */
FL_EXPORT int fl_wcwidth_(unsigned int ucs);

/* XX: return width of UTF-8 character string in columns.
 * NOTE: this may also do C1 control character (0x80 to 0x9f) to CP1252 mapping,
 * depending on original build options */
FL_EXPORT int fl_wcwidth(const char *src);

/* OD: Return true if the character is non-spacing */
FL_EXPORT unsigned int fl_nonspacing(unsigned int ucs);

/* F2: Convert UTF-8 to a local multi-byte encoding - mainly for win32? */
FL_EXPORT unsigned fl_utf8to_mb(const char *src, unsigned srclen, char *dst, unsigned dstlen);
/* OD: Convert UTF-8 to a local multi-byte encoding */
FL_EXPORT char* fl_utf2mbcs(const char *src);

/* F2: Convert a local multi-byte encoding to UTF-8 - mainly for win32? */
FL_EXPORT unsigned fl_utf8from_mb(char *dst, unsigned dstlen, const char *src, unsigned srclen);

/******************************************************************************/
#ifdef _WIN32
/* these two Windows-only functions are kept for API compatibility */
/* OD: Attempt to convert the UTF-8 string to the current locale */
FL_EXPORT char *fl_utf8_to_locale(const char *s, int len, unsigned int codepage);

/* OD: Attempt to convert a string in the current locale to UTF-8 */
FL_EXPORT char *fl_locale_to_utf8(const char *s, int len, unsigned int codepage);
#endif /* _WIN32 */

/*****************************************************************************
 * The following functions are intended to provide portable, UTF-8 aware
 * versions of standard functions
 */

/* OD: UTF-8 aware strncasecmp - converts to lower case Unicode and tests */
FL_EXPORT int fl_utf_strncasecmp(const char *s1, const char *s2, int n);

/* OD: UTF-8 aware strcasecmp - converts to Unicode and tests */
FL_EXPORT int fl_utf_strcasecmp(const char *s1, const char *s2);

/* OD: return the Unicode lower case value of ucs */
FL_EXPORT int fl_tolower(unsigned int ucs);

/* OD: return the Unicode upper case value of ucs */
FL_EXPORT int fl_toupper(unsigned int ucs);

/* OD: converts the UTF-8 string to the lower case equivalent */
FL_EXPORT int fl_utf_tolower(const unsigned char *str, int len, char *buf);

/* OD: converts the UTF-8 string to the upper case equivalent */
FL_EXPORT int fl_utf_toupper(const unsigned char *str, int len, char *buf);

/* OD: Portable UTF-8 aware chmod wrapper */
FL_EXPORT int fl_chmod(const char* f, int mode);

/* OD: Portable UTF-8 aware access wrapper */
FL_EXPORT int fl_access(const char* f, int mode);

/* OD: Portable UTF-8 aware stat wrapper */
FL_EXPORT int fl_stat(const char *path, struct stat *buffer);

/* OD: Portable UTF-8 aware getcwd wrapper */
FL_EXPORT char *fl_getcwd(char *buf, int len);

/* Portable UTF-8 aware chdir wrapper */
FL_EXPORT int fl_chdir(const char *path);

/* OD: Portable UTF-8 aware fopen wrapper */
FL_EXPORT FILE *fl_fopen(const char *f, const char *mode);

/* OD: Portable UTF-8 aware system wrapper */
FL_EXPORT int fl_system(const char* f);

/* OD: Portable UTF-8 aware execvp wrapper */
```

```
FL_EXPORT int fl_execvp(const char *file, char *const *argv);

/* OD: Portable UTF-8 aware open wrapper */
FL_EXPORT int fl_open(const char *fname, int oflags, ...);

FL_EXPORT int fl_open_ext(const char *fname, int binary, int oflags, ...);

/* Portable wrapper around unix-style close() function */
FL_EXPORT int fl_close_fd(int fd);

/* OD: Portable UTF-8 aware unlink wrapper */
FL_EXPORT int fl_unlink(const char *fname);

/* OD: Portable UTF-8 aware rmdir wrapper */
FL_EXPORT int fl_rmdir(const char *f);

/* OD: Portable UTF-8 aware getenv wrapper */
FL_EXPORT char* fl_getenv(const char *name);

/* Portable UTF-8 aware putenv wrapper */
FL_EXPORT int fl_putenv(const char *var);

/* OD: Portable UTF-8 aware mkdir wrapper */
FL_EXPORT int fl_mkdir(const char* f, int mode);

/* OD: Portable UTF-8 aware rename wrapper */
FL_EXPORT int fl_rename(const char* f, const char *t);


/* OD: Given a full pathname, this will create the directory path needed to hold the file named */
FL_EXPORT void fl_make_path_for_file( const char *path );

/* OD: recursively create a path in the file system */
FL_EXPORT char fl_make_path( const char *path );

FL_EXPORT const char *fl_utf8_next_composed_char(const char *from, const char *end);

FL_EXPORT const char *fl_utf8_previous_composed_char(const char *from, const char *begin);


/* Clean UTF-8 string by re-encoding if necessary - see src/fl_utf8.cxx */
FL_EXPORT const char* fl_utf8_clean(const char* str, int* n);

/* ************************************************************************** */

#ifdef __cplusplus
}
#endif /* __cplusplus */


#endif /* _HAVE_FL_UTF8_HDR_ */
```

14.181 Fl_Valuator.H

```
//
// Valuator header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2026 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Valuator widget . */

#ifndef Fl_Valuator_H
#define Fl_Valuator_H

#ifndef Fl_Widget_H
#include "Fl_Widget.H"
#endif

#include <string>

// shared type() values for classes that work in both directions:
#define FL_VERTICAL             0
#define FL_HORIZONTAL           1
```

```
00032
00049 class FL_EXPORT Fl_Valuator : public Fl_Widget {
00050
00051   double value_;
00052   double previous_value_;
00053   double min, max; // truncates to this range *after* rounding
00054   double A; int B; // rounds to multiples of A/B, or no rounding if A is zero
00055
00056 protected:
00058   int horizontal() const {return type()& FL_HORIZONTAL;}
00059   Fl_Valuator(int X, int Y, int W, int H, const char* L);
00060
00062   double previous_value() const {return previous_value_;}
00064   void handle_push() {previous_value_ = value_;}
00065   double softclamp(double);
00066   void handle_drag(double newvalue);
00067   void handle_release(); // use drag() value
00068   virtual void value_damage(); // cause damage() due to value() changing
00070   void set_value(double v) {value_ = v;}
00071
00072 public:
00073
00075   ~Fl_Valuator() override { }
00077     void bounds(double a, double b) {min=a; max=b;}
00079     double minimum() const {return min;}
00081     void minimum(double a) {min = a;}
00083     double maximum() const {return max;}
00085     void maximum(double a) {max = a;}
00106   void range(double a, double b) {min = a; max = b;}
00108   void step(int a) {A = a; B = 1;}
00110   void step(double a, int b) {A = a; B = b;}
00111   void step(double s);
00123   double step() const {return A/B;}
00124   void precision(int digits);
00125
00127   double value() const {return value_;}
00128   int value(double);
00129
00130   virtual int format(char*);
00131   virtual std::string format_str();
00132   double round(double); // round to nearest multiple of step
00133   double clamp(double); // keep in range
00134   double increment(double, int); // add n*step to value
00135 };
00136
00137 #endif
```

14.182 Fl_Value_Input.H

```
00001 //
00002 // Value input header file for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 1998-2022 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file.  If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 //     https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 //     https://www.fltk.org/bugs.php
00015 //
00016
00017 /* \file
00018    Fl_Value_Input widget . */
00019
00020 #ifndef Fl_Value_Input_H
00021 #define Fl_Value_Input_H
00022
00023 #include "Fl_Valuator.H"
00024 #include "Fl_Input.H"
00025
00056 class FL_EXPORT Fl_Value_Input : public Fl_Valuator {
00057 public:
00058   /* This is the encapsulated Fl_input attribute to which
00059   this class delegates the value font, color and shortcut */
00060   Fl_Input input;
00061 private:
00062   char soft_;
00063   static void input_cb(Fl_Widget*,void*);
00064   void value_damage() override; // cause damage() due to value() changing
00065 public:
00066   int handle(int) override;
```

```
protected:
  void draw() override;
public:
  void resize(int,int,int,int) override;
  Fl_Value_Input(int x,int y,int w,int h,const char *l=0);
  ~Fl_Value_Input();

  void soft(char s) {soft_ = s;}
  char soft() const {return soft_;}
  int shortcut() const {return input.shortcut();}
  void shortcut(int s) {input.shortcut(s);}

  Fl_Font textfont() const {return input.textfont();}
  void textfont(Fl_Font s) {input.textfont(s);}
  Fl_Fontsize textsize() const {return input.textsize();}
  void textsize(Fl_Fontsize s) {input.textsize(s);}
  Fl_Color textcolor() const {return input.textcolor();}
  void textcolor(Fl_Color n) {input.textcolor(n);}
  Fl_Color cursor_color() const {return input.cursor_color();}
  void cursor_color(Fl_Color n) {input.cursor_color(n);}

};

#endif
```

14.183 Fl_Value_Output.H

```
//
// Value output header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2022 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Value_Output widget . */

#ifndef Fl_Value_Output_H
#define Fl_Value_Output_H

#ifndef Fl_Valuator_H
#include "Fl_Valuator.H"
#endif

class FL_EXPORT Fl_Value_Output : public Fl_Valuator {
  Fl_Font textfont_;
  Fl_Fontsize textsize_;
  uchar soft_;
  Fl_Color textcolor_;

protected:
  void draw() override;

public:
  int handle(int) override;
  Fl_Value_Output(int x,int y,int w,int h,const char *l=0);

  void soft(uchar s) {soft_ = s;}
  uchar soft() const {return soft_;}

  Fl_Font textfont() const {return textfont_;}
  void textfont(Fl_Font s) {textfont_ = s;}
  Fl_Fontsize textsize() const {return textsize_;}
  void textsize(Fl_Fontsize s) {textsize_ = s;}
  Fl_Color textcolor() const {return textcolor_;}
  void textcolor(Fl_Color s) {textcolor_ = s;}
};

#endif
```

14.184 Fl_Value_Slider.H

```
//
```

```
// Value Slider header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2022 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_Value_Slider widget . */

#ifndef Fl_Value_Slider_H
#define Fl_Value_Slider_H

#include "Fl_Slider.H"

class FL_EXPORT Fl_Value_Slider : public Fl_Slider {
  Fl_Font textfont_;
  Fl_Fontsize textsize_;
  Fl_Color textcolor_;
  short value_width_;
  short value_height_;

protected:
  void draw() override;

public:
  int handle(int) override;
  Fl_Value_Slider(int x, int y, int w, int h, const char *l = 0);

  Fl_Font textfont() const { return textfont_; }

  void textfont(Fl_Font s) { textfont_ = s; }

  Fl_Fontsize textsize() const { return textsize_; }

  void textsize(Fl_Fontsize s) { textsize_ = s; }

  Fl_Color textcolor() const { return textcolor_; }

  void textcolor(Fl_Color s) { textcolor_ = s; }

  void value_width(int s) {
    if (s > w() - 10)
      s = w() - 10;
    if (s < 10)
      s = 10;
    value_width_ = (short)s;
  }

  int value_width() const { return (value_width_); }

  void value_height(int s) {
    if (s > h() - 10)
      s = h() - 10;
    if (s < 10)
      s = 10;
    value_height_ = (short)s;
  }
  int value_height() const { return (value_height_); }
};

#endif
```

14.185 Fl_Widget.H File Reference

Fl_Widget and Fl_Label classes.

`#include "Fl.H"`

Classes

- class Fl_Callback_User_Data

  A class prototype that allows for additional data in callbacks.

- struct Fl_Label

  This struct stores all information for a text or mixed graphics label.
- class Fl_Widget

  Fl_Widget is the base class for all widgets in FLTK.

Macros

- #define FL_RESERVED_TYPE 100

  Reserved type numbers (necessary for my cheapo RTTI) start here.

Typedefs

- typedef void() **Fl_Callback**(Fl_Widget *, void *)

  Default callback type definition for all fltk widgets (by far the most used)
- typedef void() **Fl_Callback0**(Fl_Widget *)

  One parameter callback type definition passing only the widget.
- typedef void() **Fl_Callback1**(Fl_Widget *, long)

  Callback type definition passing the widget and a long data value.
- typedef Fl_Callback * **Fl_Callback_p**

  Default callback type pointer definition for all fltk widgets.

14.185.1 Detailed Description

Fl_Widget and Fl_Label classes.

14.185.2 Macro Definition Documentation

14.185.2.1 FL_RESERVED_TYPE

```
#define FL_RESERVED_TYPE 100
```

Reserved type numbers (necessary for my cheapo RTTI) start here.
Grep the header files for "RESERVED_TYPE" to find the next available number.

14.186 Fl_Widget.H

Go to the documentation of this file.

```
00001 //
00002 // Widget header file for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 1998-2026 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file.  If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 //     https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 //     https://www.fltk.org/bugs.php
00015 //
00016
00022 #ifndef Fl_Widget_H
00023 #define Fl_Widget_H
00024
00025 #include "Fl.H"
00026
00027 class Fl_Widget;
00028 class Fl_Window;
00029 class Fl_Group;
00030 class Fl_Image;
00031
00033 typedef void (Fl_Callback )(Fl_Widget*, void*);
00035 typedef Fl_Callback* Fl_Callback_p; // needed for BORLAND
00037 typedef void (Fl_Callback0)(Fl_Widget*);
00039 typedef void (Fl_Callback1)(Fl_Widget*, long);
00040
```

```
struct FL_EXPORT Fl_Label {
  const char* value;
  Fl_Image* image;
  Fl_Image* deimage;
  Fl_Font font;
  Fl_Fontsize size;
  Fl_Color color;
  Fl_Align align_;
  uchar type;
  signed char h_margin_;
  signed char v_margin_;
  uchar spacing;

  void draw(int,int,int,int, Fl_Align) const ;
  void measure(int &w, int &h) const ;
};


class FL_EXPORT Fl_Callback_User_Data {
protected:
  Fl_Callback_User_Data() {}
public:
  virtual ~Fl_Callback_User_Data() { }
};


class FL_EXPORT Fl_Widget {
  friend class Fl_Group;

  Fl_Group* parent_;
  Fl_Callback* callback_;
  void* user_data_;
  int x_,y_,w_,h_;
  Fl_Label label_;
  unsigned int flags_;
  Fl_Color color_;
  Fl_Color selection_color_;
  uchar type_;
  uchar damage_;
  uchar box_;
  uchar when_;

  const char *tooltip_;

  Fl_Widget(const Fl_Widget &);
  Fl_Widget& operator=(const Fl_Widget &);

protected:

  Fl_Widget(int x, int y, int w, int h, const char *label=0L);

  void x(int v) {x_ = v;}
  void y(int v) {y_ = v;}
  void w(int v) {w_ = v;}
  void h(int v) {h_ = v;}
  unsigned int flags() const {return flags_;}
  void set_flag(unsigned int c) {flags_ |= c;}
  void clear_flag(unsigned int c) {flags_ &= ~c;}
  enum {
        INACTIVE        = 1«0,
        INVISIBLE       = 1«1,
        OUTPUT          = 1«2,
        NOBORDER        = 1«3,
        FORCE_POSITION  = 1«4,
        NON_MODAL       = 1«5,
        SHORTCUT_LABEL  = 1«6,
        CHANGED         = 1«7,
        OVERRIDE        = 1«8,
        VISIBLE_FOCUS   = 1«9,
        COPIED_LABEL    = 1«10,
        CLIP_CHILDREN   = 1«11,
        MENU_WINDOW     = 1«12,
        TOOLTIP_WINDOW  = 1«13,
        MODAL           = 1«14,
        NO_OVERLAY      = 1«15,
        GROUP_RELATIVE  = 1«16,
        COPIED_TOOLTIP  = 1«17,
        FULLSCREEN      = 1«18,
        MAC_USE_ACCENTS_MENU = 1«19,
        NEEDS_KEYBOARD  = 1«20,
        IMAGE_BOUND     = 1«21,
        DEIMAGE_BOUND   = 1«22,
        AUTO_DELETE_USER_DATA = 1«23,
        MAXIMIZED       = 1«24,
        POPUP           = 1«25,
        // Note to devs: add new FLTK core flags above this line (up to 1«28).

```

```
        // Three more flags, reserved for user code

        USERFLAG3       = 1«29,
        USERFLAG2       = 1«30,
        USERFLAG1       = 1«31
  };
  void draw_box() const;
  void draw_box(Fl_Boxtype t, Fl_Color c) const;
  void draw_box(Fl_Boxtype t, int x,int y,int w,int h, Fl_Color c) const;
  void draw_backdrop() const;

  void draw_focus() const {
    draw_focus(box(), x(), y(), w(), h(), color());
  }

  void draw_focus(Fl_Boxtype t, int X, int Y, int W, int H) const {
    draw_focus(t, X, Y, W, H, color());

  }
  // See documentation in Fl_Widget.cxx
  void draw_focus(Fl_Boxtype t, int x, int y, int w, int h, Fl_Color bg) const;

  void draw_label() const;
  void draw_label(int, int, int, int) const;

public:

  virtual ~Fl_Widget();

  virtual void draw() = 0;

  virtual int handle(int event);

  int send(int event);

  int is_label_copied() const {return ((flags_ & COPIED_LABEL) ? 1 : 0);}

  void needs_keyboard(bool needs) {
    if (needs) set_flag(NEEDS_KEYBOARD);
    else clear_flag(NEEDS_KEYBOARD);
  }

  bool needs_keyboard() const {
    return (flags_ & NEEDS_KEYBOARD);
  }

  Fl_Group* parent() const {return parent_;}

  void parent(Fl_Group* p) {parent_ = p;} // for hacks only, use Fl_Group::add()

  uchar type() const {return type_;}

  void type(uchar t) {type_ = t;}

  int x() const {return x_;}

  int y() const {return y_;}

  int w() const {return w_;}

  int h() const {return h_;}

  virtual void resize(int x, int y, int w, int h);

  int damage_resize(int,int,int,int);

  void position(int X,int Y) {resize(X,Y,w_,h_);}

  void size(int W,int H) {resize(x_,y_,W,H);}

  Fl_Align align() const {return label_.align_;}

  void align(Fl_Align alignment) {label_.align_ = alignment;}

  Fl_Boxtype box() const {return (Fl_Boxtype)box_;}

  void box(Fl_Boxtype new_box) {box_ = (uchar)new_box;}

  Fl_Color color() const {return color_;}

  void color(Fl_Color bg) {color_ = bg;}

  Fl_Color selection_color() const {return selection_color_;}

  void selection_color(Fl_Color a) {selection_color_ = a;}

  void color(Fl_Color bg, Fl_Color sel) {color_ = bg; selection_color_ = sel;}
```

```
00505
00510   const char* label() const {return label_.value;}
00511
00523   void label(const char* text);
00524
00535   void copy_label(const char *new_label);
00536
00540   void label(Fl_Labeltype a, const char* b) {label_.type = (uchar)a; label_.value = b;}
00541
00546   Fl_Labeltype labeltype() const {return (Fl_Labeltype)label_.type;}
00547
00556   void labeltype(Fl_Labeltype a) {label_.type = (uchar)a;}
00557
00562   Fl_Color labelcolor() const {return label_.color;}
00563
00568   void labelcolor(Fl_Color c) {label_.color=c;}
00569
00577   Fl_Font labelfont() const {return label_.font;}
00578
00586   void labelfont(Fl_Font f) {label_.font=f;}
00587
00592   Fl_Fontsize labelsize() const {return label_.size;}
00593
00598   void labelsize(Fl_Fontsize pix) {label_.size=pix;}
00599
00603   Fl_Image* image() {return label_.image;}
00604
00608   const Fl_Image* image() const {return label_.image;}
00609
00625   void image(Fl_Image* img);
00626
00631   void image(Fl_Image& img);
00632
00648   void bind_image(Fl_Image* img);
00649
00654   void bind_image(int f) { if (f) set_flag(IMAGE_BOUND); else clear_flag(IMAGE_BOUND); }
00655
00661   int image_bound() const {return ((flags_ & IMAGE_BOUND) ? 1 : 0);}
00662
00666   Fl_Image* deimage() {return label_.deimage;}
00667
00671   const Fl_Image* deimage() const {return label_.deimage;}
00672
00679   void deimage(Fl_Image* img);
00680
00685   void deimage(Fl_Image& img);
00686
00693   void bind_deimage(Fl_Image* img);
00694
00700   int deimage_bound() const {return ((flags_ & DEIMAGE_BOUND) ? 1 : 0);}
00701
00706   void bind_deimage(int f) { if (f) set_flag(DEIMAGE_BOUND); else clear_flag(DEIMAGE_BOUND); }
00707
00712   void label_image_spacing(int gap) { label_.spacing = (uchar)gap; }
00713
00717   int label_image_spacing() { return label_.spacing; }
00718
00722   void horizontal_label_margin(int px) { label_.h_margin_ = (signed char)px; }
00723
00727   int horizontal_label_margin() { return label_.h_margin_; }
00728
00732   void vertical_label_margin(int px) { label_.v_margin_ = (signed char)px; }
00733
00737   int vertical_label_margin() { return label_.v_margin_; }
00738
00743   const char *tooltip() const {return tooltip_;}
00744
00745   void tooltip(const char *text);               // see Fl_Tooltip
00746   void copy_tooltip(const char *text);          // see Fl_Tooltip
00747
00752   Fl_Callback_p callback() const {return callback_;}
00753
00759   void callback(Fl_Callback* cb, void* p) {
00760     callback_ = cb;
00761     user_data(p);
00762   }
00763
00771   void callback(Fl_Callback* cb, Fl_Callback_User_Data* p, bool auto_free) {
00772     callback_ = cb;
00773     user_data(p, auto_free);
00774   }
00775
00780   void callback(Fl_Callback* cb) {callback_ = cb;}
00781
00786   void callback(Fl_Callback0* cb) {
00787     callback_ = (Fl_Callback*)(fl_intptr_t)(cb);
00788   }
```

```
00789
00795   void callback(Fl_Callback1* cb, long p = 0) {
00796     callback_ = (Fl_Callback*)(fl_intptr_t)(cb);
00797     user_data((void*)(fl_intptr_t)p);
00798   }
00799
00804   void* user_data() const {return user_data_;}
00805
00807   void user_data(void* v);
00808
00810   void user_data(Fl_Callback_User_Data* v, bool auto_free);
00811
00823   long argument() const {return (long)(fl_intptr_t)user_data_;}
00824
00829   void argument(long v) {user_data((void*)(fl_intptr_t)v);}
00830
00839   Fl_When when() const {return (Fl_When)when_;}
00840
00878   void when(uchar i) {when_ = i;}
00879
00884   unsigned int visible() const {return !(flags_&INVISIBLE);}
00885
00890   int visible_r() const;
00891
00909   virtual void show();
00910
00914   virtual void hide();
00915
00920   void set_visible() {flags_ &= ~INVISIBLE;}
00921
00926   void clear_visible() {flags_ |= INVISIBLE;}
00927
00932   unsigned int active() const {return !(flags_&INACTIVE);}
00933
00938   int active_r() const;
00939
00945   void activate();
00946
00961   void deactivate();
00962
00971   unsigned int output() const {return (flags_&OUTPUT);}
00972
00976   void set_output() {flags_ |= OUTPUT;}
00977
00981   void clear_output() {flags_ &= ~OUTPUT;}
00982
00988   unsigned int takesevents() const {return !(flags_&(INACTIVE|INVISIBLE|OUTPUT));}
00989
01008   unsigned int changed() const {return flags_ & CHANGED;}
01009
01013   void set_changed() {flags_ |= CHANGED;}
01014
01018   void clear_changed() {flags_ &= ~CHANGED;}
01019
01024   void clear_active() {flags_ |= INACTIVE;}
01025
01030   void set_active() {flags_ &= ~INACTIVE;}
01031
01039   int take_focus();
01040
01047   void set_visible_focus() { flags_ |= VISIBLE_FOCUS; }
01048
01053   void clear_visible_focus() { flags_ &= ~VISIBLE_FOCUS; }
01054
01059   void visible_focus(int v) { if (v) set_visible_focus(); else clear_visible_focus(); }
01060
01065   unsigned int visible_focus() const { return flags_ & VISIBLE_FOCUS; }
01066
01087   static void default_callback(Fl_Widget *widget, void *data);
01088
01101   void do_callback(Fl_Callback_Reason reason=FL_REASON_UNKNOWN) {do_callback(this, user_data_,
      reason);}
01102
01110   void do_callback(Fl_Widget *widget, long arg, Fl_Callback_Reason reason=FL_REASON_UNKNOWN) {
01111     do_callback(widget, (void*)(fl_intptr_t)arg, reason);
01112   }
01113
01114   void do_callback(Fl_Widget *widget, void *arg = 0, Fl_Callback_Reason reason=FL_REASON_UNKNOWN);
01115
01116   /* Internal use only. */
01117   int test_shortcut();
01118   /* Internal use only. */
01119   static unsigned int label_shortcut(const char *t);
01120   /* Internal use only. */
01121   static int test_shortcut(const char*, const bool require_alt = false);
01122   /* Internal use only. */
01123   void _set_fullscreen() {flags_ |= FULLSCREEN;}
```

```
  void _clear_fullscreen() {flags_ &= ~FULLSCREEN;}

  int contains(const Fl_Widget *w) const ;

  int inside(const Fl_Widget *wgt) const {return wgt ? wgt->contains(this) : 0;}

  void redraw();

  void redraw_label();

  uchar damage() const {return damage_;}

  void clear_damage(uchar c = 0) {damage_ = c;}

  void damage(uchar c);

  void damage(uchar c, int x, int y, int w, int h);

  void draw_label(int, int, int, int, Fl_Align) const;

  void measure_label(int& ww, int& hh) const {label_.measure(ww, hh);}

  Fl_Window* window() const ;
  Fl_Window* top_window() const;
  Fl_Window* top_window_offset(int& xoff, int& yoff) const;

  virtual Fl_Group* as_group() { return NULL; }
  virtual Fl_Group const* as_group() const { return NULL; }

  virtual Fl_Window* as_window() { return 0; }
  virtual Fl_Window const* as_window() const { return NULL; }

  virtual class Fl_Gl_Window* as_gl_window() { return NULL; }
  virtual class Fl_Gl_Window const* as_gl_window() const { return NULL; }

  int use_accents_menu() { return flags() & MAC_USE_ACCENTS_MENU; }

  void shortcut_label(int value) {
    if (value)
      set_flag(SHORTCUT_LABEL);
    else
      clear_flag(SHORTCUT_LABEL);
  }

  int shortcut_label() const { return flags_ & SHORTCUT_LABEL; }
};

#define FL_RESERVED_TYPE 100

#endif
```

14.187 Fl_Widget_Surface.H

```
//
// Drivers code for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2016 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef Fl_Widget_Surface_h
#define Fl_Widget_Surface_h

#include <FL/Fl_Device.H>
#include <FL/Fl_Window.H>

class FL_EXPORT Fl_Widget_Surface : public Fl_Surface_Device {
private:
  void traverse(Fl_Widget *widget); // finds subwindows of widget and prints them
protected:
  int x_offset;
  int y_offset;
  Fl_Widget_Surface(Fl_Graphics_Driver *d);
public:
  virtual void translate(int x, int y);
  virtual void untranslate();
```

```
  void draw(Fl_Widget* widget, int delta_x = 0, int delta_y = 0);
  void draw_decorated_window(Fl_Window *win, int x_offset = 0, int y_offset = 0);
  void print_window_part(Fl_Window *win, int x, int y, int w, int h, int delta_x = 0, int delta_y = 0);
  virtual int printable_rect(int *w, int *h);
  virtual void origin(int x, int y);
  virtual void origin(int *x, int *y);
};

#endif /* Fl_Widget_Surface_h */
```

14.188 Fl_Widget_Tracker.H File Reference

Track widget deletion.

```
#include <FL/fl_config.h>
#include <FL/Fl_Export.H>
```

Classes

- class Fl_Widget_Tracker

  This class should be used to control safe widget deletion.

Namespaces

- namespace Fl

  Fl is the FLTK global namespace containing state information and global methods for the current application.

Functions

- void Fl::clear_widget_pointer (Fl_Widget const *w)

  Clears a widget pointer in the watch list.
- void Fl::delete_widget (Fl_Widget *w)

  Schedules a widget for deletion at the next call to the event loop.
- void Fl::do_widget_deletion ()

  Deletes widgets previously scheduled for deletion.
- void Fl::release_widget_pointer (Fl_Widget *&w)

  Releases a widget pointer from the watch list.
- void Fl::watch_widget_pointer (Fl_Widget *&w)

  Adds a widget pointer to the widget watch list.

14.188.1 Detailed Description

Track widget deletion.

14.189 Fl_Widget_Tracker.H

Go to the documentation of this file.

```
//
// Widget Tracker header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 2025 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

```

```
#ifndef Fl_Widget_Tracker_H
#define Fl_Widget_Tracker_H

#include <FL/fl_config.h> // build configuration
#include <FL/Fl_Export.H>

class Fl_Widget;


class FL_EXPORT Fl_Widget_Tracker {

  Fl_Widget* wp_;

public:

  Fl_Widget_Tracker(Fl_Widget *wi);
  // Rule of five. Note that we *can* implement these when we refactor widget
  // tracking with a C++11 map or unordered_map, for example.
  Fl_Widget_Tracker(const Fl_Widget_Tracker&) = delete;
  Fl_Widget_Tracker(Fl_Widget_Tracker&&) = delete;
  Fl_Widget_Tracker& operator=(const Fl_Widget_Tracker&) = delete;
  Fl_Widget_Tracker& operator=(Fl_Widget_Tracker&&) = delete;
  ~Fl_Widget_Tracker();

  void clear() { wp_ = nullptr; }

  Fl_Widget *widget() { return wp_; }

  int deleted() {return wp_ == nullptr;}

  int exists() { return wp_ != nullptr; }

};

namespace Fl {

// Widget deletion:
FL_EXPORT extern void delete_widget(Fl_Widget *w);
FL_EXPORT extern void do_widget_deletion();
FL_EXPORT extern void watch_widget_pointer(Fl_Widget *&w);
FL_EXPORT extern void release_widget_pointer(Fl_Widget *&w);
FL_EXPORT extern void clear_widget_pointer(Fl_Widget const *w);

} // namespace Fl

#endif // !Fl_Widget_Tracker_H
```

14.190 Fl_Window.H File Reference

Fl_Window widget .

```
#include <FL/Fl.H>
#include <FL/Fl_Group.H>
#include <FL/Fl_Bitmap.H>
```

Classes

- class Fl_Window

  This widget produces an actual window.

Macros

- #define **FL_DOUBLE_WINDOW** 0xF1

  double window type id
- #define **FL_WINDOW** 0xF0

  window type id: all subclasses have type() >= this

14.190.1 Detailed Description

Fl_Window widget .

14.191 Fl_Window.H

Go to the documentation of this file.

```
//
// Window header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2025 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef Fl_Window_H
#define Fl_Window_H

#include <FL/Fl.H>
#include <FL/Fl_Group.H>
#include <FL/Fl_Bitmap.H>

#define FL_WINDOW 0xF0
#define FL_DOUBLE_WINDOW 0xF1

class Fl_X;
class Fl_Window_Driver;
class Fl_RGB_Image;
class Fl_Double_Window;

class FL_EXPORT Fl_Window : public Fl_Group {
  friend class Fl_X;
  friend class Fl_Window_Driver;
private:
  static char *default_xclass_;
  static char show_next_window_iconic_; // 1 means create next window in iconic form

  int no_fullscreen_x;
  int no_fullscreen_y;
  int no_fullscreen_w;
  int no_fullscreen_h;
  int fullscreen_screen_top;
  int fullscreen_screen_bottom;
  int fullscreen_screen_left;
  int fullscreen_screen_right;

  // TODO: it would make sense to merge the use of Fl_X and Fl_Window_Driver, maybe simply by
  // TODO: deriving Fl_Window_Driver from Fl_X. However, there are a lot of historic kludges
  // TODO: for some platforms around Fl_X.
  Fl_X *flx_; // points at the system-specific stuff, but exists only after the window is mapped
  Fl_Window_Driver *pWindowDriver; // points at the system-specific stuff at window creation time

  const char* iconlabel_;
  char* xclass_;

  // private size_range stuff:
  int minw_, minh_, maxw_, maxh_;
  int dw_, dh_, aspect_;
  uchar size_range_set_;        // true (1) if size_range() has been set or calculated

  // cursor stuff
  Fl_Cursor cursor_default;

  void _Fl_Window(); // constructor innards

  // unimplemented copy ctor and assignment operator
  Fl_Window(const Fl_Window&);
  Fl_Window& operator=(const Fl_Window&);

  void is_maximized_(bool b);

protected:

  static Fl_Window *current_;
  void draw() override;

public:

  virtual void flush();

protected:

```

```
  void force_position(int force) {
    if (force) set_flag(FORCE_POSITION);
    else clear_flag(FORCE_POSITION);
  }

  int force_position() const { return ((flags() & FORCE_POSITION)?1:0); }

  void free_icons();

  void default_size_range();  // calculate size_range() if not set explicitly
  int is_resizable();         // calculate size_range() and return whether this is resizable

public:

    Fl_Window(int w, int h, const char *title = 0);
    Fl_Window(int x, int y, int w, int h, const char *title = 0);
    virtual ~Fl_Window();

  int handle(int) override;

  void resize(int X,int Y,int W,int H) override;
  void border(int b);
  void clear_border()   {set_flag(NOBORDER);}
  unsigned int border() const   {return !(flags() & NOBORDER);}
  void set_override()   {set_flag(NOBORDER|OVERRIDE);}
  unsigned int override() const  { return flags()&OVERRIDE; }
  void set_modal()      {set_flag(MODAL);}
  unsigned int modal() const    {return flags() & MODAL;}
  void set_non_modal()  {set_flag(NON_MODAL);}
  unsigned int non_modal() const {return flags() & (NON_MODAL|MODAL);}

  void clear_modal_states() {clear_flag(NON_MODAL | MODAL);}

  void set_menu_window()        {set_flag(MENU_WINDOW);}

  unsigned int menu_window() const {return flags() & MENU_WINDOW;}

  void set_tooltip_window()     { set_flag(TOOLTIP_WINDOW);
                                  clear_flag(MENU_WINDOW); }
  unsigned int tooltip_window() const {return flags() & TOOLTIP_WINDOW;}

  void hotspot(int x, int y, int offscreen = 0);
  void hotspot(const Fl_Widget*, int offscreen = 0);
  void hotspot(const Fl_Widget& p, int offscreen = 0) {hotspot(&p,offscreen);}

  void free_position()  { clear_flag(FORCE_POSITION); }

  void size_range(int minw, int minh, int maxw=0, int maxh=0, int dw=0, int dh=0, int aspect=0);

  uchar get_size_range(int *minw, int *minh, int *maxw=NULL, int *maxh=NULL, int *dw=NULL, int
*dh=NULL, int *aspect=NULL) const;

  const char* label() const     {return Fl_Widget::label();}
  const char* iconlabel() const {return iconlabel_;}
  void label(const char*);
  void iconlabel(const char*);
  void label(const char* label, const char* iconlabel); // platform dependent
  void copy_label(const char* a);

  static void default_xclass(const char*);
  static const char *default_xclass();
  const char* xclass() const;
  void xclass(const char* c);

  static void default_icon(const Fl_RGB_Image*);
  static void default_icons(const Fl_RGB_Image*[], int);
  void icon(const Fl_RGB_Image*);
  void icons(const Fl_RGB_Image*[], int);

#if defined(_WIN32) || defined(FL_DOXYGEN)
  typedef struct HICON__* HICON;
  // These 2 member functions break the driver model but are kept for back compatibility.
  // They are implemented in Fl_win32.cxx

  static void default_icons(HICON big_icon, HICON small_icon);

  void icons(HICON big_icon, HICON small_icon);
#endif // defined(_WIN32) || defined(FL_DOXYGEN)

  /* for legacy compatibility */
  const void* icon() const;
  void icon(const void * ic);

  int shown() const { return flx_ != 0; }

  void show() override;
  void hide() override;
```

```
  void show(int argc, char **argv);

  // Enables synchronous show(), docs in Fl_Window.cxx
  void wait_for_expose();

  void fullscreen();
  void fullscreen_off();
  void fullscreen_off(int X,int Y,int W,int H);
  unsigned int fullscreen_active() const { return flags() & FULLSCREEN; }
  void fullscreen_screens(int top, int bottom, int left, int right);

  void maximize();
  void un_maximize();
  unsigned int maximize_active() const { return flags() & MAXIMIZED; }
public:
  void iconize();

  int x_root() const ;
  int y_root() const ;

 static Fl_Window *current();
  void make_current();

  void cursor(Fl_Cursor);
  void cursor(const Fl_RGB_Image*, int, int);
  void default_cursor(Fl_Cursor);

  /* for legacy compatibility */
  void cursor(Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE);
  void default_cursor(Fl_Cursor c, Fl_Color, Fl_Color=FL_WHITE);

  static void default_callback(Fl_Window*, void* v);

  int decorated_w() const;

  int decorated_h() const;

  // Note: Doxygen docs in Fl_Widget.H to avoid redundancy.
  Fl_Window* as_window() override { return this; }
  Fl_Window const* as_window() const override { return this; }

  virtual class Fl_Overlay_Window *as_overlay_window() {return 0L; }

  virtual class Fl_Double_Window *as_double_window() {return 0L;}

  void shape(const Fl_Image* img);
  void shape(const Fl_Image& b);
  const Fl_Image *shape() const;
  void draw_backdrop();
  int screen_num() const;
  void screen_num(int screen_num);
  static bool is_a_rescale();
  fl_uintptr_t os_id() const;

  static void show_next_window_iconic(char stat) {
    show_next_window_iconic_ = stat ? 1 : 0;
  }

  static char show_next_window_iconic() {
    return show_next_window_iconic_;
  }

  void allow_expand_outside_parent();

};

#endif
```

14.192 Fl_Wizard.H

```
//
// Fl_Wizard widget definitions.
//
// Copyright 1999-2010 by Easy Software Products.
// Copyright 2011-2020 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
```

```
//

/* \file
   Fl_Wizard widget . */

//
// Include necessary header files...
//

#ifndef _Fl_Wizard_H_
#define _Fl_Wizard_H_

#include <FL/Fl_Group.H>

class FL_EXPORT Fl_Wizard : public Fl_Group {

  Fl_Widget *value_;

protected:

  void draw() override;

public:

  Fl_Wizard(int, int, int, int, const char * = 0);

  void next();
  void prev();
  Fl_Widget *value();
  void value(Fl_Widget *);
  void value(int);
};

#endif // !_Fl_Wizard_H_
```

14.193 Fl_XBM_Image.H

```
//
// XBM image header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file
   Fl_XBM_Image class . */

#ifndef Fl_XBM_Image_H
#define Fl_XBM_Image_H
#  include "Fl_Bitmap.H"

class FL_EXPORT Fl_XBM_Image : public Fl_Bitmap {

  public:

  Fl_XBM_Image(const char* filename);
};

#endif // !Fl_XBM_Image_H
```

14.194 Fl_XPM_Image.H

```
//
// XPM image header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
```

```
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* \file

   Fl_XPM_Image class . */

#ifndef Fl_XPM_Image_H
#define Fl_XPM_Image_H
#  include "Fl_Pixmap.H"

class FL_EXPORT Fl_XPM_Image : public Fl_Pixmap {

  public:

  Fl_XPM_Image(const char* filename);
};

#endif // !Fl_XPM_Image
```

14.195 forms.H

```
//
// Forms emulation header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2023 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef __FORMS_H__
#define __FORMS_H__

#include "Fl.H"
#include "Fl_Group.H"
#include "Fl_Window.H"
#include "fl_draw.H"

typedef Fl_Widget FL_OBJECT;
typedef Fl_Window FL_FORM;

// Random constants & symbols defined by forms.h file:

#ifndef NULL
#define NULL 0
#endif
#ifndef FALSE
#define FALSE 0
#define TRUE 1
#endif

#define FL_ON           1
#define FL_OK           1
#define FL_VALID        1
#define FL_PREEMPT      1
#define FL_AUTO         2
#define FL_WHEN_NEEDED  FL_AUTO
#define FL_OFF          0
#define FL_NONE         0
#define FL_CANCEL       0
#define FL_INVALID      0
#define FL_IGNORE       -1
// #define FL_CLOSE     -2 // this variable is never used in FLTK Forms. It is removed
                           // because it conflicts with the window FL_CLOSE event

#define FL_LCOL         FL_BLACK
#define FL_COL1         FL_GRAY
#define FL_MCOL         FL_LIGHT1
#define FL_LEFT_BCOL    FL_LIGHT3 // 53 is better match
#define FL_TOP_BCOL     FL_LIGHT2 // 51
#define FL_BOTTOM_BCOL  FL_DARK2  // 40
#define FL_RIGHT_BCOL   FL_DARK3  // 36
#define FL_INACTIVE     FL_INACTIVE_COLOR
#define FL_INACTIVE_COL FL_INACTIVE_COLOR
#define FL_FREE_COL1    FL_FREE_COLOR
```

```
#define FL_FREE_COL2    ((Fl_Color)(FL_FREE_COLOR+1))
#define FL_FREE_COL3    ((Fl_Color)(FL_FREE_COLOR+2))
#define FL_FREE_COL4    ((Fl_Color)(FL_FREE_COLOR+3))
#define FL_FREE_COL5    ((Fl_Color)(FL_FREE_COLOR+4))
#define FL_FREE_COL6    ((Fl_Color)(FL_FREE_COLOR+5))
#define FL_FREE_COL7    ((Fl_Color)(FL_FREE_COLOR+6))
#define FL_FREE_COL8    ((Fl_Color)(FL_FREE_COLOR+7))
#define FL_FREE_COL9    ((Fl_Color)(FL_FREE_COLOR+8))
#define FL_FREE_COL10   ((Fl_Color)(FL_FREE_COLOR+9))
#define FL_FREE_COL11   ((Fl_Color)(FL_FREE_COLOR+10))
#define FL_FREE_COL12   ((Fl_Color)(FL_FREE_COLOR+11))
#define FL_FREE_COL13   ((Fl_Color)(FL_FREE_COLOR+12))
#define FL_FREE_COL14   ((Fl_Color)(FL_FREE_COLOR+13))
#define FL_FREE_COL15   ((Fl_Color)(FL_FREE_COLOR+14))
#define FL_FREE_COL16   ((Fl_Color)(FL_FREE_COLOR+15))
#define FL_TOMATO       ((Fl_Color)(131))
#define FL_INDIANRED    ((Fl_Color)(164))
#define FL_SLATEBLUE    ((Fl_Color)(195))
#define FL_DARKGOLD     ((Fl_Color)(84))
#define FL_PALEGREEN    ((Fl_Color)(157))
#define FL_ORCHID       ((Fl_Color)(203))
#define FL_DARKCYAN     ((Fl_Color)(189))
#define FL_DARKTOMATO   ((Fl_Color)(113))
#define FL_WHEAT        ((Fl_Color)(174))

#define FL_ALIGN_BESIDE FL_ALIGN_INSIDE

#define FL_PUP_TOGGLE   2 // FL_MENU_TOGGLE
#define FL_PUP_INACTIVE 1 // FL_MENU_INACTIVE
#define FL_NO_FRAME     FL_NO_BOX
#define FL_ROUNDED3D_UPBOX      FL_ROUND_UP_BOX
#define FL_ROUNDED3D_DOWNBOX    FL_ROUND_DOWN_BOX
#define FL_OVAL3D_UPBOX         FL_ROUND_UP_BOX
#define FL_OVAL3D_DOWNBOX       FL_ROUND_DOWN_BOX

#define FL_MBUTTON1     1
#define FL_LEFTMOUSE    1
#define FL_MBUTTON2     2
#define FL_MIDDLEMOUSE  2
#define FL_MBUTTON3     3
#define FL_RIGHTMOUSE   3
#define FL_MBUTTON4     4
#define FL_MBUTTON5     5

#define FL_INVALID_STYLE 255
#define FL_NORMAL_STYLE FL_HELVETICA
#define FL_BOLD_STYLE   FL_HELVETICA_BOLD
#define FL_ITALIC_STYLE FL_HELVETICA_ITALIC
#define FL_BOLDITALIC_STYLE FL_HELVETICA_BOLD_ITALIC
#define FL_FIXED_STYLE  FL_COURIER
#define FL_FIXEDBOLD_STYLE FL_COURIER_BOLD
#define FL_FIXEDITALIC_STYLE FL_COURIER_ITALIC
#define FL_FIXEDBOLDITALIC_STYLE FL_COURIER_BOLD_ITALIC
#define FL_TIMES_STYLE  FL_TIMES
#define FL_TIMESBOLD_STYLE FL_TIMES_BOLD
#define FL_TIMESITALIC_STYLE FL_TIMES_ITALIC
#define FL_TIMESBOLDITALIC_STYLE FL_TIMES_BOLD_ITALIC

// hacks to change the labeltype() when passed to fl_set_object_lstyle():
#define FL_SHADOW_STYLE         (FL_SHADOW_LABEL«8)
#define FL_ENGRAVED_STYLE       (FL_ENGRAVED_LABEL«8)
#define FL_EMBOSSED_STYLE       (FL_EMBOSSED_LABEL«0)

// size values are different from XForms, match older Forms:
#define FL_TINY_SIZE    8
#define FL_SMALL_SIZE   11 // 10
//#define FL_NORMAL_SIZE        14 // 12
#define FL_MEDIUM_SIZE  18 // 14
#define FL_LARGE_SIZE   24 // 18
#define FL_HUGE_SIZE    32 // 24
#define FL_DEFAULT_SIZE FL_SMALL_SIZE
#define FL_TINY_FONT    FL_TINY_SIZE
#define FL_SMALL_FONT   FL_SMALL_SIZE
#define FL_NORMAL_FONT  FL_NORMAL_SIZE
#define FL_MEDIUM_FONT  FL_MEDIUM_SIZE
#define FL_LARGE_FONT   FL_LARGE_SIZE
#define FL_HUGE_FONT    FL_HUGE_SIZE
#define FL_NORMAL_FONT1 FL_SMALL_FONT
#define FL_NORMAL_FONT2 FL_NORMAL_FONT
#define FL_DEFAULT_FONT FL_SMALL_FONT

#define FL_RETURN_END_CHANGED   FL_WHEN_RELEASE
#define FL_RETURN_CHANGED       FL_WHEN_CHANGED
#define FL_RETURN_END           FL_WHEN_RELEASE_ALWAYS
#define FL_RETURN_ALWAYS        (FL_WHEN_CHANGED|FL_WHEN_NOT_CHANGED)

#define FL_BOUND_WIDTH  3
```

```

typedef int FL_Coord;
typedef int FL_COLOR;

// fltk interaction:

#define FL_CMD_OPT void
extern FL_EXPORT void fl_initialize(int*, char*[], const char*, FL_CMD_OPT*, int);
inline void fl_finish() {}

typedef void (*FL_IO_CALLBACK) (FL_SOCKET, void*);
inline void fl_add_io_callback(int fd, short w, FL_IO_CALLBACK cb, void* v) {
  Fl::add_fd(fd, w, cb, v);}
inline void fl_remove_io_callback(int fd, short, FL_IO_CALLBACK) {
  Fl::remove_fd(fd);} // removes all the callbacks!

// type of callback is different and no "id" number is returned:
inline void fl_add_timeout(long msec, void (*cb)(void*), void* v) {
  Fl::add_timeout(msec*.001, cb, v);}
inline void fl_remove_timeout(int) {}

// type of callback is different!
inline void fl_set_idle_callback(void (*cb)()) {Fl::add_idle(cb);}

FL_EXPORT Fl_Widget* fl_do_forms(void);
FL_EXPORT Fl_Widget* fl_check_forms();
inline Fl_Widget* fl_do_only_forms(void) {return fl_do_forms();}
inline Fl_Widget* fl_check_only_forms(void) {return fl_check_forms();}

// because of new redraw behavior, these are no-ops:
inline void fl_freeze_object(Fl_Widget*) {}
inline void fl_unfreeze_object(Fl_Widget*) {}
inline void fl_freeze_form(Fl_Window*) {}
inline void fl_unfreeze_form(Fl_Window*) {}
inline void fl_freeze_all_forms() {}
inline void fl_unfreeze_all_forms() {}

inline void fl_set_focus_object(Fl_Window*, Fl_Widget* o) {Fl::focus(o);}
inline void fl_reset_focus_object(Fl_Widget* o) {Fl::focus(o);}
#define fl_set_object_focus fl_set_focus_object

// void fl_set_form_atclose(Fl_Window*w,int (*cb)(Fl_Window*,void*),void* v)
// void fl_set_atclose(int (*cb)(Fl_Window*,void*),void*)
// fl_set_form_atactivate/atdeactivate not implemented!

// Fl_Widget:

inline void fl_set_object_boxtype(Fl_Widget* o, Fl_Boxtype a) {o->box(a);}
inline void fl_set_object_lsize(Fl_Widget* o,int s) {o->labelsize(s);}

/* forms lib font indexes must be byte sized - extract correct byte from style word */
inline void fl_set_object_lstyle(Fl_Widget* o,int a) {
  o->labelfont((Fl_Font)(a&0xff)); o->labeltype((Fl_Labeltype)(a»8));}
inline void fl_set_object_lcol(Fl_Widget* o, Fl_Color a) {o->labelcolor(a);}
#define fl_set_object_lcolor  fl_set_object_lcol
inline void fl_set_object_lalign(Fl_Widget* o, Fl_Align a) {o->align(a);}
#define fl_set_object_align fl_set_object_lalign
inline void fl_set_object_color(Fl_Widget* o,Fl_Color a,Fl_Color b) {o->color(a,b);}
inline void fl_set_object_label(Fl_Widget* o, const char* a) {o->label(a); o->redraw();}
inline void fl_set_object_position(Fl_Widget*o,int x,int y) {o->position(x,y);}
inline void fl_set_object_size(Fl_Widget* o, int w, int h) {o->size(w,h);}
inline void fl_set_object_geometry(Fl_Widget* o,int x,int y,int w,int h) {o->resize(x,y,w,h);}

inline void fl_get_object_geometry(Fl_Widget* o,int*x,int*y,int*w,int*h) {
  *x = o->x(); *y = o->y(); *w = o->w(); *h = o->h();}
inline void fl_get_object_position(Fl_Widget* o,int*x,int*y) {
  *x = o->x(); *y = o->y();}

typedef void (*Forms_CB)(Fl_Widget*, long);
inline void fl_set_object_callback(Fl_Widget*o,Forms_CB c,long a) {o->callback(c,a);}
#define fl_set_call_back      fl_set_object_callback
inline void fl_call_object_callback(Fl_Widget* o) {o->do_callback();}
inline void fl_trigger_object(Fl_Widget* o) {o->do_callback();}
inline void fl_set_object_return(Fl_Widget* o, int v) {
  o->when((uchar)(v|FL_WHEN_RELEASE));}

inline void fl_redraw_object(Fl_Widget* o) {o->redraw();}
inline void fl_show_object(Fl_Widget* o) {o->show();}
inline void fl_hide_object(Fl_Widget* o) {o->hide();}
inline void fl_free_object(Fl_Widget* x) {delete x;}
inline void fl_delete_object(Fl_Widget* o) {o->parent()->remove(*o);}
inline void fl_activate_object(Fl_Widget* o) {o->activate();}
inline void fl_deactivate_object(Fl_Widget* o) {o->deactivate();}

inline void fl_add_object(Fl_Window* f, Fl_Widget* x) {f->add(x);}
inline void fl_insert_object(Fl_Widget* o, Fl_Widget* b) {b->parent()->insert(*o,b);}

```

```
inline Fl_Window* FL_ObjWin(Fl_Widget* o) {return o->window();}

// things that appered in the demos a lot that I don't emulate, but
// I did not want to edit out of all the demos...

inline int fl_get_border_width() {return 3;}
inline void fl_set_border_width(int) {}
inline void fl_set_object_dblbuffer(Fl_Widget*, int) {}
inline void fl_set_form_dblbuffer(Fl_Window*, int) {}

// Fl_Window:

inline void fl_free_form(Fl_Window* x) {delete x;}
inline void fl_redraw_form(Fl_Window* f) {f->redraw();}

inline Fl_Window* fl_bgn_form(Fl_Boxtype b,int w,int h) {
  Fl_Window* g = new Fl_Window(w,h,0);
  g->box(b);
  return g;
}
FL_EXPORT void fl_end_form();
inline void fl_addto_form(Fl_Window* f) {f->begin();}
inline Fl_Group* fl_bgn_group() {return new Fl_Group(0,0,0,0,0);}
inline void fl_end_group() {Fl_Group::current()->forms_end();}
inline void fl_addto_group(Fl_Widget* o) {((Fl_Group* )o)->begin();}
#define resizebox _ddfdesign_kludge()

inline void fl_scale_form(Fl_Window* f, double x, double y) {
  f->resizable(f); f->size(int(f->w()*x),int(f->h()*y));}
inline void fl_set_form_position(Fl_Window* f,int x,int y) {f->position(x,y);}
inline void fl_set_form_size(Fl_Window* f, int w, int h) {f->size(w,h);}
inline void fl_set_form_geometry(Fl_Window* f,int x,int y,int w,int h) {
  f->resize(x,y,w,h);}
#define fl_set_initial_placement fl_set_form_geometry
inline void fl_adjust_form_size(Fl_Window*) {}

FL_EXPORT void fl_show_form(Fl_Window* f,int p,int b,const char* n);
enum {  // "p" argument values:
  FL_PLACE_FREE = 0,     // make resizable
  FL_PLACE_MOUSE = 1,    // mouse centered on form
  FL_PLACE_CENTER = 2,   // center of the screen
  FL_PLACE_POSITION = 4,// fixed position, resizable
  FL_PLACE_SIZE = 8,     // fixed size, normal fltk behavior
  FL_PLACE_GEOMETRY =16,// fixed size and position
  FL_PLACE_ASPECT = 32, // keep aspect ratio (ignored)
  FL_PLACE_FULLSCREEN=64,// fill screen
  FL_PLACE_HOTSPOT = 128,// enables hotspot
  FL_PLACE_ICONIC = 256,// iconic (ignored)
  FL_FREE_SIZE=(1«14), // force resizable
  FL_FIX_SIZE =(1«15)  // force off resizable
};
#define FL_PLACE_FREE_CENTER (FL_PLACE_CENTER|FL_FREE_SIZE)
#define FL_PLACE_CENTERFREE  (FL_PLACE_CENTER|FL_FREE_SIZE)
enum {  // "b" argument values:
  FL_NOBORDER = 0,
  FL_FULLBORDER,
  FL_TRANSIENT
//FL_MODAL = (1«8)      // not implemented yet in Forms
};
inline void fl_set_form_hotspot(Fl_Window* w,int x,int y) {w->hotspot(x,y);}
inline void fl_set_form_hotobject(Fl_Window* w, Fl_Widget* o) {w->hotspot(o);}
extern FL_EXPORT char fl_flip;  // in forms.C
inline void fl_flip_yorigin() {fl_flip = 1;}

#define fl_prepare_form_window fl_show_form
inline void fl_show_form_window(Fl_Window*) {}

inline void fl_raise_form(Fl_Window* f) {f->show();}

inline void fl_hide_form(Fl_Window* f) {f->hide();}
inline void fl_pop_form(Fl_Window* f) {f->show();}

extern FL_EXPORT char fl_modal_next; // in forms.C
inline void fl_activate_all_forms() {}
inline void fl_deactivate_all_forms() {fl_modal_next = 1;}
inline void fl_deactivate_form(Fl_Window*w) {w->deactivate();}
inline void fl_activate_form(Fl_Window*w) {w->activate();}

inline void fl_set_form_title(Fl_Window* f, const char* s) {f->label(s);}
inline void fl_title_form(Fl_Window* f, const char* s) {f->label(s);}

typedef void (*Forms_FormCB)(Fl_Widget*);
inline void fl_set_form_callback(Fl_Window* f,Forms_FormCB c) {f->callback(c);}
#define fl_set_form_call_back fl_set_form_callback

inline void fl_init() {}
FL_EXPORT void fl_set_graphics_mode(int,int);
```

```

inline int fl_form_is_visible(Fl_Window* f) {return f->visible();}

inline int fl_mouse_button() {return Fl::event_button();}
#define fl_mousebutton fl_mouse_button

#define fl_free       free
#define fl_malloc     malloc
#define fl_calloc     calloc
#define fl_realloc    realloc

// Drawing functions.  Only usable inside an Fl_Free object?

inline void fl_drw_box(Fl_Boxtype b,int x,int y,int w,int h,Fl_Color bgc,int=3) {
    fl_draw_box(b,x,y,w,h,bgc);}
inline void fl_drw_frame(Fl_Boxtype b,int x,int y,int w,int h,Fl_Color bgc,int=3) {
    fl_draw_box(b,x,y,w,h,bgc);}

inline void fl_drw_text(Fl_Align align, int x, int y, int w, int h,
                  Fl_Color fgcolor, int size, Fl_Font style,
                  const char* s) {
  fl_font(style,size);
  fl_color(fgcolor);
  fl_draw(s,x,y,w,h,align);
}

// this does not work except for CENTER...
inline void fl_drw_text_beside(Fl_Align align, int x, int y, int w, int h,
                  Fl_Color fgcolor, int size, Fl_Font style,
                  const char* s) {
  fl_font(style,size);
  fl_color(fgcolor);
  fl_draw(s,x,y,w,h,align);
}

inline void fl_set_font_name(Fl_Font n,const char* s) {Fl::set_font(n,s);}

inline void fl_mapcolor(Fl_Color c, uchar r, uchar g, uchar b) {Fl::set_color(c,r,g,b);}

#define fl_set_clipping(x,y,w,h) fl_push_clip(x,y,w,h)
#define fl_unset_clipping() fl_pop_clip()

// Forms classes:

inline Fl_Widget* fl_add_new(Fl_Widget* p) {return p;}
inline Fl_Widget* fl_add_new(uchar t,Fl_Widget* p) {p->type(t); return p;}

#define forms_constructor(type,name) \
inline type* name(uchar t,int x,int y,int w,int h,const char* l) { \
 return (type*)(fl_add_new(t, new type(x,y,w,h,l)));}
#define forms_constructort(type,name) \
inline type* name(uchar t,int x,int y,int w,int h,const char* l) { \
 return (type*)(fl_add_new(new type(t,x,y,w,h,l)));}
#define forms_constructorb(type,name) \
inline type* name(Fl_Boxtype t,int x,int y,int w,int h,const char* l) { \
 return (type*)(fl_add_new(new type(t,x,y,w,h,l)));}

#include "Fl_FormsBitmap.H"
#define FL_NORMAL_BITMAP FL_NO_BOX
forms_constructorb(Fl_FormsBitmap, fl_add_bitmap)
inline void fl_set_bitmap_data(Fl_Widget* o, int w, int h, const uchar* b) {
    ((Fl_FormsBitmap*)o)->set(w,h,b);
}

#include "Fl_FormsPixmap.H"
#define FL_NORMAL_PIXMAP FL_NO_BOX
forms_constructorb(Fl_FormsPixmap, fl_add_pixmap)
inline void fl_set_pixmap_data(Fl_Widget* o, char*const* b) {
    ((Fl_FormsPixmap*)o)->set(b);
}
//inline void fl_set_pixmap_file(Fl_Widget*, const char*);
inline void fl_set_pixmap_align(Fl_Widget* o,Fl_Align a,int,int) {o->align(a);}
//inline void fl_set_pixmap_colorcloseness(int, int, int);

#include "Fl_Box.H"
forms_constructorb(Fl_Box, fl_add_box)

#include "Fl_Browser.H"
forms_constructor(Fl_Browser, fl_add_browser)

inline void fl_clear_browser(Fl_Widget* o) {
    ((Fl_Browser*)o)->clear();}
inline void fl_add_browser_line(Fl_Widget* o, const char* s) {
    ((Fl_Browser*)o)->add(s);}
inline void fl_addto_browser(Fl_Widget* o, const char* s) {
    ((Fl_Browser*)o)->add(s);} /* should also scroll to bottom */
//inline void fl_addto_browser_chars(Fl_Widget*, const char*)
```

```
//#define fl_append_browser fl_addto_browser_chars
inline void fl_insert_browser_line(Fl_Widget* o, int n, const char* s) {
    ((Fl_Browser*)o)->insert(n,s);}
inline void fl_delete_browser_line(Fl_Widget* o, int n) {
    ((Fl_Browser*)o)->remove(n);}
inline void fl_replace_browser_line(Fl_Widget* o, int n, const char* s) {
    ((Fl_Browser*)o)->replace(n,s);}
inline char* fl_get_browser_line(Fl_Widget* o, int n) {
    return (char*)(((Fl_Browser*)o)->text(n));}
inline int fl_load_browser(Fl_Widget* o, const char* f) {
    return ((Fl_Browser*)o)->load(f);}
inline void fl_select_browser_line(Fl_Widget* o, int n) {
    ((Fl_Browser*)o)->select(n,1);}
inline void fl_deselect_browser_line(Fl_Widget* o, int n) {
    ((Fl_Browser*)o)->select(n,0);}
inline void fl_deselect_browser(Fl_Widget* o) {
    ((Fl_Browser*)o)->deselect();}
inline int fl_isselected_browser_line(Fl_Widget* o, int n) {
    return ((Fl_Browser*)o)->selected(n);}
inline int fl_get_browser_topline(Fl_Widget* o) {
    return ((Fl_Browser*)o)->topline();}
inline int fl_get_browser(Fl_Widget* o) {
    return ((Fl_Browser*)o)->value();}
inline int fl_get_browser_maxline(Fl_Widget* o) {
    return ((Fl_Browser*)o)->size();}
//linline int fl_get_browser_screenlines(Fl_Widget*);
inline void fl_set_browser_topline(Fl_Widget* o, int n) {
    ((Fl_Browser*)o)->topline(n);}
inline void fl_set_browser_fontsize(Fl_Widget* o, int s) {
    ((Fl_Browser*)o)->textsize(s);}
inline void fl_set_browser_fontstyle(Fl_Widget* o, Fl_Font s) {
    ((Fl_Browser*)o)->textfont(s);}
inline void fl_set_browser_specialkey(Fl_Widget* o, char c) {
    ((Fl_Browser*)o)->format_char(c);}
//inline void fl_set_browser_vscrollbar(Fl_Widget*, int);
//inline void fl_set_browser_hscrollbar(Fl_Widget*, int);
//inline void fl_set_browser_leftslider(Fl_Widget*, int);
//#define fl_set_browser_leftscrollbar fl_set_browser_leftslider
//inline void fl_set_browser_line_selectable(Fl_Widget*, int, int);
//inline void fl_get_browser_dimension(Fl_Widget*,int*,int*,int*,int*);
//inline void fl_set_browser_dblclick_callback(Fl_Widget*,FL_CALLBACKPTR,long);
//inline void fl_set_browser_xoffset(Fl_Widget*, FL_Coord);
//inline void fl_set_browser_scrollbarsize(Fl_Widget*, int, int);
inline void fl_setdisplayed_browser_line(Fl_Widget* o, int n, int i) {
    ((Fl_Browser*)o)->display(n,i);}
inline int fl_isdisplayed_browser_line(Fl_Widget* o, int n) {
    return ((Fl_Browser*)o)->displayed(n);}

#include "Fl_Button.H"

#define FL_NORMAL_BUTTON        0
#define FL_TOUCH_BUTTON         4
#define FL_INOUT_BUTTON         5
#define FL_RETURN_BUTTON        6
#define FL_HIDDEN_RET_BUTTON    7
#define FL_PUSH_BUTTON          FL_TOGGLE_BUTTON
#define FL_MENU_BUTTON          9

FL_EXPORT Fl_Button* fl_add_button(uchar t,int x,int y,int w,int h,const char* l);
inline int fl_get_button(Fl_Widget* b) {return ((Fl_Button*)b)->value();}
inline void fl_set_button(Fl_Widget* b, int v) {((Fl_Button*)b)->value(v);}
inline int fl_get_button_numb(Fl_Widget*) {return Fl::event_button();}
inline void fl_set_button_shortcut(Fl_Widget* b, const char* s,int=0) {
    ((Fl_Button*)b)->shortcut(s);}
//#define fl_set_object_shortcut(b,s) fl_set_button_shortcut(b,s)

#include "Fl_Light_Button.H"
forms_constructor(Fl_Light_Button, fl_add_lightbutton)

#include "Fl_Round_Button.H"
forms_constructor(Fl_Round_Button, fl_add_roundbutton)
forms_constructor(Fl_Round_Button, fl_add_round3dbutton)

#include "Fl_Check_Button.H"
forms_constructor(Fl_Check_Button, fl_add_checkbutton)

inline Fl_Widget* fl_add_bitmapbutton(int t,int x,int y,int w,int h,const char* l) {
    Fl_Widget* o = fl_add_button((uchar)t,x,y,w,h,l);
    return o;
}

inline void fl_set_bitmapbutton_data(Fl_Widget *o, int a, int b, uchar *c) {
  (new Fl_Bitmap(c, a, b))->label(o); // does not delete old Fl_Bitmap!
}

inline Fl_Widget* fl_add_pixmapbutton(int t,int x,int y,int w,int h,const char* l) {
    Fl_Widget* o = fl_add_button((uchar)t,x,y,w,h,l);
```

```
    return o;
}

inline void fl_set_pixmapbutton_data(Fl_Widget *o, const char *const *c) {
  (new Fl_Pixmap(c))->label(o); // does not delete old Fl_Pixmap!
}

// Fl_Canvas object not yet implemented!

#include "Fl_Chart.H"

forms_constructor(Fl_Chart, fl_add_chart)
inline void fl_clear_chart(Fl_Widget* o) {
  ((Fl_Chart*)o)->clear();}
inline void fl_add_chart_value(Fl_Widget* o,double v,const char* s,uchar c){
  ((Fl_Chart*)o)->add(v,s,c);}
inline void fl_insert_chart_value(Fl_Widget* o, int i, double v, const char* s, uchar c) {
  ((Fl_Chart*)o)->insert(i,v,s,c);}
inline void fl_replace_chart_value(Fl_Widget* o, int i, double v, const char* s, uchar c) {
  ((Fl_Chart*)o)->replace(i,v,s,c);}
inline void fl_set_chart_bounds(Fl_Widget* o, double a, double b) {
  ((Fl_Chart*)o)->bounds(a,b);}
inline void fl_set_chart_maxnumb(Fl_Widget* o, int v) {
  ((Fl_Chart*)o)->maxsize(v);}
inline void fl_set_chart_autosize(Fl_Widget* o, int v) {
  ((Fl_Chart*)o)->autosize((uchar)v);
}
inline void fl_set_chart_lstyle(Fl_Widget* o, Fl_Font v) {
  ((Fl_Chart*)o)->textfont(v);}
inline void fl_set_chart_lsize(Fl_Widget* o, int v) {
  ((Fl_Chart*)o)->textsize(v);}
inline void fl_set_chart_lcolor(Fl_Widget* o, Fl_Color v) {
  ((Fl_Chart*)o)->textcolor(v);}
#define fl_set_chart_lcol   fl_set_chart_lcolor

#include "Fl_Choice.H"

#define FL_NORMAL_CHOICE        0
#define FL_NORMAL_CHOICE2       0
#define FL_DROPLIST_CHOICE      0

forms_constructor(Fl_Choice, fl_add_choice)
inline void fl_clear_choice(Fl_Widget* o) {
    ((Fl_Choice*)o)->clear();}
inline void fl_addto_choice(Fl_Widget* o, const char* s) {
    ((Fl_Choice*)o)->add(s);}
inline void fl_replace_choice(Fl_Widget* o, int i, const char* s) {
    ((Fl_Choice*)o)->replace(i-1,s);}
inline void fl_delete_choice(Fl_Widget* o, int i) {
    ((Fl_Choice*)o)->remove(i-1);}
inline void fl_set_choice(Fl_Widget* o, int i) {
    ((Fl_Choice*)o)->value(i-1);}
// inline void fl_set_choice_text(Fl_Widget*, const char*);
inline int fl_get_choice(Fl_Widget* o) {
    return ((Fl_Choice*)o)->value()+1;}
// inline const char* fl_get_choice_item_text(Fl_Widget*, int);
// inline int fl_get_choice_maxitems(Fl_Widget*);
inline const char* fl_get_choice_text(Fl_Widget* o) {
    return ((Fl_Choice*)o)->text();}
inline void fl_set_choice_fontsize(Fl_Widget* o, int x) {
    ((Fl_Choice*)o)->textsize(x);}
inline void fl_set_choice_fontstyle(Fl_Widget* o, Fl_Font x) {
    ((Fl_Choice*)o)->textfont(x);}
// inline void fl_set_choice_item_mode(Fl_Widget*, int, unsigned);
// inline void fl_set_choice_item_shortcut(Fl_Widget*, int, const char*);

#include "Fl_Clock.H"
forms_constructort(Fl_Clock, fl_add_clock)
inline void fl_get_clock(Fl_Widget* o, int* h, int* m, int* s) {
    *h = ((Fl_Clock*)o)->hour();
    *m = ((Fl_Clock*)o)->minute();
    *s = ((Fl_Clock*)o)->second();
}

#include "Fl_Counter.H"
forms_constructor(Fl_Counter, fl_add_counter)
inline void fl_set_counter_value(Fl_Widget* o, double v) {
    ((Fl_Counter*)o)->value(v);}
inline void fl_set_counter_bounds(Fl_Widget* o, double a, double b) {
    ((Fl_Counter*)o)->bounds(a,b);}
inline void fl_set_counter_step(Fl_Widget* o, double a, double b) {
    ((Fl_Counter*)o)->step(a,b);}
inline void fl_set_counter_precision(Fl_Widget* o, int v) {
    ((Fl_Counter*)o)->precision(v);}
inline void fl_set_counter_return(Fl_Widget* o, int v) {
    ((Fl_Counter*)o)->when((uchar)(v|FL_WHEN_RELEASE));
}
```

```
inline double fl_get_counter_value(Fl_Widget* o) {
    return ((Fl_Counter*)o)->value();}
inline void fl_get_counter_bounds(Fl_Widget* o, float* a, float* b) {
  *a = float(((Fl_Counter*)o)->minimum());
  *b = float(((Fl_Counter*)o)->maximum());
}
//inline void fl_set_counter_filter(Fl_Widget*,const char* (*)(Fl_Widget*,double,int));

// Cursor stuff cannot be emulated because it uses X stuff
inline void fl_set_cursor(Fl_Window* w, Fl_Cursor c) {w->cursor(c);}
#define FL_INVISIBLE_CURSOR FL_CURSOR_NONE
#define FL_DEFAULT_CURSOR FL_CURSOR_DEFAULT

#include "Fl_Dial.H"

#define FL_DIAL_COL1 FL_GRAY
#define FL_DIAL_COL2 37

forms_constructor(Fl_Dial, fl_add_dial)
inline void fl_set_dial_value(Fl_Widget* o, double v) {
  ((Fl_Dial*)o)->value(v);}
inline double fl_get_dial_value(Fl_Widget* o) {
  return ((Fl_Dial*)o)->value();}
inline void fl_set_dial_bounds(Fl_Widget* o, double a, double b) {
  ((Fl_Dial*)o)->bounds(a, b);}
inline void fl_get_dial_bounds(Fl_Widget* o, float* a, float* b) {
  *a = float(((Fl_Dial*)o)->minimum());
  *b = float(((Fl_Dial*)o)->maximum());
}
inline void fl_set_dial_return(Fl_Widget* o, int i) {
  ((Fl_Dial*)o)->when((uchar)(i|FL_WHEN_RELEASE));
}
inline void fl_set_dial_angles(Fl_Widget* o, int a, int b) {
  ((Fl_Dial*)o)->angles((short)a, (short)b);}
//inline void fl_set_dial_cross(Fl_Widget* o, int);
// inline void fl_set_dial_direction(Fl_Widget* o, uchar d) {
//   ((Fl_Dial*)o)->direction(d);}
inline void fl_set_dial_step(Fl_Widget* o, double v) {
  ((Fl_Dial*)o)->step(v);}

// Frames:

inline Fl_Widget* fl_add_frame(Fl_Boxtype i,int x,int y,int w,int h,const char* l) {
  return fl_add_box(i,x-3,y-3,w+6,h+6,l);}

// labelframe nyi
inline Fl_Widget* fl_add_labelframe(Fl_Boxtype i,int x,int y,int w,int h,const char* l) {
  Fl_Widget* o = fl_add_box(i,x-3,y-3,w+6,h+6,l);
  o->align(FL_ALIGN_TOP_LEFT);
  return o;
}

#include "Fl_Free.H"
inline Fl_Free* fl_add_free(int t,double x,double y,double w,double h,const char* l,
                            FL_HANDLEPTR hdl) {
  return (Fl_Free*)(fl_add_new(new Fl_Free((uchar)t,int(x),int(y),int(w),int(h),l,hdl)));
}

#include "fl_ask.H"
#include "fl_show_colormap.H"

inline int fl_show_question(const char* c, int = 0) {return fl_choice("%s",fl_no,fl_yes,0L,c);}
FL_EXPORT void fl_show_message(const char *,const char *,const char *);
FL_EXPORT void fl_show_alert(const char *,const char *,const char *,int=0);
FL_EXPORT int fl_show_question(const char *,const char *,const char *);
inline const char *fl_show_input(const char *l,const char*d=0) {return fl_input("%s",d,l);}
FL_EXPORT /*const*/ char *fl_show_simple_input(const char *label, const char *deflt = 0);
FL_EXPORT int fl_show_choice(
    const char *m1,
    const char *m2,
    const char *m3,
    int numb,
    const char *b0,
    const char *b1,
    const char *b2);

inline void fl_set_goodies_font(Fl_Font a, Fl_Fontsize b) {fl_message_font(a,b);}
#define fl_show_messages fl_message
inline int fl_show_choices(const char* c,int n,const char* b1,const char* b2,
                           const char* b3, int) {
  return fl_show_choice(0,c,0,n,b1,b2,b3);
}

#include "filename.H"
#include "Fl_File_Chooser.H"
inline int do_matching(char* a, const char* b) {return fl_filename_match(a,b);}

```

```
// Forms-compatible file chooser (implementation in fselect.C):
FL_EXPORT char* fl_show_file_selector(const char* message,const char* dir,
                                      const char* pat,const char* fname);
FL_EXPORT char* fl_get_directory();
FL_EXPORT char* fl_get_pattern();
FL_EXPORT char* fl_get_filename();

#include "Fl_Input.H"
forms_constructor(Fl_Input, fl_add_input)
inline void fl_set_input(Fl_Widget* o, const char* v) {
    ((Fl_Input*)o)->value(v);}
inline void fl_set_input_return(Fl_Widget* o, int x) {
    ((Fl_Input*)o)->when((uchar)(x | FL_WHEN_RELEASE));
}
inline void fl_set_input_color(Fl_Widget* o, Fl_Color a, Fl_Color b) {
    ((Fl_Input*)o)->textcolor(a);
    ((Fl_Input*)o)->cursor_color(b);
}
// inline void fl_set_input_scroll(Fl_Widget*, int);
inline void fl_set_input_cursorpos(Fl_Widget* o, int x, int /*y*/) {
  ((Fl_Input*)o)->insert_position(x);}
// inline void fl_set_input_selected(Fl_Widget*, int);
// inline void fl_set_input_selected_range(Fl_Widget*, int, int);
// inline void fl_set_input_maxchars(Fl_Widget*, int);
// inline void fl_set_input_format(Fl_Widget*, int, int);
// inline void fl_set_input_hscrollbar(Fl_Widget*, int);
// inline void fl_set_input_vscrollbar(Fl_Widget*, int);
// inline void fl_set_input_xoffset(Fl_Widget*, int);
// inline void fl_set_input_topline(Fl_Widget*, int);
// inline void fl_set_input_scrollbarsize(Fl_Widget*, int, int);
// inline int fl_get_input_topline(Fl_Widget*);
// inline int fl_get_input_screenlines(Fl_Widget*);
inline int fl_get_input_cursorpos(Fl_Widget* o, int*x, int*y) {
  *x = ((Fl_Input*)o)->insert_position(); *y = 0; return *x;}
// inline int fl_get_input_numberoflines(Fl_Widget*);
// inline void fl_get_input_format(Fl_Widget*, int*, int*);
inline const char* fl_get_input(Fl_Widget* o) {return ((Fl_Input*)o)->value();}

#include "Fl_Menu_Button.H"

// types are not implemented, they all act like FL_PUSH_MENU:
#define FL_TOUCH_MENU           0
#define FL_PUSH_MENU            1
#define FL_PULLDOWN_MENU        2
forms_constructor(Fl_Menu_Button, fl_add_menu)

inline void fl_clear_menu(Fl_Widget* o) {
    ((Fl_Menu_Button*)o)->clear();}
inline void fl_set_menu(Fl_Widget* o, const char* s) {
    ((Fl_Menu_Button*)o)->clear(); ((Fl_Menu_Button*)o)->add(s);}
inline void fl_addto_menu(Fl_Widget* o, const char* s) {
    ((Fl_Menu_Button*)o)->add(s);}
inline void fl_replace_menu_item(Fl_Widget* o, int i, const char* s) {
    ((Fl_Menu_Button*)o)->replace(i-1,s);}
inline void fl_delete_menu_item(Fl_Widget* o, int i) {
    ((Fl_Menu_Button*)o)->remove(i-1);}
inline void fl_set_menu_item_shortcut(Fl_Widget* o, int i, const char* s) {
    ((Fl_Menu_Button*)o)->shortcut(i-1,fl_old_shortcut(s));}
inline void fl_set_menu_item_mode(Fl_Widget* o, int i, long x) {
    ((Fl_Menu_Button*)o)->mode(i-1,(int)x);}
inline void fl_show_menu_symbol(Fl_Widget*, int ) {
/*    ((Fl_Menu_Button*)o)->show_menu_symbol(i); */}
// inline void fl_set_menu_popup(Fl_Widget*, int);
inline int fl_get_menu(Fl_Widget* o) {
    return ((Fl_Menu_Button*)o)->value()+1;}
inline const char* fl_get_menu_item_text(Fl_Widget* o, int i) {
    return ((Fl_Menu_Button*)o)->text(i);}
inline int fl_get_menu_maxitems(Fl_Widget* o) {
    return ((Fl_Menu_Button*)o)->size();}
inline int fl_get_menu_item_mode(Fl_Widget* o, int i) {
    return ((Fl_Menu_Button*)o)->mode(i);}
inline const char* fl_get_menu_text(Fl_Widget* o) {
    return ((Fl_Menu_Button*)o)->text();}

#include "Fl_Positioner.H"
#define FL_NORMAL_POSITIONER    0
forms_constructor(Fl_Positioner, fl_add_positioner)
inline void fl_set_positioner_xvalue(Fl_Widget* o, double v) {
    ((Fl_Positioner*)o)->xvalue(v);}
inline double fl_get_positioner_xvalue(Fl_Widget* o) {
    return ((Fl_Positioner*)o)->xvalue();}
inline void fl_set_positioner_xbounds(Fl_Widget* o, double a, double b) {
    ((Fl_Positioner*)o)->xbounds(a,b);}
inline void fl_get_positioner_xbounds(Fl_Widget* o, float* a, float* b) {
  *a = float(((Fl_Positioner*)o)->xminimum());
  *b = float(((Fl_Positioner*)o)->xmaximum());
}
```

```
inline void fl_set_positioner_yvalue(Fl_Widget* o, double v) {
    ((Fl_Positioner*)o)->yvalue(v);}
inline double fl_get_positioner_yvalue(Fl_Widget* o) {
    return ((Fl_Positioner*)o)->yvalue();}
inline void fl_set_positioner_ybounds(Fl_Widget* o, double a, double b) {
    ((Fl_Positioner*)o)->ybounds(a,b);}
inline void fl_get_positioner_ybounds(Fl_Widget* o, float* a, float* b) {
  *a = float(((Fl_Positioner*)o)->yminimum());
  *b = float(((Fl_Positioner*)o)->ymaximum());
}
inline void fl_set_positioner_xstep(Fl_Widget* o, double v) {
    ((Fl_Positioner*)o)->xstep(v);}
inline void fl_set_positioner_ystep(Fl_Widget* o, double v) {
    ((Fl_Positioner*)o)->ystep(v);}
inline void fl_set_positioner_return(Fl_Widget* o, int v) {
    ((Fl_Positioner*)o)->when((uchar)(v|FL_WHEN_RELEASE));
}

#include "Fl_Slider.H"

#define FL_HOR_BROWSER_SLIDER FL_HOR_SLIDER
#define FL_VERT_BROWSER_SLIDER FL_VERT_SLIDER

forms_constructort(Fl_Slider, fl_add_slider)
#define FL_SLIDER_COL1 FL_GRAY
inline void fl_set_slider_value(Fl_Widget* o, double v) {
    ((Fl_Slider*)o)->value(v);}
inline double fl_get_slider_value(Fl_Widget* o) {
    return ((Fl_Slider*)o)->value();}
inline void fl_set_slider_bounds(Fl_Widget* o, double a, double b) {
    ((Fl_Slider*)o)->bounds(a, b);}
inline void fl_get_slider_bounds(Fl_Widget* o, float* a, float* b) {
  *a = float(((Fl_Slider*)o)->minimum());
  *b = float(((Fl_Slider*)o)->maximum());
}
inline void fl_set_slider_return(Fl_Widget* o, int i) {
    ((Fl_Slider*)o)->when((uchar)(i|FL_WHEN_RELEASE));
}
inline void fl_set_slider_step(Fl_Widget* o, double v) {
    ((Fl_Slider*)o)->step(v);}
// inline void fl_set_slider_increment(Fl_Widget* o, double v, double);
inline void fl_set_slider_size(Fl_Widget* o, double v) {
    ((Fl_Slider*)o)->slider_size(v);}

#include "Fl_Value_Slider.H"
forms_constructor(Fl_Value_Slider, fl_add_valslider)

inline void fl_set_slider_precision(Fl_Widget* o, int i) {
    ((Fl_Value_Slider*)o)->precision(i);}
// filter function!

// The forms text object was the same as an Fl_Box except it inverted the
// meaning of FL_ALIGN_INSIDE.  Implementation in forms.cxx
class FL_EXPORT Fl_FormsText : public Fl_Widget {
protected:
    void draw() override;
public:
    Fl_FormsText(Fl_Boxtype b, int X, int Y, int W, int H, const char* l=0)
        : Fl_Widget(X,Y,W,H,l) {box(b); align(FL_ALIGN_LEFT);}
};
#define FL_NORMAL_TEXT FL_NO_BOX
forms_constructorb(Fl_FormsText, fl_add_text)

#include "Fl_Timer.H"
forms_constructort(Fl_Timer, fl_add_timer)
inline void fl_set_timer(Fl_Widget* o, double v) {((Fl_Timer*)o)->value(v);}
inline double fl_get_timer(Fl_Widget* o) {return ((Fl_Timer*)o)->value();}
inline void fl_suspend_timer(Fl_Widget* o) {((Fl_Timer*)o)->suspended(1);}
inline void fl_resume_timer(Fl_Widget* o) {((Fl_Timer*)o)->suspended(0);}
inline void fl_set_timer_countup(Fl_Widget* o,char d) {((Fl_Timer*)o)->direction(d);}
void FL_EXPORT fl_gettime(long* sec, long* usec);

// Fl_XYPlot nyi


// stuff from DDForms:

inline int fl_double_click() {return Fl::event_clicks();}
inline void fl_draw() {Fl::flush();}

#endif  /* define __FORMS_H__ */
```

14.196 gl.h File Reference

This file defines wrapper functions for OpenGL in FLTK.

```
#include "Enumerations.H"
#include <GL/gl.h>
```

Functions

- void gl_color (Fl_Color i)

  Sets the curent OpenGL color to an FLTK color.
- void **gl_color** (int c)

  back compatibility
- int **gl_descent** ()

  Returns the current font's descent.
- void gl_draw (const char ∗)

  Draws a nul-terminated string in the current font at the current position.
- void gl_draw (const char ∗, float x, float y)

  Draws a nul-terminated string in the current font at the given position.
- void gl_draw (const char ∗, int n)

  Draws an array of n characters of the string in the current font at the current position.
- void gl_draw (const char ∗, int n, float x, float y)

  Draws n characters of the string in the current font at the given position.
- void gl_draw (const char ∗, int n, int x, int y)

  Draws n characters of the string in the current font at the given position.
- void gl_draw (const char ∗, int x, int y)

  Draws a nul-terminated string in the current font at the given position.
- void gl_draw (const char ∗, int x, int y, int w, int h, Fl_Align)

  Draws a string formatted into a box, with newlines and tabs expanded, other control characters changed to ^X.
- void **gl_draw_image** (const uchar ∗, int x, int y, int w, int h, int d=3, int ld=0)
- void **gl_finish** ()

  Releases an OpenGL context.
- void gl_font (int fontid, int size)

  Sets the current OpenGL font to the same font as calling fl_font().
- int **gl_height** ()

  Returns the current font's height.
- void **gl_measure** (const char ∗, int &x, int &y)

  Measure how wide and tall the string will be when drawn by the gl_draw() function.
- void gl_rect (int x, int y, int w, int h)

  Outlines the given rectangle with the current color.
- void gl_rectf (int x, int y, int w, int h)

  Fills the given rectangle with the current color.
- void **gl_start** ()

  Creates an OpenGL context.
- int gl_texture_pile_height ()

  Returns the current maximum height of the pile of pre-computed string textures.
- void gl_texture_pile_height (int max)

  Changes the maximum height of the pile of pre-computed string textures.
- void **gl_texture_reset** ()

  To call after GL operations that may invalidate textures used to draw text in GL scenes (e.g., switch between FL_↩ DOUBLE / FL_SINGLE modes).
- double **gl_width** (const char ∗)

Returns the width of the string in the current fnt.

- double **gl_width** (const char *, int n)

  Returns the width of n characters of the string in the current font.

- double **gl_width** (uchar)

  Returns the width of the character in the current font.

14.196.1 Detailed Description

This file defines wrapper functions for OpenGL in FLTK.
To use OpenGL from within an FLTK application you MUST use gl_visual() to select the default visual before doing show() on any windows. Mesa will crash if you try to use a visual not returned by glxChooseVisual.
Historically, this did not always work well with Fl_Double_Window's! It can try to draw into the front buffer. Depending on the system this might either crash or do nothing (when pixmaps are being used as back buffer and GL is being done by hardware), work correctly (when GL is done with software, such as Mesa), or draw into the front buffer and be erased when the buffers are swapped (when double buffer hardware is being used)

14.196.2 Function Documentation

14.196.2.1 gl_color()

```
void gl_color (
            Fl_Color i )
```

Sets the curent OpenGL color to an FLTK color.
For color-index modes it will use fl_xpixel(c), which is only right if the window uses the default colormap!

14.196.2.2 gl_draw() [1/7]

```
void gl_draw (
            const char * str )
```

Draws a nul-terminated string in the current font at the current position.

See also

gl_texture_pile_height(int)

14.196.2.3 gl_draw() [2/7]

```
void gl_draw (
            const char * str,
            float x,
            float y )
```

Draws a nul-terminated string in the current font at the given position.

See also

gl_texture_pile_height(int)

14.196.2.4 gl_draw() [3/7]

```
void gl_draw (
            const char * str,
            int n )
```

Draws an array of n characters of the string in the current font at the current position.

See also

gl_texture_pile_height(int)

14.196.2.5 gl_draw() [4/7]

```
void gl_draw (
            const char * str,
            int n,
            float x,
            float y )
```

Draws n characters of the string in the current font at the given position.

See also

gl_texture_pile_height(int)

14.196.2.6 gl_draw() [5/7]

```
void gl_draw (
            const char * str,
            int n,
            int x,
            int y )
```

Draws n characters of the string in the current font at the given position.

See also

gl_texture_pile_height(int)

14.196.2.7 gl_draw() [6/7]

```
void gl_draw (
            const char * str,
            int x,
            int y )
```

Draws a nul-terminated string in the current font at the given position.

See also

gl_texture_pile_height(int)

14.196.2.8 gl_draw() [7/7]

```
void gl_draw (
            const char * str,
            int x,
            int y,
            int w,
            int h,
            Fl_Align align )
```

Draws a string formatted into a box, with newlines and tabs expanded, other control characters changed to ^X. and aligned with the edges or center. Exactly the same output as fl_draw().

14.196.2.9 gl_font()

```
void gl_font (
            int fontid,
            int size )
```

Sets the current OpenGL font to the same font as calling fl_font().

See also

Fl::draw_GL_text_with_textures(int val)

14.196.2.10 gl_rect()

```
void gl_rect (
            int x,
            int y,
            int w,
            int h )
```

Outlines the given rectangle with the current color.
If Fl_Gl_Window::ortho() has been called, then the rectangle will exactly fill the given pixel rectangle.

14.196.2.11 gl_rectf()

```
void gl_rectf (
            int x,
            int y,
            int w,
            int h )
```

Fills the given rectangle with the current color.

See also

gl_rect(int x, int y, int w, int h)

14.196.2.12 gl_texture_pile_height() [1/2]

```
int gl_texture_pile_height (
            void )
```

Returns the current maximum height of the pile of pre-computed string textures.
The default value is 100

See also

Fl::draw_GL_text_with_textures(int)

14.196.2.13 gl_texture_pile_height() [2/2]

```
void gl_texture_pile_height (
            int max )
```

Changes the maximum height of the pile of pre-computed string textures.
Strings that are often re-displayed can be processed much faster if this pile is set high enough to hold all of them.

Parameters

max	Maximum height of the texture pile

See also

Fl::draw_GL_text_with_textures(int)

14.197 gl.h

Go to the documentation of this file.

```
00001 //
00002 // OpenGL header file for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 1998-2018 by Bill Spitzak and others.
00005 //
00006 // You must include this instead of GL/gl.h to get the Microsoft
00007 // APIENTRY stuff included (from <windows.h>) prior to the OpenGL
00008 // header files.
00009 //
00010 // This file also provides "missing" OpenGL functions, and
```

```
// gl_start() and gl_finish() to allow OpenGL to be used in any window
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef FL_gl_H
#  define FL_gl_H

#  include "Enumerations.H" // for color names
#  ifdef _WIN32
#    include <windows.h>
#  endif
#  ifndef APIENTRY
#    if defined(__CYGWIN__)
#      define APIENTRY __attribute__ ((__stdcall__))
#    else
#      define APIENTRY
#    endif
#  endif

#  ifdef __APPLE__ // PORTME: OpenGL path abstraction
#    ifndef GL_SILENCE_DEPRECATION
#      define GL_SILENCE_DEPRECATION 1
#    endif
#    if !defined(__gl3_h_) // make sure OpenGL/gl3.h was not included before
#      include <OpenGL/gl.h>
#    endif
#  else
#    include <GL/gl.h>
#  endif  // __APPLE__ // PORTME: OpenGL Path abstraction

FL_EXPORT void gl_start();
FL_EXPORT void gl_finish();

FL_EXPORT void gl_color(Fl_Color i);
inline void gl_color(int c) {gl_color((Fl_Color)c);}

FL_EXPORT void gl_rect(int x,int y,int w,int h);
FL_EXPORT void gl_rectf(int x,int y,int w,int h);

FL_EXPORT void gl_font(int fontid, int size);
FL_EXPORT int  gl_height();
FL_EXPORT int  gl_descent();
FL_EXPORT double gl_width(const char *);
FL_EXPORT double gl_width(const char *, int n);
FL_EXPORT double gl_width(uchar);

FL_EXPORT void gl_draw(const char*);
FL_EXPORT void gl_draw(const char*, int n);
FL_EXPORT void gl_draw(const char*, int x, int y);
FL_EXPORT void gl_draw(const char*, float x, float y);
FL_EXPORT void gl_draw(const char*, int n, int x, int y);
FL_EXPORT void gl_draw(const char*, int n, float x, float y);
FL_EXPORT void gl_draw(const char*, int x, int y, int w, int h, Fl_Align);
FL_EXPORT void gl_measure(const char*, int& x, int& y);
FL_EXPORT void gl_texture_pile_height(int max);
FL_EXPORT int  gl_texture_pile_height();
FL_EXPORT void gl_texture_reset();

FL_EXPORT void gl_draw_image(const uchar *, int x,int y,int w,int h, int d=3, int ld=0);

#endif // !FL_gl_H
```

14.198 gl2opengl.h

```
/*      gl.h

        GL to OpenGL translator.
        If you include this, you might be able to port old GL programs.
        There are also much better emulators available on the net.

*/

#ifndef _FL_gl2opengl_h_
#define _FL_gl2opengl_h_

#include <FL/gl.h>
```

```
#include "gl_draw.H"

inline void clear() {glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);}
#define RGBcolor(r,g,b) glColor3ub(r,g,b)
#define bgnline() glBegin(GL_LINE_STRIP)
#define bgnpolygon() glBegin(GL_POLYGON)
#define bgnclosedline() glBegin(GL_LINE_LOOP)
#define endline() glEnd()
#define endpolygon() glEnd()
#define endclosedline() glEnd()
#define v2f(v) glVertex2fv(v)
#define v2s(v) glVertex2sv(v)
#define cmov(x,y,z) glRasterPos3f(x,y,z)
#define charstr(s) gl_draw(s)
#define fmprstr(s) gl_draw(s)
typedef float Matrix[4][4];
inline void pushmatrix() {glPushMatrix();}
inline void popmatrix() {glPopMatrix();}
inline void multmatrix(Matrix m) {glMultMatrixf((float *)m);}
inline void color(int n) {glIndexi(n);}
inline void rect(int x,int y,int r,int t) {gl_rect(x,y,r-x,t-y);}
inline void rectf(int x,int y,int r,int t) {glRectf(x,y,r+1,t+1);}
inline void recti(int x,int y,int r,int t) {gl_rect(x,y,r-x,t-y);}
inline void rectfi(int x,int y,int r,int t) {glRecti(x,y,r+1,t+1);}
inline void rects(int x,int y,int r,int t) {gl_rect(x,y,r-x,t-y);}
inline void rectfs(int x,int y,int r,int t) {glRects(x,y,r+1,t+1);}

#endif /* _FL_gl2opengl_h_ */
```

14.199 gl_draw.H

```
//
// OpenGL header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#include "gl.h"

extern FL_EXPORT void gl_remove_displaylist_fonts();
```

14.200 glu.h

```
//
// GLu header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// You must include this instead of GL/gl.h to get the Microsoft
// APIENTRY stuff included (from <windows.h>) prior to the OpenGL
// header files.
//
// This file also provides "missing" OpenGL functions, and
// gl_start() and gl_finish() to allow OpenGL to be used in any window
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef FL_glu_H
#  define FL_glu_H

#  include "Enumerations.H" // for color names
#  ifdef _WIN32
#    include <windows.h>
```

```
#  endif
#  ifndef APIENTRY
#    if defined(__CYGWIN__)
#      define APIENTRY __attribute__ ((__stdcall__))
#    else
#      define APIENTRY
#    endif
#  endif

#  ifdef __APPLE__  // PORTME: OpenGL Path abstraction
#    include <OpenGL/glu.h>
#  else
#    include <GL/glu.h>
#  endif

#endif // !FL_glu_H
```

14.201 glut.H

```
//
// GLUT emulation header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2023 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

// Emulation of GLUT using fltk.

// GLUT is Copyright (c) Mark J. Kilgard, 1994, 1995, 1996:
// "This program is freely distributable without licensing fees  and is
// provided without guarantee or warrantee expressed or  implied. This
// program is -not- in the public domain."

// Although I have copied the GLUT API, none of my code is based on
// any GLUT implementation details and is therefore covered by the LGPL.

// Commented out lines indicate parts of GLUT that are not emulated.

// Notes: as pointed out in STR #3458 the current GLUT window,
// i.e. the global static variable 'glut_window' can be NULL ...
//  (a) if not (yet) initialized
//  (b) if the current GLUT window is deleted at any time.
// The FLTK implementation silently ignores function calls if the current
// window is NULL to avoid dereferencing a NULL pointer. This is obviously
// compatible with GLUT version 3.7 according to comment #5 on STR #3458.
// According to the same comment FreeGLUT 3.0 would issue an error message
// and quit.
// Albrecht-S, Oct 2023

#ifndef _FL_glut_H_
#  define _FL_glut_H_

#  include "gl.h"


#  include "Fl.H"
#  include "Fl_Gl_Window.H"

class FL_EXPORT Fl_Glut_Window : public Fl_Gl_Window {
  void _init();
  int mouse_down;
protected:
  void draw() override;
  void draw_overlay() override;
  int handle(int) override;
public: // so the inline functions work
  int number;
  int menu[3];
  void make_current();
  void (*display)();
  void (*overlaydisplay)();
  void (*reshape)(int w, int h);
  void (*keyboard)(uchar, int x, int y);
  void (*mouse)(int b, int state, int x, int y);
  void (*motion)(int x, int y);
  void (*passivemotion)(int x, int y);
```

```
  void (*entry)(int);
  void (*visibility)(int);
  void (*special)(int, int x, int y);
  Fl_Glut_Window(int w, int h, const char *t=0);
  Fl_Glut_Window(int x, int y, int w, int h, const char *t=0);
  ~Fl_Glut_Window();
};

extern FL_EXPORT Fl_Glut_Window *glut_window;   // the current window
extern FL_EXPORT int glut_menu;                 // the current menu

// function pointers that are not per-window:
extern FL_EXPORT void (*glut_idle_function)();
extern FL_EXPORT void (*glut_menustate_function)(int);
extern FL_EXPORT void (*glut_menustatus_function)(int,int,int);


//#  define GLUT_API_VERSION This does not match any version of GLUT exactly...

FL_EXPORT void glutInit(int *argcp, char **argv); // creates first window

FL_EXPORT void glutInitDisplayMode(unsigned int mode);
// the FL_ symbols have the same value as the GLUT ones:
#  define GLUT_RGB            FL_RGB
#  define GLUT_RGBA           FL_RGB
#  define GLUT_INDEX          FL_INDEX
#  define GLUT_SINGLE         FL_SINGLE
#  define GLUT_DOUBLE         FL_DOUBLE
#  define GLUT_ACCUM          FL_ACCUM
#  define GLUT_ALPHA          FL_ALPHA
#  define GLUT_DEPTH          FL_DEPTH
#  define GLUT_STENCIL        FL_STENCIL
#  define GLUT_MULTISAMPLE    FL_MULTISAMPLE
#  define GLUT_STEREO         FL_STEREO
// #  define GLUT_LUMINANCE   512

FL_EXPORT void glutInitWindowPosition(int x, int y);

FL_EXPORT void glutInitWindowSize(int w, int h);

FL_EXPORT void glutMainLoop();

FL_EXPORT int glutCreateWindow(char *title);
FL_EXPORT int glutCreateWindow(const char *title);

FL_EXPORT int glutCreateSubWindow(int win, int x, int y, int width, int height);

FL_EXPORT void glutDestroyWindow(int win);

inline void glutPostRedisplay() {
  if (glut_window) glut_window->redraw();
}

FL_EXPORT void glutPostWindowRedisplay(int win);

FL_EXPORT void glutSwapBuffers();

inline int glutGetWindow() {
  return glut_window ? glut_window->number : 0;
}

FL_EXPORT void glutSetWindow(int win);

inline void glutSetWindowTitle(char *t) {
  if (glut_window) glut_window->label(t);
}

inline void glutSetIconTitle(char *t) {
  if (glut_window) glut_window->iconlabel(t);
}

inline void glutPositionWindow(int x, int y) {
  if (glut_window) glut_window->position(x,y);
}

inline void glutReshapeWindow(int w, int h) {
  if (glut_window) glut_window->size(w,h);
}

inline void glutPopWindow() {
  if (glut_window) glut_window->show();
}

inline void glutPushWindow() { /* do nothing */ }

inline void glutIconifyWindow() {
  if (glut_window) glut_window->iconize();
```

```
}

inline void glutShowWindow() {
  if (glut_window) glut_window->show();
}

inline void glutHideWindow() {
  if (glut_window) glut_window->hide();
}

inline void glutFullScreen() {
  if (glut_window) glut_window->fullscreen();
}

inline void glutSetCursor(Fl_Cursor cursor) {
  if (glut_window) glut_window->cursor(cursor);
}

// notice that the numeric values are different than glut:
#  define GLUT_CURSOR_RIGHT_ARROW              ((Fl_Cursor)2)
#  define GLUT_CURSOR_LEFT_ARROW               ((Fl_Cursor)67)
#  define GLUT_CURSOR_INFO                     FL_CURSOR_HAND
#  define GLUT_CURSOR_DESTROY                  ((Fl_Cursor)45)
#  define GLUT_CURSOR_HELP                     FL_CURSOR_HELP
#  define GLUT_CURSOR_CYCLE                    ((Fl_Cursor)26)
#  define GLUT_CURSOR_SPRAY                    ((Fl_Cursor)63)
#  define GLUT_CURSOR_WAIT                     FL_CURSOR_WAIT
#  define GLUT_CURSOR_TEXT                     FL_CURSOR_INSERT
#  define GLUT_CURSOR_CROSSHAIR                FL_CURSOR_CROSS
#  define GLUT_CURSOR_UP_DOWN                  FL_CURSOR_NS
#  define GLUT_CURSOR_LEFT_RIGHT               FL_CURSOR_WE
#  define GLUT_CURSOR_TOP_SIDE                 FL_CURSOR_N
#  define GLUT_CURSOR_BOTTOM_SIDE              FL_CURSOR_S
#  define GLUT_CURSOR_LEFT_SIDE                FL_CURSOR_W
#  define GLUT_CURSOR_RIGHT_SIDE               FL_CURSOR_E
#  define GLUT_CURSOR_TOP_LEFT_CORNER          FL_CURSOR_NW
#  define GLUT_CURSOR_TOP_RIGHT_CORNER         FL_CURSOR_NE
#  define GLUT_CURSOR_BOTTOM_RIGHT_CORNER      FL_CURSOR_SE
#  define GLUT_CURSOR_BOTTOM_LEFT_CORNER       FL_CURSOR_SW
#  define GLUT_CURSOR_INHERIT                  FL_CURSOR_DEFAULT
#  define GLUT_CURSOR_NONE                     FL_CURSOR_NONE
#  define GLUT_CURSOR_FULL_CROSSHAIR           FL_CURSOR_CROSS

inline void glutWarpPointer(int, int) { /* do nothing */ }

inline void glutEstablishOverlay() {
  if (glut_window) glut_window->make_overlay_current();
}

inline void glutRemoveOverlay() {
  if (glut_window) glut_window->hide_overlay();
}

inline void glutUseLayer(GLenum layer) {
  if (!glut_window)
    return;
  layer ? glut_window->make_overlay_current() : glut_window->make_current();
}

enum {GLUT_NORMAL, GLUT_OVERLAY};

inline void glutPostOverlayRedisplay() {
  if (glut_window) glut_window->redraw_overlay();
}

inline void glutShowOverlay() {
  if (glut_window) glut_window->redraw_overlay();
}

inline void glutHideOverlay() {
  if (glut_window) glut_window->hide_overlay();
}

FL_EXPORT int glutCreateMenu(void (*)(int));

FL_EXPORT void glutDestroyMenu(int menu);

inline int glutGetMenu() {return glut_menu;}

inline void glutSetMenu(int m) {glut_menu = m;}

FL_EXPORT void glutAddMenuEntry(const char *label, int value);

FL_EXPORT void glutAddSubMenu(char *label, int submenu);

FL_EXPORT void glutChangeToMenuEntry(int item, char *labela, int value);

```

```
FL_EXPORT void glutChangeToSubMenu(int item, char *label, int submenu);

FL_EXPORT void glutRemoveMenuItem(int item);

inline void glutAttachMenu(int b) {
  if (glut_window) glut_window->menu[b] = glut_menu;
}

inline void glutDetachMenu(int b) {
  if (glut_window) glut_window->menu[b] = 0;
}

inline void glutDisplayFunc(void (*f)()) {
  if (glut_window) glut_window->display = f;
}

inline void glutReshapeFunc(void (*f)(int w, int h)) {
  if (glut_window) glut_window->reshape = f;
}

inline void glutKeyboardFunc(void (*f)(uchar key, int x, int y)) {
  if (glut_window) glut_window->keyboard = f;
}

inline void glutMouseFunc(void (*f)(int b, int state, int x, int y)) {
  if (glut_window) glut_window->mouse = f;
}

#  define GLUT_LEFT_BUTTON              0
#  define GLUT_MIDDLE_BUTTON            1
#  define GLUT_RIGHT_BUTTON             2
#  define GLUT_DOWN                     0
#  define GLUT_UP                       1

inline void glutMotionFunc(void (*f)(int x, int y)) {
  if (glut_window) glut_window->motion = f;
}

inline void glutPassiveMotionFunc(void (*f)(int x, int y)) {
  if (glut_window) glut_window->passivemotion = f;
}

inline void glutEntryFunc(void (*f)(int s)) {
  if (glut_window) glut_window->entry = f;
}

enum {GLUT_LEFT, GLUT_ENTERED};

inline void glutVisibilityFunc(void (*f)(int s)) {
  if (glut_window) glut_window->visibility = f;
}
enum {GLUT_NOT_VISIBLE, GLUT_VISIBLE};

FL_EXPORT void glutIdleFunc(void (*f)());

inline void glutTimerFunc(unsigned int msec, void (*f)(int), int value) {
  Fl::add_timeout(msec*.001, (void (*)(void *))f, (void *)(fl_intptr_t)value);
}

inline void glutMenuStateFunc(void (*f)(int state)) {
  glut_menustate_function = f;}

inline void glutMenuStatusFunc(void (*f)(int status, int x, int y)) {
  glut_menustatus_function = f;}
enum {GLUT_MENU_NOT_IN_USE, GLUT_MENU_IN_USE};

inline void glutSpecialFunc(void (*f)(int key, int x, int y)) {
  if (glut_window) glut_window->special = f;
}

#  define GLUT_KEY_F1                   1
#  define GLUT_KEY_F2                   2
#  define GLUT_KEY_F3                   3
#  define GLUT_KEY_F4                   4
#  define GLUT_KEY_F5                   5
#  define GLUT_KEY_F6                   6
#  define GLUT_KEY_F7                   7
#  define GLUT_KEY_F8                   8
#  define GLUT_KEY_F9                   9
#  define GLUT_KEY_F10                  10
#  define GLUT_KEY_F11                  11
#  define GLUT_KEY_F12                  12
// WARNING: Different values than GLUT uses:
#  define GLUT_KEY_LEFT                 FL_Left
#  define GLUT_KEY_UP                   FL_Up
#  define GLUT_KEY_RIGHT                FL_Right
#  define GLUT_KEY_DOWN                 FL_Down
```

```
#  define GLUT_KEY_PAGE_UP              FL_Page_Up
#  define GLUT_KEY_PAGE_DOWN            FL_Page_Down
#  define GLUT_KEY_HOME                 FL_Home
#  define GLUT_KEY_END                  FL_End
#  define GLUT_KEY_INSERT               FL_Insert

// inline void glutSpaceballMotionFunc(void (*)(int x, int y, int z));

// inline void glutSpaceballRotateFunc(void (*)(int x, int y, int z));

// inline void glutSpaceballButtonFunc(void (*)(int button, int state));

// inline void glutButtonBoxFunc(void (*)(int button, int state));

// inline void glutDialsFunc(void (*)(int dial, int value));

// inline void glutTabletMotionFunc(void (*)(int x, int y));

// inline void glutTabletButtonFunc(void (*)(int button, int state, int x, int y));

inline void glutOverlayDisplayFunc(void (*f)()) {
  if (glut_window) glut_window->overlaydisplay = f;
}

// inline void glutWindowStatusFunc(void (*)(int state));
// enum {GLUT_HIDDEN, GLUT_FULLY_RETAINED, GLUT_PARTIALLY_RETAINED,
//       GLUT_FULLY_COVERED};

// inline void glutSetColor(int, GLfloat red, GLfloat green, GLfloat blue);

// inline GLfloat glutGetColor(int ndx, int component);
// #define GLUT_RED                      0
// #define GLUT_GREEN                    1
// #define GLUT_BLUE                     2

// inline void glutCopyColormap(int win);

// Warning: values are changed from GLUT!
// Also relies on the GL_ symbols having values greater than 100
FL_EXPORT int glutGet(GLenum type);
enum {
  GLUT_RETURN_ZERO = 0,
  GLUT_WINDOW_X,
  GLUT_WINDOW_Y,
  GLUT_WINDOW_WIDTH,
  GLUT_WINDOW_HEIGHT,
  GLUT_WINDOW_PARENT,
  GLUT_SCREEN_WIDTH,
  GLUT_SCREEN_HEIGHT,
  GLUT_MENU_NUM_ITEMS,
  GLUT_DISPLAY_MODE_POSSIBLE,
  GLUT_INIT_WINDOW_X,
  GLUT_INIT_WINDOW_Y,
  GLUT_INIT_WINDOW_WIDTH,
  GLUT_INIT_WINDOW_HEIGHT,
  GLUT_INIT_DISPLAY_MODE,
  GLUT_WINDOW_BUFFER_SIZE,
  GLUT_VERSION,
//GLUT_WINDOW_NUM_CHILDREN,
//GLUT_WINDOW_CURSOR,
//GLUT_SCREEN_WIDTH_MM,
//GLUT_SCREEN_HEIGHT_MM,
  GLUT_ELAPSED_TIME
};

#  define GLUT_WINDOW_STENCIL_SIZE      GL_STENCIL_BITS
#  define GLUT_WINDOW_DEPTH_SIZE        GL_DEPTH_BITS
#  define GLUT_WINDOW_RED_SIZE          GL_RED_BITS
#  define GLUT_WINDOW_GREEN_SIZE        GL_GREEN_BITS
#  define GLUT_WINDOW_BLUE_SIZE         GL_BLUE_BITS
#  define GLUT_WINDOW_ALPHA_SIZE        GL_ALPHA_BITS
#  define GLUT_WINDOW_ACCUM_RED_SIZE    GL_ACCUM_RED_BITS
#  define GLUT_WINDOW_ACCUM_GREEN_SIZE  GL_ACCUM_GREEN_BITS
#  define GLUT_WINDOW_ACCUM_BLUE_SIZE   GL_ACCUM_BLUE_BITS
#  define GLUT_WINDOW_ACCUM_ALPHA_SIZE  GL_ACCUM_ALPHA_BITS
#  define GLUT_WINDOW_DOUBLEBUFFER      GL_DOUBLEBUFFER
#  define GLUT_WINDOW_RGBA              GL_RGBA
#  define GLUT_WINDOW_COLORMAP_SIZE     GL_INDEX_BITS
#  ifdef GL_SAMPLES_SGIS
#    define GLUT_WINDOW_NUM_SAMPLES     GL_SAMPLES_SGIS
#  else
#    define GLUT_WINDOW_NUM_SAMPLES     GLUT_RETURN_ZERO
#  endif
#  define GLUT_WINDOW_STEREO            GL_STEREO

#  define GLUT_HAS_KEYBOARD             600
#  define GLUT_HAS_MOUSE                601
```

```
#  define GLUT_HAS_SPACEBALL            602
#  define GLUT_HAS_DIAL_AND_BUTTON_BOX  603
#  define GLUT_HAS_TABLET               604
#  define GLUT_NUM_MOUSE_BUTTONS        605
#  define GLUT_NUM_SPACEBALL_BUTTONS    606
#  define GLUT_NUM_BUTTON_BOX_BUTTONS   607
#  define GLUT_NUM_DIALS                608
#  define GLUT_NUM_TABLET_BUTTONS       609
FL_EXPORT int glutDeviceGet(GLenum type);

// WARNING: these values are different than GLUT uses:
#  define GLUT_ACTIVE_SHIFT               FL_SHIFT
#  define GLUT_ACTIVE_CTRL                FL_CTRL
#  define GLUT_ACTIVE_ALT                 FL_ALT

inline int glutGetModifiers() {
  return Fl::event_state() & (GLUT_ACTIVE_SHIFT | GLUT_ACTIVE_CTRL | GLUT_ACTIVE_ALT);
}

FL_EXPORT int glutLayerGet(GLenum);
#  define GLUT_OVERLAY_POSSIBLE         800
//#define GLUT_LAYER_IN_USE             801
//#define GLUT_HAS_OVERLAY              802
#  define GLUT_TRANSPARENT_INDEX        803
#  define GLUT_NORMAL_DAMAGED           804
#  define GLUT_OVERLAY_DAMAGED          805

extern "C" {
typedef void (*GLUTproc)();
}

FL_EXPORT GLUTproc glutGetProcAddress(const char *procName);

// inline int glutVideoResizeGet(GLenum param);
// #define GLUT_VIDEO_RESIZE_POSSIBLE     900
// #define GLUT_VIDEO_RESIZE_IN_USE       901
// #define GLUT_VIDEO_RESIZE_X_DELTA      902
// #define GLUT_VIDEO_RESIZE_Y_DELTA      903
// #define GLUT_VIDEO_RESIZE_WIDTH_DELTA  904
// #define GLUT_VIDEO_RESIZE_HEIGHT_DELTA 905
// #define GLUT_VIDEO_RESIZE_X            906
// #define GLUT_VIDEO_RESIZE_Y            907
// #define GLUT_VIDEO_RESIZE_WIDTH        908
// #define GLUT_VIDEO_RESIZE_HEIGHT       909

// inline void glutSetupVideoResizing();

// inline void glutStopVideoResizing();

// inline void glutVideoResize(int x, int y, int width, int height);

// inline void glutVideoPan(int x, int y, int width, int height);

// Font argument must be a void* for compatibility, so...
struct Fl_Glut_Bitmap_Font {Fl_Font font; Fl_Fontsize size;};

extern FL_EXPORT struct Fl_Glut_Bitmap_Font
  glutBitmap9By15, glutBitmap8By13, glutBitmapTimesRoman10,
  glutBitmapTimesRoman24, glutBitmapHelvetica10, glutBitmapHelvetica12,
  glutBitmapHelvetica18;
#  define GLUT_BITMAP_9_BY_15             (&glutBitmap9By15)
#  define GLUT_BITMAP_8_BY_13             (&glutBitmap8By13)
#  define GLUT_BITMAP_TIMES_ROMAN_10      (&glutBitmapTimesRoman10)
#  define GLUT_BITMAP_TIMES_ROMAN_24      (&glutBitmapTimesRoman24)
#  define GLUT_BITMAP_HELVETICA_10        (&glutBitmapHelvetica10)
#  define GLUT_BITMAP_HELVETICA_12        (&glutBitmapHelvetica12)
#  define GLUT_BITMAP_HELVETICA_18        (&glutBitmapHelvetica18)

FL_EXPORT void glutBitmapCharacter(void *font, int character);
FL_EXPORT int glutBitmapHeight(void *font);
FL_EXPORT int glutBitmapLength(void *font, const unsigned char *string);
FL_EXPORT void glutBitmapString(void *font, const unsigned char *string);
FL_EXPORT int glutBitmapWidth(void *font, int character);

FL_EXPORT int glutExtensionSupported(char *name);

/* GLUT stroked font sub-API */
struct Fl_Glut_StrokeVertex {
  GLfloat X, Y;
};

struct Fl_Glut_StrokeStrip {
  int Number;
  const Fl_Glut_StrokeVertex* Vertices;
};

struct Fl_Glut_StrokeChar {
```

```
  GLfloat Right;
  int Number;
  const Fl_Glut_StrokeStrip* Strips;
};

struct Fl_Glut_StrokeFont {
  char* Name;                          // The source font name
  int Quantity;                        // Number of chars in font
  GLfloat Height;                      // Height of the characters
  const Fl_Glut_StrokeChar** Characters;// The characters mapping
};
extern FL_EXPORT Fl_Glut_StrokeFont glutStrokeRoman;
extern FL_EXPORT Fl_Glut_StrokeFont glutStrokeMonoRoman;
#  define GLUT_STROKE_ROMAN             (&glutStrokeRoman)
#  define GLUT_STROKE_MONO_ROMAN        (&glutStrokeMonoRoman)

FL_EXPORT void glutStrokeCharacter(void *font, int character);
FL_EXPORT GLfloat glutStrokeHeight(void *font);
FL_EXPORT int glutStrokeLength(void *font, const unsigned char *string);
FL_EXPORT void glutStrokeString(void *font, const unsigned char *string);
FL_EXPORT int glutStrokeWidth(void *font, int character);

/* GLUT pre-built models sub-API */
FL_EXPORT void glutWireSphere(GLdouble radius, GLint slices, GLint stacks);
FL_EXPORT void glutSolidSphere(GLdouble radius, GLint slices, GLint stacks);
FL_EXPORT void glutWireCone(GLdouble base, GLdouble height, GLint slices, GLint stacks);
FL_EXPORT void glutSolidCone(GLdouble base, GLdouble height, GLint slices, GLint stacks);
FL_EXPORT void glutWireCube(GLdouble size);
FL_EXPORT void glutSolidCube(GLdouble size);
FL_EXPORT void glutWireTorus(GLdouble innerRadius, GLdouble outerRadius, GLint sides, GLint rings);
FL_EXPORT void glutSolidTorus(GLdouble innerRadius, GLdouble outerRadius, GLint sides, GLint rings);
FL_EXPORT void glutWireDodecahedron();
FL_EXPORT void glutSolidDodecahedron();
FL_EXPORT void glutWireTeapot(GLdouble size);
FL_EXPORT void glutSolidTeapot(GLdouble size);
FL_EXPORT void glutWireOctahedron();
FL_EXPORT void glutSolidOctahedron();
FL_EXPORT void glutWireTetrahedron();
FL_EXPORT void glutSolidTetrahedron();
FL_EXPORT void glutWireIcosahedron();
FL_EXPORT void glutSolidIcosahedron();

#endif // !_FL_glut_H_
```

14.202 mac.H File Reference

Mac OS X-specific symbols.

Classes

- class Fl_Mac_App_Menu

Functions

- Fl_Window ∗ **fl_mac_find** (FLWindow ∗)

  Returns the Fl_Window corresponding to the given macOS-specific window reference.
- CGContextRef **fl_mac_gc** ()

  Returns the macOS-specific graphics context for the current window.
- void fl_mac_set_about (Fl_Callback ∗cb, void ∗user_data, int shortcut=0)

  Attaches a callback to the "About myprog" item of the system application menu.
- FLWindow ∗ **fl_mac_xid** (const Fl_Window ∗win)

  Returns the macOS-specific window reference corresponding to the given Fl_Window object.

Variables

- int fl_mac_os_version

  The version number of the running Mac OS X (e.g., 100604 for 10.6.4, 101300 for 10.13, 140102 for 14.1.2).

14.202.1 Detailed Description

Mac OS X-specific symbols.

14.203 mac.H

Go to the documentation of this file.

```
//
// Mac header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2026 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

// Do not directly include this file, instead use <FL/platform.H>.  It will
// include this file if "__APPLE__" is defined.  This is to encourage
// portability of even the system-specific code...
#ifndef FL_DOXYGEN

#if !defined(FL_PLATFORM_H)
#  error "Never use <FL/mac.H> directly; include <FL/platform.H> instead."
#endif // !FL_PLATFORM_H

#ifdef __OBJC__
  @class NSOpenGLContext;
#  ifndef GL_SILENCE_DEPRECATION
#    define GL_SILENCE_DEPRECATION 1
#  endif
#elif defined(__cplusplus)
  class NSOpenGLContext;
#endif /* __OBJC__ */
extern NSOpenGLContext *fl_mac_glcontext(GLContext rc);

#ifdef __OBJC__
@class FLWindow; // a subclass of the NSWindow Cocoa class
typedef FLWindow *Window;
#else
typedef class FLWindow *Window; // pointer to the FLWindow objective-c class
#endif // __OBJC__

#include <FL/Fl_Widget.H> // for Fl_Callback

#if (defined(FL_LIBRARY) || defined(FL_INTERNALS)) // this part must be compiled when building the FLTK libraries

// Standard MacOS C/C++ includes...
#include <ApplicationServices/ApplicationServices.h>
#undef check // because of Fl::check()

#ifndef MAC_OS_X_VERSION_10_4
#define MAC_OS_X_VERSION_10_4 1040
#endif
#ifndef MAC_OS_X_VERSION_10_5
#define MAC_OS_X_VERSION_10_5 1050
#endif
#ifndef MAC_OS_X_VERSION_10_6
#define MAC_OS_X_VERSION_10_6 1060
#endif
#ifndef MAC_OS_X_VERSION_10_7
#define MAC_OS_X_VERSION_10_7 1070
#endif
#ifndef MAC_OS_X_VERSION_10_8
#define MAC_OS_X_VERSION_10_8 1080
#endif
#ifndef MAC_OS_X_VERSION_10_9
#define MAC_OS_X_VERSION_10_9 1090
#endif
#ifndef MAC_OS_X_VERSION_10_10
#define MAC_OS_X_VERSION_10_10 101000
#endif
#ifndef MAC_OS_X_VERSION_10_11
#define MAC_OS_X_VERSION_10_11 101100
#endif
#ifndef MAC_OS_X_VERSION_10_12
#define MAC_OS_X_VERSION_10_12 101200
#endif
#ifndef MAC_OS_X_VERSION_10_13
#define MAC_OS_X_VERSION_10_13 101300
#endif
#ifndef MAC_OS_X_VERSION_10_14
#define MAC_OS_X_VERSION_10_14 101400
```

```
#endif
#ifndef MAC_OS_X_VERSION_10_15
#define MAC_OS_X_VERSION_10_15 101500
#endif
#ifndef MAC_OS_X_VERSION_10_16
#define MAC_OS_X_VERSION_10_16 101600
#endif
#ifndef MAC_OS_VERSION_11_0
#define MAC_OS_VERSION_11_0 110000
#endif
#ifndef MAC_OS_VERSION_12_0
#define MAC_OS_VERSION_12_0 120000
#endif
#ifndef MAC_OS_VERSION_13_0
#define MAC_OS_VERSION_13_0 130000
#endif
#ifndef MAC_OS_VERSION_14_0
#define MAC_OS_VERSION_14_0 140000
#endif
#ifndef MAC_OS_VERSION_15_0
#define MAC_OS_VERSION_15_0 150000
#endif


#ifndef NSINTEGER_DEFINED // appears with 10.5 in NSObjCRuntime.h
#if defined(__LP64__) && __LP64__
typedef long NSInteger;
typedef unsigned long NSUInteger;
#else
typedef int NSInteger;
typedef unsigned int NSUInteger;
#endif
#endif

struct flCocoaRegion {
  int count;
  CGRect *rects;
}; // a region is the union of a series of rectangles

#ifndef CGFLOAT_DEFINED //appears with 10.5 in CGBase.h
#if defined(__LP64__) && __LP64__
typedef double CGFloat;
#else
typedef float CGFloat;
#endif
#endif // CGFLOAT_DEFINED

#else

typedef struct CGContext* CGContextRef;

#endif // FL_LIBRARY || FL_INTERNALS

extern CGContextRef fl_gc;

#endif // FL_DOXYGEN
void fl_mac_set_about(Fl_Callback *cb, void *user_data, int shortcut = 0);

extern CGContextRef fl_mac_gc();
extern FLWindow *fl_mac_xid(const Fl_Window *win);
extern Fl_Window *fl_mac_find(FLWindow *);
class Fl_Gl_Window;

extern int fl_mac_os_version;

struct Fl_Menu_Item;

class Fl_Mac_App_Menu {
public:
  static const char *about;
  static const char *print;
  static const char *print_no_titlebar;
  static const char *toggle_print_titlebar;
  static const char *services;
  static const char *hide;
  static const char *hide_others;
  static const char *show;
  static const char *quit;
  static void custom_application_menu_items(const Fl_Menu_Item *m);
};

```

14.204 math.h

```
//
```

```
// Math header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2020 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

// Xcode on macOS includes files by recursing down into directories.
// This code catches the cycle and directly includes the required file.
#ifdef fl_math_h_cyclic_include
#  include "/usr/include/math.h"
#endif

#ifndef fl_math_h
#  define fl_math_h

#  define fl_math_h_cyclic_include
#  include <math.h>
#  undef fl_math_h_cyclic_include

#  ifndef M_PI
#    define M_PI            3.14159265358979323846
#    define M_PI_2          1.57079632679489661923
#    define M_PI_4          0.78539816339744830962
#    define M_1_PI          0.31830988618379067154
#    define M_2_PI          0.63661977236758134308
#  endif // !M_PI

#  ifndef M_SQRT2
#    define M_SQRT2         1.41421356237309504880
#    define M_SQRT1_2       0.70710678118654752440
#  endif // !M_SQRT2

#  if (defined(_WIN32) || defined(CRAY)) && !defined(__MINGW32__)

inline double rint(double v) {return floor(v+.5);}
inline double copysign(double a, double b) {return b<0 ? -a : a;}

#  endif // (_WIN32 || CRAY) && !__MINGW32__

#endif // !fl_math_h
```

14.205 names.h File Reference

This file defines arrays of human readable names for FLTK symbolic constants.

```
#include <FL/Fl.H>
#include <string>
#include <map>
```

Functions

- std::string fl_callback_reason_str (int reason)

  Return the C++ symbol of an Fl_Callback_Reason as a string.
- std::string fl_eventname_str (int event)

  Return the C++ symbol of an Fl_Event as a string.
- std::string fl_fontname_str (int font)

  Return the C++ symbol of an Fl_Font as a string.

Variables

- const char *const fl_callback_reason_names [ ]

  This is an array of callback reason names you can use to convert callback reasons into names.
- const char *const fl_fontnames [ ]

  This is an array of font names you can use to convert font numbers into names.

14.205.1 Detailed Description

This file defines arrays of human readable names for FLTK symbolic constants.

14.206 names.h

Go to the documentation of this file.

```
00001 //
00002 // Event and other names header file for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 1998-2025 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file.  If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 //     https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 //     https://www.fltk.org/bugs.php
00015 //
00016
00017 // Thanks to Greg Ercolano for this addition.
00018
00024 #ifndef FL_NAMES_H
00025 #define FL_NAMES_H
00026
00027 #include <FL/Fl.H>  // for event constants
00028 #include <string>
00029 #include <map>
00030
00050 static std::map<int, const char*> fl_eventnames = {
00051   { FL_NO_EVENT, "FL_NO_EVENT" },
00052   { FL_PUSH, "FL_PUSH" },
00053   { FL_RELEASE, "FL_RELEASE" },
00054   { FL_ENTER, "FL_ENTER" },
00055   { FL_LEAVE, "FL_LEAVE" },
00056   { FL_DRAG, "FL_DRAG" },
00057   { FL_FOCUS, "FL_FOCUS" },
00058   { FL_UNFOCUS, "FL_UNFOCUS" },
00059   { FL_KEYDOWN, "FL_KEYDOWN" },
00060   { FL_KEYUP, "FL_KEYUP" },
00061   { FL_CLOSE, "FL_CLOSE" },
00062   { FL_MOVE, "FL_MOVE" },
00063   { FL_SHORTCUT, "FL_SHORTCUT" },
00064   { FL_DEACTIVATE, "FL_DEACTIVATE" },
00065   { FL_ACTIVATE, "FL_ACTIVATE" },
00066   { FL_HIDE, "FL_HIDE" },
00067   { FL_SHOW, "FL_SHOW" },
00068   { FL_PASTE, "FL_PASTE" },
00069   { FL_SELECTIONCLEAR, "FL_SELECTIONCLEAR" },
00070   { FL_MOUSEWHEEL, "FL_MOUSEWHEEL" },
00071   { FL_DND_ENTER, "FL_DND_ENTER" },
00072   { FL_DND_DRAG, "FL_DND_DRAG" },
00073   { FL_DND_LEAVE, "FL_DND_LEAVE" },
00074   { FL_DND_RELEASE, "FL_DND_RELEASE" },
00075   { FL_SCREEN_CONFIGURATION_CHANGED, "FL_SCREEN_CONFIGURATION_CHANGED" },
00076   { FL_FULLSCREEN, "FL_FULLSCREEN" },
00077   { FL_ZOOM_GESTURE, "FL_ZOOM_GESTURE" },
00078   { FL_ZOOM_EVENT, "FL_ZOOM_EVENT" },
00079   { FL_BEFORE_TOOLTIP, "FL_BEFORE_TOOLTIP" },
00080   { FL_BEFORE_MENU, "FL_BEFORE_MENU" },
00081   { FL_APP_ACTIVATE, "FL_APP_ACTIVATE" },
00082   { FL_APP_DEACTIVATE, "FL_APP_DEACTIVATE" },
00083   { Fl::Pen::DETECTED, "Fl::Pen::DETECTED" },
00084   { Fl::Pen::CHANGED, "Fl::Pen::CHANGED" },
00085   { Fl::Pen::IN_RANGE, "Fl::Pen::IN_RANGE" },
00086   { Fl::Pen::OUT_OF_RANGE, "Fl::Pen::OUT_OF_RANGE" },
00087   { Fl::Pen::ENTER, "Fl::Pen::ENTER" },
00088   { Fl::Pen::LEAVE, "Fl::Pen::LEAVE" },
00089   { Fl::Pen::TOUCH, "Fl::Pen::TOUCH" },
00090   { Fl::Pen::LIFT, "Fl::Pen::LIFT" },
00091   { Fl::Pen::HOVER, "Fl::Pen::HOVER" },
00092   { Fl::Pen::DRAW, "Fl::Pen::DRAW" },
00093   { Fl::Pen::BUTTON_PUSH, "Fl::Pen::BUTTON_PUSH" },
00094   { Fl::Pen::BUTTON_RELEASE, "Fl::Pen::BUTTON_RELEASE" }
00095 };
00096
00102 inline std::string fl_eventname_str(int event) {
00103   auto it = fl_eventnames.find(event);
00104   if (it == fl_eventnames.end()) {
00105     return "FL_EVENT_" + std::to_string(event);
00106   } else {
```

```
    return it->second;
  }
}


const char * const fl_fontnames[] =
{
  "FL_HELVETICA",
  "FL_HELVETICA_BOLD",
  "FL_HELVETICA_ITALIC",
  "FL_HELVETICA_BOLD_ITALIC",
  "FL_COURIER",
  "FL_COURIER_BOLD",
  "FL_COURIER_ITALIC",
  "FL_COURIER_BOLD_ITALIC",
  "FL_TIMES",
  "FL_TIMES_BOLD",
  "FL_TIMES_ITALIC",
  "FL_TIMES_BOLD_ITALIC",
  "FL_SYMBOL",
  "FL_SCREEN",
  "FL_SCREEN_BOLD",
  "FL_ZAPF_DINGBATS",
};

inline std::string fl_fontname_str(int font) {
  if ((font < 0) || (font >= FL_ZAPF_DINGBATS)) {
    return "FL_FONT_" + std::to_string(font);
  } else {
    return fl_fontnames[font];
  }
}


const char * const fl_callback_reason_names[] =
{
  "FL_REASON_UNKNOWN",
  "FL_REASON_SELECTED",
  "FL_REASON_DESELECTED",
  "FL_REASON_RESELECTED",
  "FL_REASON_OPENED",
  "FL_REASON_CLOSED",
  "FL_REASON_DRAGGED",
  "FL_REASON_CANCELLED",
  "FL_REASON_CHANGED",
  "FL_REASON_GOT_FOCUS",
  "FL_REASON_LOST_FOCUS",
  "FL_REASON_RELEASED",
  "FL_REASON_ENTER_KEY",
  NULL, NULL, NULL,
  NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
  NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
  "FL_REASON_USER", "FL_REASON_USER+1", "FL_REASON_USER+2", "FL_REASON_USER+3",
};

inline std::string fl_callback_reason_str(int reason) {
  if ((reason < 0) || (reason >= FL_REASON_USER+3) || (fl_callback_reason_names[reason] == nullptr)) {
    return "FL_REASON_" + std::to_string(reason);
  } else {
    return fl_callback_reason_names[reason];
  }
}

#endif /* FL_NAMES_H */
```

14.207 platform.H

```
//
// Platform header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2023 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

// These are FLTK symbols that are necessary or useful for calling
// platform specific functions. This file #include's certain platform
```

```
// specific system header files that are necessary to declare platform
// specific FLTK functions, for instance "Windows.h" under Windows.

// You should include this file if (and ONLY if) you need to call
// platform specific functions directly.

// See FLTK documentation: chapter "Operating System Issues" on when
// you need to #include <FL/platform.H>

#if !defined(FL_PLATFORM_H) && !defined(FL_DOXYGEN)
#  define FL_PLATFORM_H

#   include <FL/Fl_Export.H>
#   include <FL/platform_types.h> // will bring in FL/fl_config.h
#   include <FL/fl_types.h> // for uchar
class Fl_Window;

#  ifdef _WIN32
#    include "win32.H"
#  elif defined(FLTK_USE_WAYLAND)
#    include "wayland.H"
#  elif defined(FLTK_USE_X11)
#    include "x11.H"
#  elif defined(__APPLE__)
#    include "mac.H"
#  endif // _WIN32

//
// cross-platform declarations
//
#if defined(FL_LIBRARY) || defined(FL_INTERNALS)
#  include <FL/Fl_Window.H>

class FL_EXPORT Fl_X {
public:
  fl_uintptr_t xid;
  Fl_Window* w;
  Fl_Region region;
  Fl_X *next;
  // static variables, static functions and member functions
  static Fl_X* first;
  static Fl_X* flx(const Fl_Window* w) {return w ? (Fl_X*)w->flx_ : 0;}
#  if defined(FLTK_USE_X11) && FLTK_USE_X11 // for backward compatibility
  static void make_xid(Fl_Window*, XVisualInfo* =fl_visual, Colormap=fl_colormap);
  static Fl_X* set_xid(Fl_Window*, Window);
  static inline Fl_X* i(const Fl_Window* w) {return flx(w);}
#  endif
};

inline Window fl_xid(const Fl_Window* w) { Fl_X *xTemp = Fl_X::flx(w); return xTemp ?
(Window)xTemp->xid : 0; }
#else
extern FL_EXPORT Window fl_xid_(const Fl_Window* w);
#  define fl_xid(w) fl_xid_(w)
#endif // FL_LIBRARY || FL_INTERNALS

extern FL_EXPORT Fl_Window* fl_find(Window xid);
extern FL_EXPORT void fl_open_display();
extern FL_EXPORT void fl_close_display();
extern FL_EXPORT Window fl_window;
extern FL_EXPORT int fl_parse_color(const char* p, uchar& r, uchar& g, uchar& b);
extern FL_EXPORT void fl_open_callback(void (*)(const char *));

#endif // !FL_PLATFORM_H
```

14.208 platform_types.h File Reference

Definitions of platform-dependent types.

Macros

- #define **FL_COMMAND** opaque

  An alias for FL_CTRL on Windows and X11, or FL_META on MacOS X.
- #define **FL_CONTROL** opaque

  An alias for FL_META on Windows and X11, or FL_CTRL on MacOS X.

Typedefs

- typedef opaque fl_intptr_t

An integral type large enough to store a pointer or a long value.

- typedef opaque Fl_Offscreen

  Platform-specific value representing an offscreen drawing buffer.

- typedef struct opaque ∗ Fl_Region

  Pointer to a platform-specific structure representing a collection of rectangles.

- typedef opaque **FL_SOCKET**

  socket or file descriptor

- typedef opaque Fl_Timestamp

  Platform-specific point in time, used for delta time calculation.

- typedef opaque fl_uintptr_t

  An unsigned integral type large enough to store a pointer or an unsigned long value.

- typedef struct opaque ∗ GLContext

  Pointer to a platform-specific structure representing the window's OpenGL rendering context.

14.208.1 Detailed Description

Definitions of platform-dependent types.
The exact nature of these types varies with the platform. Therefore, portable FLTK applications should not assume these types have a specific size, or that they are pointers.

14.208.2 Typedef Documentation

14.208.2.1 fl_intptr_t

`typedef opaque fl_intptr_t`
An integral type large enough to store a pointer or a long value.
A pointer value can be safely cast to fl_intptr_t, and later cast back to its initial pointer type without change to the pointer value. A variable of type fl_intptr_t can also store a long int value.

14.208.2.2 Fl_Offscreen

`typedef opaque Fl_Offscreen`
Platform-specific value representing an offscreen drawing buffer.

Note

This value can be safely cast to these types on each platform:

- X11: Pixmap
- Wayland: cairo_t ∗
- Windows: HBITMAP
- macOS: CGContextRef

14.208.2.3 Fl_Region

`typedef struct opaque* Fl_Region`
Pointer to a platform-specific structure representing a collection of rectangles.

Note

This pointer can be safely cast to these types on each platform:

- X11: Region as defined by X11
- Wayland: cairo_region_t ∗
- Windows: HRGN
- macOS: struct flCocoaRegion ∗

14.208.2.4 Fl_Timestamp

`typedef opaque Fl_Timestamp`

Platform-specific point in time, used for delta time calculation.

Note

This type may be a struct. sizeof(Fl_Timestamp) may be different on different platforms. Fl_Timestamp may change with future ABI changes.

14.208.2.5 fl_uintptr_t

`typedef opaque fl_uintptr_t`

An unsigned integral type large enough to store a pointer or an unsigned long value.
A pointer value can be safely cast to fl_uintptr_t, and later cast back to its initial pointer type without change to the pointer value. A variable of type fl_uintptr_t can also store an unsigned long int value.

14.208.2.6 GLContext

`typedef struct opaque* GLContext`

Pointer to a platform-specific structure representing the window's OpenGL rendering context.

Note

This pointer can be safely cast to these types on each platform:

- X11: GLXContext
- Wayland: EGLContext
- Windows: HGLRC
- macOS: NSOpenGLContext *

14.209 platform_types.h

Go to the documentation of this file.

```
00001 /*
00002  * Copyright 2016-2026 by Bill Spitzak and others.
00003  *
00004  * This library is free software. Distribution and use rights are outlined in
00005  * the file "COPYING" which should have been included with this file.  If this
00006  * file is missing or damaged, see the license at:
00007  *
00008  *     https://www.fltk.org/COPYING.php
00009  *
00010  * Please see the following page on how to report bugs and issues:
00011  *
00012  *     https://www.fltk.org/bugs.php
00013  */
00014
00015 #ifndef Fl_Platform_Types_H
00016 #define Fl_Platform_Types_H
00017
00025 #ifdef FL_DOXYGEN
00026
00031 typedef opaque fl_intptr_t;
00036 typedef opaque fl_uintptr_t;
00037
00046 typedef opaque Fl_Offscreen;
00047
00056 typedef struct opaque *Fl_Region;
00057 typedef opaque FL_SOCKET;
00066 typedef struct opaque *GLContext;
00067
00073 typedef opaque Fl_Timestamp;
00074
00075 #  define FL_COMMAND  opaque
00076 #  define FL_CONTROL  opaque
00078 #else /* FL_DOXYGEN */
00079
00080 #ifndef FL_PLATFORM_TYPES_H
00081 #define FL_PLATFORM_TYPES_H
00082
```

```
#include <FL/fl_config.h>
#include <time.h> // for time_t

/* Platform-dependent types are defined here.
  These types must be defined by any platform:
  FL_SOCKET, struct dirent, fl_intptr_t, fl_uintptr_t

  NOTE: *FIXME* AlbrechtS 13 Apr 2016 (concerning FL_SOCKET)
  ----------------------------------------------------------
    The Fl::add_fd() API is partially inconsistent because some of the methods
    explicitly use 'int', but the callback typedefs use FL_SOCKET. With the
    definition of FL_SOCKET below we can have different data sizes and
    different signedness of socket numbers on *some* platforms.
 */

#ifdef _WIN64

#if defined(_MSC_VER) && (_MSC_VER < 1600)
# include <stddef.h>  /* stdint.h not available before VS 2010 (1600) */
#else
# include <stdint.h>
#endif

typedef intptr_t fl_intptr_t;
typedef uintptr_t fl_uintptr_t;

#else /* ! _WIN64 */

typedef long fl_intptr_t;
typedef unsigned long fl_uintptr_t;

#endif /* _WIN64 */

typedef void *GLContext;
typedef void *Fl_Region;
typedef fl_uintptr_t  Fl_Offscreen;

/* Allows all hybrid combinations except WIN32 + X11 with MSVC */
#if defined(_WIN32) && !defined(__MINGW32__)
  struct dirent {char d_name[1];};
#else
#  include <dirent.h>
#endif

# if defined(_WIN64) && defined(_MSC_VER)
typedef  unsigned __int64 FL_SOCKET;    /* *FIXME* - FL_SOCKET (see above) */
# else
typedef  int FL_SOCKET;
# endif

#endif /* FL_PLATFORM_TYPES_H */

/* This is currently the same for all platforms but may change in the future */
struct Fl_Timestamp_t {
  time_t sec;
  int usec;
};

typedef struct Fl_Timestamp_t Fl_Timestamp;

#endif /* FL_DOXYGEN */

#endif /* Fl_Platform_Types_H */

```

14.210 rgb_colors.H

```
//
// Extra color names for the Fast Light Tool Kit (FLTK).
//
// Copyright 2026 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

// The source of color information was rgb.txt.

```

```
const Fl_Color FL_RGB_ALICE_BLUE             = 0xF0F8FF00; // (240, 248, 255)
const Fl_Color FL_RGB_ANTIQUE_WHITE          = 0xFAEBD700; // (250, 235, 215)
const Fl_Color FL_RGB_AQUAMARINE             = 0x7FFFD400; // (127, 255, 212)
const Fl_Color FL_RGB_AQUAMARINE1            = 0x7FFFD400; // (127, 255, 212)
const Fl_Color FL_RGB_AQUAMARINE2            = 0x76EEC600; // (118, 238, 198)
const Fl_Color FL_RGB_AQUAMARINE3            = 0x66CDAA00; // (102, 205, 170)
const Fl_Color FL_RGB_AQUAMARINE4            = 0x458B7400; // ( 69, 139, 116)
const Fl_Color FL_RGB_AZURE                  = 0xF0FFFF00; // (240, 255, 255)
const Fl_Color FL_RGB_AZURE1                 = 0xF0FFFF00; // (240, 255, 255)
const Fl_Color FL_RGB_AZURE2                 = 0xE0EEEE00; // (224, 238, 238)
const Fl_Color FL_RGB_AZURE3                 = 0xC1CDCD00; // (193, 205, 205)
const Fl_Color FL_RGB_AZURE4                 = 0x838B8B00; // (131, 139, 139)
const Fl_Color FL_RGB_BEIGE                  = 0xF5F5DC00; // (245, 245, 220)
const Fl_Color FL_RGB_BISQUE                 = 0xFFE4C400; // (255, 228, 196)
const Fl_Color FL_RGB_BISQUE1                = 0xFFE4C400; // (255, 228, 196)
const Fl_Color FL_RGB_BISQUE2                = 0xEED5B700; // (238, 213, 183)
const Fl_Color FL_RGB_BISQUE3                = 0xCDB79E00; // (205, 183, 158)
const Fl_Color FL_RGB_BISQUE4                = 0x8B7D6B00; // (139, 125, 107)
const Fl_Color FL_RGB_BLACK                  = 0x00000000; // (  0,   0,   0)
const Fl_Color FL_RGB_BLANCHED_ALMOND        = 0xFFEBCD00; // (255, 235, 205)
const Fl_Color FL_RGB_BLUE                   = 0x0000FF00; // (  0,   0, 255)
const Fl_Color FL_RGB_BLUE1                  = 0x0000FF00; // (  0,   0, 255)
const Fl_Color FL_RGB_BLUE2                  = 0x0000EE00; // (  0,   0, 238)
const Fl_Color FL_RGB_BLUE3                  = 0x0000CD00; // (  0,   0, 205)
const Fl_Color FL_RGB_BLUE4                  = 0x00008B00; // (  0,   0, 139)
const Fl_Color FL_RGB_BLUE_VIOLET            = 0x8A2BE200; // (138,  43, 226)
const Fl_Color FL_RGB_BROWN                  = 0xA52A2A00; // (165,  42,  42)
const Fl_Color FL_RGB_BROWN1                 = 0xFF404000; // (255,  64,  64)
const Fl_Color FL_RGB_BROWN2                 = 0xEE3B3B00; // (238,  59,  59)
const Fl_Color FL_RGB_BROWN3                 = 0xCD333300; // (205,  51,  51)
const Fl_Color FL_RGB_BROWN4                 = 0x8B232300; // (139,  35,  35)
const Fl_Color FL_RGB_BURLYWOOD              = 0xDEB88700; // (222, 184, 135)
const Fl_Color FL_RGB_BURLYWOOD1             = 0xFFD39B00; // (255, 211, 155)
const Fl_Color FL_RGB_BURLYWOOD2             = 0xEEC59100; // (238, 197, 145)
const Fl_Color FL_RGB_BURLYWOOD3             = 0xCDAA7D00; // (205, 170, 125)
const Fl_Color FL_RGB_BURLYWOOD4             = 0x8B735500; // (139, 115,  85)
const Fl_Color FL_RGB_CADET_BLUE             = 0x5F9EA000; // ( 95, 158, 160)
const Fl_Color FL_RGB_CHARTREUSE             = 0x7FFF0000; // (127, 255,   0)
const Fl_Color FL_RGB_CHARTREUSE1            = 0x7FFF0000; // (127, 255,   0)
const Fl_Color FL_RGB_CHARTREUSE2            = 0x76EE0000; // (118, 238,   0)
const Fl_Color FL_RGB_CHARTREUSE3            = 0x66CD0000; // (102, 205,   0)
const Fl_Color FL_RGB_CHARTREUSE4            = 0x458B0000; // ( 69, 139,   0)
const Fl_Color FL_RGB_CHOCOLATE              = 0xD2691E00; // (210, 105,  30)
const Fl_Color FL_RGB_CHOCOLATE1             = 0xFF7F2400; // (255, 127,  36)
const Fl_Color FL_RGB_CHOCOLATE2             = 0xEE762100; // (238, 118,  33)
const Fl_Color FL_RGB_CHOCOLATE3             = 0xCD661D00; // (205, 102,  29)
const Fl_Color FL_RGB_CHOCOLATE4             = 0x8B451300; // (139,  69,  19)
const Fl_Color FL_RGB_CORAL                  = 0xFF7F5000; // (255, 127,  80)
const Fl_Color FL_RGB_CORAL1                 = 0xFF725600; // (255, 114,  86)
const Fl_Color FL_RGB_CORAL2                 = 0xEE6A5000; // (238, 106,  80)
const Fl_Color FL_RGB_CORAL3                 = 0xCD5B4500; // (205,  91,  69)
const Fl_Color FL_RGB_CORAL4                 = 0x8B3E2F00; // (139,  62,  47)
const Fl_Color FL_RGB_CORNFLOWER_BLUE        = 0x6495ED00; // (100, 149, 237)
const Fl_Color FL_RGB_CORNSILK               = 0xFFF8DC00; // (255, 248, 220)
const Fl_Color FL_RGB_CORNSILK1              = 0xFFF8DC00; // (255, 248, 220)
const Fl_Color FL_RGB_CORNSILK2              = 0xEEE8CD00; // (238, 232, 205)
const Fl_Color FL_RGB_CORNSILK3              = 0xCDC8B100; // (205, 200, 177)
const Fl_Color FL_RGB_CORNSILK4              = 0x8B887800; // (139, 136, 120)
const Fl_Color FL_RGB_CYAN                   = 0x00FFFF00; // (  0, 255, 255)
const Fl_Color FL_RGB_CYAN1                  = 0x00FFFF00; // (  0, 255, 255)
const Fl_Color FL_RGB_CYAN2                  = 0x00EEEE00; // (  0, 238, 238)
const Fl_Color FL_RGB_CYAN3                  = 0x00CDCD00; // (  0, 205, 205)
const Fl_Color FL_RGB_CYAN4                  = 0x008B8B00; // (  0, 139, 139)
const Fl_Color FL_RGB_DARK_BLUE              = 0x00008B00; // (  0,   0, 139)
const Fl_Color FL_RGB_DARK_CYAN              = 0x008B8B00; // (  0, 139, 139)
const Fl_Color FL_RGB_DARK_GOLDENROD         = 0xB8860B00; // (184, 134,  11)
const Fl_Color FL_RGB_DARK_GRAY              = 0xA9A9A900; // (169, 169, 169)
const Fl_Color FL_RGB_DARK_GREEN             = 0x00640000; // (  0, 100,   0)
const Fl_Color FL_RGB_DARK_GREY              = 0xA9A9A900; // (169, 169, 169)
const Fl_Color FL_RGB_DARK_KHAKI             = 0xBDB76B00; // (189, 183, 107)
const Fl_Color FL_RGB_DARK_MAGENTA           = 0x8B008B00; // (139,   0, 139)
const Fl_Color FL_RGB_DARK_OLIVE_GREEN       = 0x556B2F00; // ( 85, 107,  47)
const Fl_Color FL_RGB_DARK_ORANGE            = 0xFF8C0000; // (255, 140,   0)
const Fl_Color FL_RGB_DARK_ORCHID            = 0x9932CC00; // (153,  50, 204)
const Fl_Color FL_RGB_DARK_RED               = 0x8B000000; // (139,   0,   0)
const Fl_Color FL_RGB_DARK_SALMON            = 0xE9967A00; // (233, 150, 122)
const Fl_Color FL_RGB_DARK_SEA_GREEN         = 0x8FBC8F00; // (143, 188, 143)
const Fl_Color FL_RGB_DARK_SLATE_BLUE        = 0x483D8B00; // ( 72,  61, 139)
const Fl_Color FL_RGB_DARK_SLATE_GRAY        = 0x2F4F4F00; // ( 47,  79,  79)
const Fl_Color FL_RGB_DARK_SLATE_GREY        = 0x2F4F4F00; // ( 47,  79,  79)
const Fl_Color FL_RGB_DARK_TURQUOISE         = 0x00CED100; // (  0, 206, 209)
const Fl_Color FL_RGB_DARK_VIOLET            = 0x9400D300; // (148,   0, 211)
const Fl_Color FL_RGB_DEBIAN_RED             = 0xD7075100; // (215,   7,  81)
const Fl_Color FL_RGB_DEEP_PINK              = 0xFF149300; // (255,  20, 147)
const Fl_Color FL_RGB_DEEP_SKY_BLUE          = 0x00BFFF00; // (  0, 191, 255)
const Fl_Color FL_RGB_DIM_GRAY               = 0x69696900; // (105, 105, 105)
const Fl_Color FL_RGB_DIM_GREY               = 0x69696900; // (105, 105, 105)
```

```
const Fl_Color FL_RGB_DODGER_BLUE            = 0x1E90FF00; // ( 30, 144, 255)
const Fl_Color FL_RGB_FIREBRICK              = 0xB2222200; // (178,  34,  34)
const Fl_Color FL_RGB_FIREBRICK1             = 0xFF303000; // (255,  48,  48)
const Fl_Color FL_RGB_FIREBRICK2             = 0xEE2C2C00; // (238,  44,  44)
const Fl_Color FL_RGB_FIREBRICK3             = 0xCD262600; // (205,  38,  38)
const Fl_Color FL_RGB_FIREBRICK4             = 0x8B1A1A00; // (139,  26,  26)
const Fl_Color FL_RGB_FLORAL_WHITE           = 0xFFFAF000; // (255, 250, 240)
const Fl_Color FL_RGB_FOREST_GREEN           = 0x228B2200; // ( 34, 139,  34)
const Fl_Color FL_RGB_GAINSBORO              = 0xDCDCDC00; // (220, 220, 220)
const Fl_Color FL_RGB_GHOST_WHITE            = 0xF8F8FF00; // (248, 248, 255)
const Fl_Color FL_RGB_GOLD                   = 0xFFD70000; // (255, 215,   0)
const Fl_Color FL_RGB_GOLD1                  = 0xFFD70000; // (255, 215,   0)
const Fl_Color FL_RGB_GOLD2                  = 0xEEC90000; // (238, 201,   0)
const Fl_Color FL_RGB_GOLD3                  = 0xCDAD0000; // (205, 173,   0)
const Fl_Color FL_RGB_GOLD4                  = 0x8B750000; // (139, 117,   0)
const Fl_Color FL_RGB_GOLDENROD              = 0xDAA52000; // (218, 165,  32)
const Fl_Color FL_RGB_GOLDENROD1             = 0xFFC12500; // (255, 193,  37)
const Fl_Color FL_RGB_GOLDENROD2             = 0xEEB42200; // (238, 180,  34)
const Fl_Color FL_RGB_GOLDENROD3             = 0xCD9B1D00; // (205, 155,  29)
const Fl_Color FL_RGB_GOLDENROD4             = 0x8B691400; // (139, 105,  20)
const Fl_Color FL_RGB_GRAY                   = 0xBEBEBE00; // (190, 190, 190)
const Fl_Color FL_RGB_GRAY0                  = 0x00000000; // (  0,   0,   0)
const Fl_Color FL_RGB_GRAY1                  = 0x03030300; // (  3,   3,   3)
const Fl_Color FL_RGB_GRAY10                 = 0x1A1A1A00; // ( 26,  26,  26)
const Fl_Color FL_RGB_GRAY100                = 0xFFFFFF00; // (255, 255, 255)
const Fl_Color FL_RGB_GRAY11                 = 0x1C1C1C00; // ( 28,  28,  28)
const Fl_Color FL_RGB_GRAY12                 = 0x1F1F1F00; // ( 31,  31,  31)
const Fl_Color FL_RGB_GRAY13                 = 0x21212100; // ( 33,  33,  33)
const Fl_Color FL_RGB_GRAY14                 = 0x24242400; // ( 36,  36,  36)
const Fl_Color FL_RGB_GRAY15                 = 0x26262600; // ( 38,  38,  38)
const Fl_Color FL_RGB_GRAY16                 = 0x29292900; // ( 41,  41,  41)
const Fl_Color FL_RGB_GRAY17                 = 0x2B2B2B00; // ( 43,  43,  43)
const Fl_Color FL_RGB_GRAY18                 = 0x2E2E2E00; // ( 46,  46,  46)
const Fl_Color FL_RGB_GRAY19                 = 0x30303000; // ( 48,  48,  48)
const Fl_Color FL_RGB_GRAY2                  = 0x05050500; // (  5,   5,   5)
const Fl_Color FL_RGB_GRAY20                 = 0x33333300; // ( 51,  51,  51)
const Fl_Color FL_RGB_GRAY21                 = 0x36363600; // ( 54,  54,  54)
const Fl_Color FL_RGB_GRAY22                 = 0x38383800; // ( 56,  56,  56)
const Fl_Color FL_RGB_GRAY23                 = 0x3B3B3B00; // ( 59,  59,  59)
const Fl_Color FL_RGB_GRAY24                 = 0x3D3D3D00; // ( 61,  61,  61)
const Fl_Color FL_RGB_GRAY25                 = 0x40404000; // ( 64,  64,  64)
const Fl_Color FL_RGB_GRAY26                 = 0x42424200; // ( 66,  66,  66)
const Fl_Color FL_RGB_GRAY27                 = 0x45454500; // ( 69,  69,  69)
const Fl_Color FL_RGB_GRAY28                 = 0x47474700; // ( 71,  71,  71)
const Fl_Color FL_RGB_GRAY29                 = 0x4A4A4A00; // ( 74,  74,  74)
const Fl_Color FL_RGB_GRAY3                  = 0x08080800; // (  8,   8,   8)
const Fl_Color FL_RGB_GRAY30                 = 0x4D4D4D00; // ( 77,  77,  77)
const Fl_Color FL_RGB_GRAY31                 = 0x4F4F4F00; // ( 79,  79,  79)
const Fl_Color FL_RGB_GRAY32                 = 0x52525200; // ( 82,  82,  82)
const Fl_Color FL_RGB_GRAY33                 = 0x54545400; // ( 84,  84,  84)
const Fl_Color FL_RGB_GRAY34                 = 0x57575700; // ( 87,  87,  87)
const Fl_Color FL_RGB_GRAY35                 = 0x59595900; // ( 89,  89,  89)
const Fl_Color FL_RGB_GRAY36                 = 0x5C5C5C00; // ( 92,  92,  92)
const Fl_Color FL_RGB_GRAY37                 = 0x5E5E5E00; // ( 94,  94,  94)
const Fl_Color FL_RGB_GRAY38                 = 0x61616100; // ( 97,  97,  97)
const Fl_Color FL_RGB_GRAY39                 = 0x63636300; // ( 99,  99,  99)
const Fl_Color FL_RGB_GRAY4                  = 0x0A0A0A00; // ( 10,  10,  10)
const Fl_Color FL_RGB_GRAY40                 = 0x66666600; // (102, 102, 102)
const Fl_Color FL_RGB_GRAY41                 = 0x69696900; // (105, 105, 105)
const Fl_Color FL_RGB_GRAY42                 = 0x6B6B6B00; // (107, 107, 107)
const Fl_Color FL_RGB_GRAY43                 = 0x6E6E6E00; // (110, 110, 110)
const Fl_Color FL_RGB_GRAY44                 = 0x70707000; // (112, 112, 112)
const Fl_Color FL_RGB_GRAY45                 = 0x73737300; // (115, 115, 115)
const Fl_Color FL_RGB_GRAY46                 = 0x75757500; // (117, 117, 117)
const Fl_Color FL_RGB_GRAY47                 = 0x78787800; // (120, 120, 120)
const Fl_Color FL_RGB_GRAY48                 = 0x7A7A7A00; // (122, 122, 122)
const Fl_Color FL_RGB_GRAY49                 = 0x7D7D7D00; // (125, 125, 125)
const Fl_Color FL_RGB_GRAY5                  = 0x0D0D0D00; // ( 13,  13,  13)
const Fl_Color FL_RGB_GRAY50                 = 0x7F7F7F00; // (127, 127, 127)
const Fl_Color FL_RGB_GRAY51                 = 0x82828200; // (130, 130, 130)
const Fl_Color FL_RGB_GRAY52                 = 0x85858500; // (133, 133, 133)
const Fl_Color FL_RGB_GRAY53                 = 0x87878700; // (135, 135, 135)
const Fl_Color FL_RGB_GRAY54                 = 0x8A8A8A00; // (138, 138, 138)
const Fl_Color FL_RGB_GRAY55                 = 0x8C8C8C00; // (140, 140, 140)
const Fl_Color FL_RGB_GRAY56                 = 0x8F8F8F00; // (143, 143, 143)
const Fl_Color FL_RGB_GRAY57                 = 0x91919100; // (145, 145, 145)
const Fl_Color FL_RGB_GRAY58                 = 0x94949400; // (148, 148, 148)
const Fl_Color FL_RGB_GRAY59                 = 0x96969600; // (150, 150, 150)
const Fl_Color FL_RGB_GRAY6                  = 0x0F0F0F00; // ( 15,  15,  15)
const Fl_Color FL_RGB_GRAY60                 = 0x99999900; // (153, 153, 153)
const Fl_Color FL_RGB_GRAY61                 = 0x9C9C9C00; // (156, 156, 156)
const Fl_Color FL_RGB_GRAY62                 = 0x9E9E9E00; // (158, 158, 158)
const Fl_Color FL_RGB_GRAY63                 = 0xA1A1A100; // (161, 161, 161)
const Fl_Color FL_RGB_GRAY64                 = 0xA3A3A300; // (163, 163, 163)
const Fl_Color FL_RGB_GRAY65                 = 0xA6A6A600; // (166, 166, 166)
const Fl_Color FL_RGB_GRAY66                 = 0xA8A8A800; // (168, 168, 168)
const Fl_Color FL_RGB_GRAY67                 = 0xABABAB00; // (171, 171, 171)
```

```
const Fl_Color FL_RGB_GRAY68                = 0xADADAD00; // (173, 173, 173)
const Fl_Color FL_RGB_GRAY69                = 0xB0B0B000; // (176, 176, 176)
const Fl_Color FL_RGB_GRAY7                 = 0x12121200; // ( 18,  18,  18)
const Fl_Color FL_RGB_GRAY70                = 0xB3B3B300; // (179, 179, 179)
const Fl_Color FL_RGB_GRAY71                = 0xB5B5B500; // (181, 181, 181)
const Fl_Color FL_RGB_GRAY72                = 0xB8B8B800; // (184, 184, 184)
const Fl_Color FL_RGB_GRAY73                = 0xBABABA00; // (186, 186, 186)
const Fl_Color FL_RGB_GRAY74                = 0xBDBDBD00; // (189, 189, 189)
const Fl_Color FL_RGB_GRAY75                = 0xBFBFBF00; // (191, 191, 191)
const Fl_Color FL_RGB_GRAY76                = 0xC2C2C200; // (194, 194, 194)
const Fl_Color FL_RGB_GRAY77                = 0xC4C4C400; // (196, 196, 196)
const Fl_Color FL_RGB_GRAY78                = 0xC7C7C700; // (199, 199, 199)
const Fl_Color FL_RGB_GRAY79                = 0xC9C9C900; // (201, 201, 201)
const Fl_Color FL_RGB_GRAY8                 = 0x14141400; // ( 20,  20,  20)
const Fl_Color FL_RGB_GRAY80                = 0xCCCCCC00; // (204, 204, 204)
const Fl_Color FL_RGB_GRAY81                = 0xCFCFCF00; // (207, 207, 207)
const Fl_Color FL_RGB_GRAY82                = 0xD1D1D100; // (209, 209, 209)
const Fl_Color FL_RGB_GRAY83                = 0xD4D4D400; // (212, 212, 212)
const Fl_Color FL_RGB_GRAY84                = 0xD6D6D600; // (214, 214, 214)
const Fl_Color FL_RGB_GRAY85                = 0xD9D9D900; // (217, 217, 217)
const Fl_Color FL_RGB_GRAY86                = 0xDBDBDB00; // (219, 219, 219)
const Fl_Color FL_RGB_GRAY87                = 0xDEDEDE00; // (222, 222, 222)
const Fl_Color FL_RGB_GRAY88                = 0xE0E0E000; // (224, 224, 224)
const Fl_Color FL_RGB_GRAY89                = 0xE3E3E300; // (227, 227, 227)
const Fl_Color FL_RGB_GRAY9                 = 0x17171700; // ( 23,  23,  23)
const Fl_Color FL_RGB_GRAY90                = 0xE5E5E500; // (229, 229, 229)
const Fl_Color FL_RGB_GRAY91                = 0xE8E8E800; // (232, 232, 232)
const Fl_Color FL_RGB_GRAY92                = 0xEBEBEB00; // (235, 235, 235)
const Fl_Color FL_RGB_GRAY93                = 0xEDEDED00; // (237, 237, 237)
const Fl_Color FL_RGB_GRAY94                = 0xF0F0F000; // (240, 240, 240)
const Fl_Color FL_RGB_GRAY95                = 0xF2F2F200; // (242, 242, 242)
const Fl_Color FL_RGB_GRAY96                = 0xF5F5F500; // (245, 245, 245)
const Fl_Color FL_RGB_GRAY97                = 0xF7F7F700; // (247, 247, 247)
const Fl_Color FL_RGB_GRAY98                = 0xFAFAFA00; // (250, 250, 250)
const Fl_Color FL_RGB_GRAY99                = 0xFCFCFC00; // (252, 252, 252)
const Fl_Color FL_RGB_GREEN                 = 0x00FF0000; // (  0, 255,   0)
const Fl_Color FL_RGB_GREEN1                = 0x00FF0000; // (  0, 255,   0)
const Fl_Color FL_RGB_GREEN2                = 0x00EE0000; // (  0, 238,   0)
const Fl_Color FL_RGB_GREEN3                = 0x00CD0000; // (  0, 205,   0)
const Fl_Color FL_RGB_GREEN4                = 0x008B0000; // (  0, 139,   0)
const Fl_Color FL_RGB_GREEN_YELLOW          = 0xADFF2F00; // (173, 255,  47)
const Fl_Color FL_RGB_GREY                  = 0xBEBEBE00; // (190, 190, 190)
const Fl_Color FL_RGB_GREY0                 = 0x00000000; // (  0,   0,   0)
const Fl_Color FL_RGB_GREY1                 = 0x03030300; // (  3,   3,   3)
const Fl_Color FL_RGB_GREY10                = 0x1A1A1A00; // ( 26,  26,  26)
const Fl_Color FL_RGB_GREY100               = 0xFFFFFF00; // (255, 255, 255)
const Fl_Color FL_RGB_GREY11                = 0x1C1C1C00; // ( 28,  28,  28)
const Fl_Color FL_RGB_GREY12                = 0x1F1F1F00; // ( 31,  31,  31)
const Fl_Color FL_RGB_GREY13                = 0x21212100; // ( 33,  33,  33)
const Fl_Color FL_RGB_GREY14                = 0x24242400; // ( 36,  36,  36)
const Fl_Color FL_RGB_GREY15                = 0x26262600; // ( 38,  38,  38)
const Fl_Color FL_RGB_GREY16                = 0x29292900; // ( 41,  41,  41)
const Fl_Color FL_RGB_GREY17                = 0x2B2B2B00; // ( 43,  43,  43)
const Fl_Color FL_RGB_GREY18                = 0x2E2E2E00; // ( 46,  46,  46)
const Fl_Color FL_RGB_GREY19                = 0x30303000; // ( 48,  48,  48)
const Fl_Color FL_RGB_GREY2                 = 0x05050500; // (  5,   5,   5)
const Fl_Color FL_RGB_GREY20                = 0x33333300; // ( 51,  51,  51)
const Fl_Color FL_RGB_GREY21                = 0x36363600; // ( 54,  54,  54)
const Fl_Color FL_RGB_GREY22                = 0x38383800; // ( 56,  56,  56)
const Fl_Color FL_RGB_GREY23                = 0x3B3B3B00; // ( 59,  59,  59)
const Fl_Color FL_RGB_GREY24                = 0x3D3D3D00; // ( 61,  61,  61)
const Fl_Color FL_RGB_GREY25                = 0x40404000; // ( 64,  64,  64)
const Fl_Color FL_RGB_GREY26                = 0x42424200; // ( 66,  66,  66)
const Fl_Color FL_RGB_GREY27                = 0x45454500; // ( 69,  69,  69)
const Fl_Color FL_RGB_GREY28                = 0x47474700; // ( 71,  71,  71)
const Fl_Color FL_RGB_GREY29                = 0x4A4A4A00; // ( 74,  74,  74)
const Fl_Color FL_RGB_GREY3                 = 0x08080800; // (  8,   8,   8)
const Fl_Color FL_RGB_GREY30                = 0x4D4D4D00; // ( 77,  77,  77)
const Fl_Color FL_RGB_GREY31                = 0x4F4F4F00; // ( 79,  79,  79)
const Fl_Color FL_RGB_GREY32                = 0x52525200; // ( 82,  82,  82)
const Fl_Color FL_RGB_GREY33                = 0x54545400; // ( 84,  84,  84)
const Fl_Color FL_RGB_GREY34                = 0x57575700; // ( 87,  87,  87)
const Fl_Color FL_RGB_GREY35                = 0x59595900; // ( 89,  89,  89)
const Fl_Color FL_RGB_GREY36                = 0x5C5C5C00; // ( 92,  92,  92)
const Fl_Color FL_RGB_GREY37                = 0x5E5E5E00; // ( 94,  94,  94)
const Fl_Color FL_RGB_GREY38                = 0x61616100; // ( 97,  97,  97)
const Fl_Color FL_RGB_GREY39                = 0x63636300; // ( 99,  99,  99)
const Fl_Color FL_RGB_GREY4                 = 0x0A0A0A00; // ( 10,  10,  10)
const Fl_Color FL_RGB_GREY40                = 0x66666600; // (102, 102, 102)
const Fl_Color FL_RGB_GREY41                = 0x69696900; // (105, 105, 105)
const Fl_Color FL_RGB_GREY42                = 0x6B6B6B00; // (107, 107, 107)
const Fl_Color FL_RGB_GREY43                = 0x6E6E6E00; // (110, 110, 110)
const Fl_Color FL_RGB_GREY44                = 0x70707000; // (112, 112, 112)
const Fl_Color FL_RGB_GREY45                = 0x73737300; // (115, 115, 115)
const Fl_Color FL_RGB_GREY46                = 0x75757500; // (117, 117, 117)
const Fl_Color FL_RGB_GREY47                = 0x78787800; // (120, 120, 120)
const Fl_Color FL_RGB_GREY48                = 0x7A7A7A00; // (122, 122, 122)
```

```
const Fl_Color FL_RGB_GREY49                 = 0x7D7D7D00; // (125, 125, 125)
const Fl_Color FL_RGB_GREY5                  = 0x0D0D0D00; // ( 13,  13,  13)
const Fl_Color FL_RGB_GREY50                 = 0x7F7F7F00; // (127, 127, 127)
const Fl_Color FL_RGB_GREY51                 = 0x82828200; // (130, 130, 130)
const Fl_Color FL_RGB_GREY52                 = 0x85858500; // (133, 133, 133)
const Fl_Color FL_RGB_GREY53                 = 0x87878700; // (135, 135, 135)
const Fl_Color FL_RGB_GREY54                 = 0x8A8A8A00; // (138, 138, 138)
const Fl_Color FL_RGB_GREY55                 = 0x8C8C8C00; // (140, 140, 140)
const Fl_Color FL_RGB_GREY56                 = 0x8F8F8F00; // (143, 143, 143)
const Fl_Color FL_RGB_GREY57                 = 0x91919100; // (145, 145, 145)
const Fl_Color FL_RGB_GREY58                 = 0x94949400; // (148, 148, 148)
const Fl_Color FL_RGB_GREY59                 = 0x96969600; // (150, 150, 150)
const Fl_Color FL_RGB_GREY6                  = 0x0F0F0F00; // ( 15,  15,  15)
const Fl_Color FL_RGB_GREY60                 = 0x99999900; // (153, 153, 153)
const Fl_Color FL_RGB_GREY61                 = 0x9C9C9C00; // (156, 156, 156)
const Fl_Color FL_RGB_GREY62                 = 0x9E9E9E00; // (158, 158, 158)
const Fl_Color FL_RGB_GREY63                 = 0xA1A1A100; // (161, 161, 161)
const Fl_Color FL_RGB_GREY64                 = 0xA3A3A300; // (163, 163, 163)
const Fl_Color FL_RGB_GREY65                 = 0xA6A6A600; // (166, 166, 166)
const Fl_Color FL_RGB_GREY66                 = 0xA8A8A800; // (168, 168, 168)
const Fl_Color FL_RGB_GREY67                 = 0xABABAB00; // (171, 171, 171)
const Fl_Color FL_RGB_GREY68                 = 0xADADAD00; // (173, 173, 173)
const Fl_Color FL_RGB_GREY69                 = 0xB0B0B000; // (176, 176, 176)
const Fl_Color FL_RGB_GREY7                  = 0x12121200; // ( 18,  18,  18)
const Fl_Color FL_RGB_GREY70                 = 0xB3B3B300; // (179, 179, 179)
const Fl_Color FL_RGB_GREY71                 = 0xB5B5B500; // (181, 181, 181)
const Fl_Color FL_RGB_GREY72                 = 0xB8B8B800; // (184, 184, 184)
const Fl_Color FL_RGB_GREY73                 = 0xBABABA00; // (186, 186, 186)
const Fl_Color FL_RGB_GREY74                 = 0xBDBDBD00; // (189, 189, 189)
const Fl_Color FL_RGB_GREY75                 = 0xBFBFBF00; // (191, 191, 191)
const Fl_Color FL_RGB_GREY76                 = 0xC2C2C200; // (194, 194, 194)
const Fl_Color FL_RGB_GREY77                 = 0xC4C4C400; // (196, 196, 196)
const Fl_Color FL_RGB_GREY78                 = 0xC7C7C700; // (199, 199, 199)
const Fl_Color FL_RGB_GREY79                 = 0xC9C9C900; // (201, 201, 201)
const Fl_Color FL_RGB_GREY8                  = 0x14141400; // ( 20,  20,  20)
const Fl_Color FL_RGB_GREY80                 = 0xCCCCCC00; // (204, 204, 204)
const Fl_Color FL_RGB_GREY81                 = 0xCFCFCF00; // (207, 207, 207)
const Fl_Color FL_RGB_GREY82                 = 0xD1D1D100; // (209, 209, 209)
const Fl_Color FL_RGB_GREY83                 = 0xD4D4D400; // (212, 212, 212)
const Fl_Color FL_RGB_GREY84                 = 0xD6D6D600; // (214, 214, 214)
const Fl_Color FL_RGB_GREY85                 = 0xD9D9D900; // (217, 217, 217)
const Fl_Color FL_RGB_GREY86                 = 0xDBDBDB00; // (219, 219, 219)
const Fl_Color FL_RGB_GREY87                 = 0xDEDEDE00; // (222, 222, 222)
const Fl_Color FL_RGB_GREY88                 = 0xE0E0E000; // (224, 224, 224)
const Fl_Color FL_RGB_GREY89                 = 0xE3E3E300; // (227, 227, 227)
const Fl_Color FL_RGB_GREY9                  = 0x17171700; // ( 23,  23,  23)
const Fl_Color FL_RGB_GREY90                 = 0xE5E5E500; // (229, 229, 229)
const Fl_Color FL_RGB_GREY91                 = 0xE8E8E800; // (232, 232, 232)
const Fl_Color FL_RGB_GREY92                 = 0xEBEBEB00; // (235, 235, 235)
const Fl_Color FL_RGB_GREY93                 = 0xEDEDED00; // (237, 237, 237)
const Fl_Color FL_RGB_GREY94                 = 0xF0F0F000; // (240, 240, 240)
const Fl_Color FL_RGB_GREY95                 = 0xF2F2F200; // (242, 242, 242)
const Fl_Color FL_RGB_GREY96                 = 0xF5F5F500; // (245, 245, 245)
const Fl_Color FL_RGB_GREY97                 = 0xF7F7F700; // (247, 247, 247)
const Fl_Color FL_RGB_GREY98                 = 0xFAFAFA00; // (250, 250, 250)
const Fl_Color FL_RGB_GREY99                 = 0xFCFCFC00; // (252, 252, 252)
const Fl_Color FL_RGB_HONEYDEW               = 0xF0FFF000; // (240, 255, 240)
const Fl_Color FL_RGB_HONEYDEW1              = 0xF0FFF000; // (240, 255, 240)
const Fl_Color FL_RGB_HONEYDEW2              = 0xE0EEE000; // (224, 238, 224)
const Fl_Color FL_RGB_HONEYDEW3              = 0xC1CDC100; // (193, 205, 193)
const Fl_Color FL_RGB_HONEYDEW4              = 0x838B8300; // (131, 139, 131)
const Fl_Color FL_RGB_HOT_PINK               = 0xFF69B400; // (255, 105, 180)
const Fl_Color FL_RGB_INDIAN_RED             = 0xCD5C5C00; // (205,  92,  92)
const Fl_Color FL_RGB_IVORY                  = 0xFFFFF000; // (255, 255, 240)
const Fl_Color FL_RGB_IVORY1                 = 0xFFFFF000; // (255, 255, 240)
const Fl_Color FL_RGB_IVORY2                 = 0xEEEEE000; // (238, 238, 224)
const Fl_Color FL_RGB_IVORY3                 = 0xCDCDC100; // (205, 205, 193)
const Fl_Color FL_RGB_IVORY4                 = 0x8B8B8300; // (139, 139, 131)
const Fl_Color FL_RGB_KHAKI                  = 0xF0E68C00; // (240, 230, 140)
const Fl_Color FL_RGB_KHAKI1                 = 0xFFF68F00; // (255, 246, 143)
const Fl_Color FL_RGB_KHAKI2                 = 0xEEE68500; // (238, 230, 133)
const Fl_Color FL_RGB_KHAKI3                 = 0xCDC67300; // (205, 198, 115)
const Fl_Color FL_RGB_KHAKI4                 = 0x8B864E00; // (139, 134,  78)
const Fl_Color FL_RGB_LAVENDER               = 0xE6E6FA00; // (230, 230, 250)
const Fl_Color FL_RGB_LAVENDER_BLUSH         = 0xFFF0F500; // (255, 240, 245)
const Fl_Color FL_RGB_LAWN_GREEN             = 0x7CFC0000; // (124, 252,   0)
const Fl_Color FL_RGB_LEMON_CHIFFON          = 0xFFFACD00; // (255, 250, 205)
const Fl_Color FL_RGB_LIGHT_BLUE             = 0xADD8E600; // (173, 216, 230)
const Fl_Color FL_RGB_LIGHT_CORAL            = 0xF0808000; // (240, 128, 128)
const Fl_Color FL_RGB_LIGHT_CYAN             = 0xE0FFFF00; // (224, 255, 255)
const Fl_Color FL_RGB_LIGHT_GOLDENROD        = 0xEEDD8200; // (238, 221, 130)
const Fl_Color FL_RGB_LIGHT_GOLDENROD_YELLOW = 0xFAFAD200; // (250, 250, 210)
const Fl_Color FL_RGB_LIGHT_GRAY             = 0xD3D3D300; // (211, 211, 211)
const Fl_Color FL_RGB_LIGHT_GREEN            = 0x90EE9000; // (144, 238, 144)
const Fl_Color FL_RGB_LIGHT_GREY             = 0xD3D3D300; // (211, 211, 211)
const Fl_Color FL_RGB_LIGHT_PINK             = 0xFFB6C100; // (255, 182, 193)
const Fl_Color FL_RGB_LIGHT_SALMON           = 0xFFA07A00; // (255, 160, 122)
```

```
const Fl_Color FL_RGB_LIGHT_SEA_GREEN        = 0x20B2AA00; // ( 32, 178, 170)
const Fl_Color FL_RGB_LIGHT_SKY_BLUE         = 0x87CEFA00; // (135, 206, 250)
const Fl_Color FL_RGB_LIGHT_SLATE_BLUE       = 0x8470FF00; // (132, 112, 255)
const Fl_Color FL_RGB_LIGHT_SLATE_GRAY       = 0x77889900; // (119, 136, 153)
const Fl_Color FL_RGB_LIGHT_SLATE_GREY       = 0x77889900; // (119, 136, 153)
const Fl_Color FL_RGB_LIGHT_STEEL_BLUE       = 0xB0C4DE00; // (176, 196, 222)
const Fl_Color FL_RGB_LIGHT_YELLOW           = 0xFFFFE000; // (255, 255, 224)
const Fl_Color FL_RGB_LIME_GREEN             = 0x32CD3200; // ( 50, 205,  50)
const Fl_Color FL_RGB_LINEN                  = 0xFAF0E600; // (250, 240, 230)
const Fl_Color FL_RGB_MAGENTA                = 0xFF00FF00; // (255,   0, 255)
const Fl_Color FL_RGB_MAGENTA1               = 0xFF00FF00; // (255,   0, 255)
const Fl_Color FL_RGB_MAGENTA2               = 0xEE00EE00; // (238,   0, 238)
const Fl_Color FL_RGB_MAGENTA3               = 0xCD00CD00; // (205,   0, 205)
const Fl_Color FL_RGB_MAGENTA4               = 0x8B008B00; // (139,   0, 139)
const Fl_Color FL_RGB_MAROON                 = 0xB0306000; // (176,  48,  96)
const Fl_Color FL_RGB_MAROON1                = 0xFF34B300; // (255,  52, 179)
const Fl_Color FL_RGB_MAROON2                = 0xEE30A700; // (238,  48, 167)
const Fl_Color FL_RGB_MAROON3                = 0xCD299000; // (205,  41, 144)
const Fl_Color FL_RGB_MAROON4                = 0x8B1C6200; // (139,  28,  98)
const Fl_Color FL_RGB_MEDIUM_AQUAMARINE      = 0x66CDAA00; // (102, 205, 170)
const Fl_Color FL_RGB_MEDIUM_BLUE            = 0x0000CD00; // (  0,   0, 205)
const Fl_Color FL_RGB_MEDIUM_ORCHID          = 0xBA55D300; // (186,  85, 211)
const Fl_Color FL_RGB_MEDIUM_PURPLE          = 0x9370DB00; // (147, 112, 219)
const Fl_Color FL_RGB_MEDIUM_SEA_GREEN       = 0x3CB37100; // ( 60, 179, 113)
const Fl_Color FL_RGB_MEDIUM_SLATE_BLUE      = 0x7B68EE00; // (123, 104, 238)
const Fl_Color FL_RGB_MEDIUM_SPRING_GREEN    = 0x00FA9A00; // (  0, 250, 154)
const Fl_Color FL_RGB_MEDIUM_TURQUOISE       = 0x48D1CC00; // ( 72, 209, 204)
const Fl_Color FL_RGB_MEDIUM_VIOLET_RED      = 0xC7158500; // (199,  21, 133)
const Fl_Color FL_RGB_MIDNIGHT_BLUE          = 0x19197000; // ( 25,  25, 112)
const Fl_Color FL_RGB_MINT_CREAM             = 0xF5FFFA00; // (245, 255, 250)
const Fl_Color FL_RGB_MISTY_ROSE             = 0xFFE4E100; // (255, 228, 225)
const Fl_Color FL_RGB_MOCCASIN               = 0xFFE4B500; // (255, 228, 181)
const Fl_Color FL_RGB_NAVAJO_WHITE           = 0xFFDEAD00; // (255, 222, 173)
const Fl_Color FL_RGB_NAVY                   = 0x00008000; // (  0,   0, 128)
const Fl_Color FL_RGB_NAVY_BLUE              = 0x00008000; // (  0,   0, 128)
const Fl_Color FL_RGB_OLD_LACE               = 0xFDF5E600; // (253, 245, 230)
const Fl_Color FL_RGB_OLIVE_DRAB             = 0x6B8E2300; // (107, 142,  35)
const Fl_Color FL_RGB_ORANGE                 = 0xFFA50000; // (255, 165,   0)
const Fl_Color FL_RGB_ORANGE1                = 0xFFA50000; // (255, 165,   0)
const Fl_Color FL_RGB_ORANGE2                = 0xEE9A0000; // (238, 154,   0)
const Fl_Color FL_RGB_ORANGE3                = 0xCD850000; // (205, 133,   0)
const Fl_Color FL_RGB_ORANGE4                = 0x8B5A0000; // (139,  90,   0)
const Fl_Color FL_RGB_ORANGE_RED             = 0xFF450000; // (255,  69,   0)
const Fl_Color FL_RGB_ORCHID                 = 0xDA70D600; // (218, 112, 214)
const Fl_Color FL_RGB_ORCHID1                = 0xFF83FA00; // (255, 131, 250)
const Fl_Color FL_RGB_ORCHID2                = 0xEE7AE900; // (238, 122, 233)
const Fl_Color FL_RGB_ORCHID3                = 0xCD69C900; // (205, 105, 201)
const Fl_Color FL_RGB_ORCHID4                = 0x8B478900; // (139,  71, 137)
const Fl_Color FL_RGB_PALE_GOLDENROD         = 0xEEE8AA00; // (238, 232, 170)
const Fl_Color FL_RGB_PALE_GREEN             = 0x98FB9800; // (152, 251, 152)
const Fl_Color FL_RGB_PALE_TURQUOISE         = 0xAFEEEE00; // (175, 238, 238)
const Fl_Color FL_RGB_PALE_VIOLET_RED        = 0xDB709300; // (219, 112, 147)
const Fl_Color FL_RGB_PAPAYA_WHIP            = 0xFFEFD500; // (255, 239, 213)
const Fl_Color FL_RGB_PEACH_PUFF             = 0xFFDAB900; // (255, 218, 185)
const Fl_Color FL_RGB_PERU                   = 0xCD853F00; // (205, 133,  63)
const Fl_Color FL_RGB_PINK                   = 0xFFC0CB00; // (255, 192, 203)
const Fl_Color FL_RGB_PINK1                  = 0xFFB5C500; // (255, 181, 197)
const Fl_Color FL_RGB_PINK2                  = 0xEEA9B800; // (238, 169, 184)
const Fl_Color FL_RGB_PINK3                  = 0xCD919E00; // (205, 145, 158)
const Fl_Color FL_RGB_PINK4                  = 0x8B636C00; // (139,  99, 108)
const Fl_Color FL_RGB_PLUM                   = 0xDDA0DD00; // (221, 160, 221)
const Fl_Color FL_RGB_PLUM1                  = 0xFFBBFF00; // (255, 187, 255)
const Fl_Color FL_RGB_PLUM2                  = 0xEEAEEE00; // (238, 174, 238)
const Fl_Color FL_RGB_PLUM3                  = 0xCD96CD00; // (205, 150, 205)
const Fl_Color FL_RGB_PLUM4                  = 0x8B668B00; // (139, 102, 139)
const Fl_Color FL_RGB_POWDER_BLUE            = 0xB0E0E600; // (176, 224, 230)
const Fl_Color FL_RGB_PURPLE                 = 0xA020F000; // (160,  32, 240)
const Fl_Color FL_RGB_PURPLE1                = 0x9B30FF00; // (155,  48, 255)
const Fl_Color FL_RGB_PURPLE2                = 0x912CEE00; // (145,  44, 238)
const Fl_Color FL_RGB_PURPLE3                = 0x7D26CD00; // (125,  38, 205)
const Fl_Color FL_RGB_PURPLE4                = 0x551A8B00; // ( 85,  26, 139)
const Fl_Color FL_RGB_RED                    = 0xFF000000; // (255,   0,   0)
const Fl_Color FL_RGB_RED1                   = 0xFF000000; // (255,   0,   0)
const Fl_Color FL_RGB_RED2                   = 0xEE000000; // (238,   0,   0)
const Fl_Color FL_RGB_RED3                   = 0xCD000000; // (205,   0,   0)
const Fl_Color FL_RGB_RED4                   = 0x8B000000; // (139,   0,   0)
const Fl_Color FL_RGB_ROSY_BROWN             = 0xBC8F8F00; // (188, 143, 143)
const Fl_Color FL_RGB_ROYAL_BLUE             = 0x4169E100; // ( 65, 105, 225)
const Fl_Color FL_RGB_SADDLE_BROWN           = 0x8B451300; // (139,  69,  19)
const Fl_Color FL_RGB_SALMON                 = 0xFA807200; // (250, 128, 114)
const Fl_Color FL_RGB_SALMON1                = 0xFF8C6900; // (255, 140, 105)
const Fl_Color FL_RGB_SALMON2                = 0xEE826200; // (238, 130,  98)
const Fl_Color FL_RGB_SALMON3                = 0xCD705400; // (205, 112,  84)
const Fl_Color FL_RGB_SALMON4                = 0x8B4C3900; // (139,  76,  57)
const Fl_Color FL_RGB_SANDY_BROWN            = 0xF4A46000; // (244, 164,  96)
const Fl_Color FL_RGB_SEASHELL               = 0xFFF5EE00; // (255, 245, 238)
const Fl_Color FL_RGB_SEASHELL1              = 0xFFF5EE00; // (255, 245, 238)
```

```
const Fl_Color FL_RGB_SEASHELL2              = 0xEEE5DE00; // (238, 229, 222)
const Fl_Color FL_RGB_SEASHELL3              = 0xCDC5BF00; // (205, 197, 191)
const Fl_Color FL_RGB_SEASHELL4              = 0x8B868200; // (139, 134, 130)
const Fl_Color FL_RGB_SEA_GREEN              = 0x2E8B5700; // ( 46, 139,  87)
const Fl_Color FL_RGB_SIENNA                 = 0xA0522D00; // (160,  82,  45)
const Fl_Color FL_RGB_SIENNA1                = 0xFF824700; // (255, 130,  71)
const Fl_Color FL_RGB_SIENNA2                = 0xEE794200; // (238, 121,  66)
const Fl_Color FL_RGB_SIENNA3                = 0xCD683900; // (205, 104,  57)
const Fl_Color FL_RGB_SIENNA4                = 0x8B472600; // (139,  71,  38)
const Fl_Color FL_RGB_SKY_BLUE               = 0x87CEEB00; // (135, 206, 235)
const Fl_Color FL_RGB_SLATE_BLUE             = 0x6A5ACD00; // (106,  90, 205)
const Fl_Color FL_RGB_SLATE_GRAY             = 0x70809000; // (112, 128, 144)
const Fl_Color FL_RGB_SLATE_GREY             = 0x70809000; // (112, 128, 144)
const Fl_Color FL_RGB_SNOW                   = 0xFFFAFA00; // (255, 250, 250)
const Fl_Color FL_RGB_SNOW1                  = 0xFFFAFA00; // (255, 250, 250)
const Fl_Color FL_RGB_SNOW2                  = 0xEEE9E900; // (238, 233, 233)
const Fl_Color FL_RGB_SNOW3                  = 0xCDC9C900; // (205, 201, 201)
const Fl_Color FL_RGB_SNOW4                  = 0x8B898900; // (139, 137, 137)
const Fl_Color FL_RGB_SPRING_GREEN           = 0x00FF7F00; // (  0, 255, 127)
const Fl_Color FL_RGB_STEEL_BLUE             = 0x4682B400; // ( 70, 130, 180)
const Fl_Color FL_RGB_TAN                    = 0xD2B48C00; // (210, 180, 140)
const Fl_Color FL_RGB_TAN1                   = 0xFFA54F00; // (255, 165,  79)
const Fl_Color FL_RGB_TAN2                   = 0xEE9A4900; // (238, 154,  73)
const Fl_Color FL_RGB_TAN3                   = 0xCD853F00; // (205, 133,  63)
const Fl_Color FL_RGB_TAN4                   = 0x8B5A2B00; // (139,  90,  43)
const Fl_Color FL_RGB_THISTLE                = 0xD8BFD800; // (216, 191, 216)
const Fl_Color FL_RGB_THISTLE1               = 0xFFE1FF00; // (255, 225, 255)
const Fl_Color FL_RGB_THISTLE2               = 0xEED2EE00; // (238, 210, 238)
const Fl_Color FL_RGB_THISTLE3               = 0xCDB5CD00; // (205, 181, 205)
const Fl_Color FL_RGB_THISTLE4               = 0x8B7B8B00; // (139, 123, 139)
const Fl_Color FL_RGB_TOMATO                 = 0xFF634700; // (255,  99,  71)
const Fl_Color FL_RGB_TOMATO1                = 0xFF634700; // (255,  99,  71)
const Fl_Color FL_RGB_TOMATO2                = 0xEE5C4200; // (238,  92,  66)
const Fl_Color FL_RGB_TOMATO3                = 0xCD4F3900; // (205,  79,  57)
const Fl_Color FL_RGB_TOMATO4                = 0x8B362600; // (139,  54,  38)
const Fl_Color FL_RGB_TURQUOISE              = 0x40E0D000; // ( 64, 224, 208)
const Fl_Color FL_RGB_TURQUOISE1             = 0x00F5FF00; // (  0, 245, 255)
const Fl_Color FL_RGB_TURQUOISE2             = 0x00E5EE00; // (  0, 229, 238)
const Fl_Color FL_RGB_TURQUOISE3             = 0x00C5CD00; // (  0, 197, 205)
const Fl_Color FL_RGB_TURQUOISE4             = 0x00868B00; // (  0, 134, 139)
const Fl_Color FL_RGB_VIOLET                 = 0xEE82EE00; // (238, 130, 238)
const Fl_Color FL_RGB_VIOLET_RED             = 0xD0209000; // (208,  32, 144)
const Fl_Color FL_RGB_WHEAT                  = 0xF5DEB300; // (245, 222, 179)
const Fl_Color FL_RGB_WHEAT1                 = 0xFFE7BA00; // (255, 231, 186)
const Fl_Color FL_RGB_WHEAT2                 = 0xEED8AE00; // (238, 216, 174)
const Fl_Color FL_RGB_WHEAT3                 = 0xCDBA9600; // (205, 186, 150)
const Fl_Color FL_RGB_WHEAT4                 = 0x8B7E6600; // (139, 126, 102)
const Fl_Color FL_RGB_WHITE                  = 0xFFFFFF00; // (255, 255, 255)
const Fl_Color FL_RGB_WHITE_SMOKE            = 0xF5F5F500; // (245, 245, 245)
const Fl_Color FL_RGB_YELLOW                 = 0xFFFF0000; // (255, 255,   0)
const Fl_Color FL_RGB_YELLOW1                = 0xFFFF0000; // (255, 255,   0)
const Fl_Color FL_RGB_YELLOW2                = 0xEEEE0000; // (238, 238,   0)
const Fl_Color FL_RGB_YELLOW3                = 0xCDCD0000; // (205, 205,   0)
const Fl_Color FL_RGB_YELLOW4                = 0x8B8B0000; // (139, 139,   0)
const Fl_Color FL_RGB_YELLOW_GREEN           = 0x9ACD3200; // (154, 205,  50)
```

14.211 wayland.H File Reference

Definitions of functions specific to the Wayland platform.

Typedefs

- typedef struct _cairo **cairo_t**
- typedef void ∗ **EGLContext**

Functions

- int **fl_wl_buffer_scale** (Fl_Window ∗window)

  Returns the current buffer scaling factor for `window`*.*
- struct wl_compositor ∗ fl_wl_compositor ()

  Returns the wl_compositor of the current Wayland session.
- struct wl_display ∗ **fl_wl_display** ()

  Returns the Wayland display in use.
- Fl_Window ∗ **fl_wl_find** (struct wld_window ∗)

  Returns the Fl_Window corresponding to a given the platform-specific window reference.

- cairo_t * fl_wl_gc ()

  Returns the cairo context associated to the current window or Fl_Image_Surface.
- EGLContext **fl_wl_glcontext** (GLContext rc)

  Returns the EGLContext corresponding to the given GLContext.
- struct wl_surface * **fl_wl_surface** (struct wld_window *xid)

  Returns the wl_surface associated to a shown window.
- struct wld_window * **fl_wl_xid** (const Fl_Window *win)

  Returns a platform-specific reference associated to a shown window.

14.211.1 Detailed Description

Definitions of functions specific to the Wayland platform.

14.211.2 Function Documentation

14.211.2.1 fl_wl_compositor()

```
struct wl_compositor * fl_wl_compositor ( )  [extern]
```

Returns the wl_compositor of the current Wayland session.
This allows, for example, to create a wl_surface with

```
struct wl_surface *my_wl_surface = wl_compositor_create_surface(fl_wl_compositor());
```

14.211.2.2 fl_wl_gc()

```
cairo_t * fl_wl_gc ( )  [extern]
```

Returns the cairo context associated to the current window or Fl_Image_Surface.
Usable both when the Wayland/X11 hybrid FLTK library runs as a Wayland client and when it runs as an X11 client.

14.212 wayland.H

Go to the documentation of this file.

```
00001 //
00002 // Wayland/X11 hybrid platform header file for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 1998-2022 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file.  If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 //     https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 //     https://www.fltk.org/bugs.php
00015 //
00016
00017 #if !defined(FL_PLATFORM_H)
00018 #  error "Never use <FL/wayland.H> directly; include <FL/platform.H> instead."
00019 #endif // !FL_PLATFORM_H
00020
00025 // ***********  for Wayland component ***********
00026
00027 typedef struct _cairo cairo_t;
00028
00030 extern FL_EXPORT struct wl_display *fl_wl_display();
00032 extern FL_EXPORT struct wl_surface *fl_wl_surface(struct wld_window *xid);
00034 extern FL_EXPORT struct wld_window *fl_wl_xid(const Fl_Window *win);
00036 extern FL_EXPORT Fl_Window *fl_wl_find(struct wld_window *);
00039 extern FL_EXPORT cairo_t *fl_wl_gc();
00046 extern FL_EXPORT struct wl_compositor *fl_wl_compositor();
00048 extern FL_EXPORT int fl_wl_buffer_scale(Fl_Window *window);
00049 typedef void *EGLContext;
00051 extern FL_EXPORT EGLContext fl_wl_glcontext(GLContext rc);
00052
00053 #ifndef FL_DOXYGEN
00054
00055 #  ifdef FLTK_USE_X11
00056 //  *********** for X11 component ***********
00057 #    include "x11.H"
00058 #  else
```

```
      typedef struct wld_window *Window;
#  endif

#endif // FL_DOXYGEN
```

14.213 win32.H File Reference

Definitions of functions specific to the Windows platform.

Functions

- HICON **fl_image_to_HICON** (const Fl_RGB_Image *image, bool is_icon, int hotx, int hoty)

  Convert an RGB image into a HICON resource.
- HINSTANCE **fl_win32_display** ()

  Returns the Windows-specific display in use
- Fl_Window * **fl_win32_find** (HWND)

  Returns the Fl_Window corresponding to the given Windows-specific window reference.
- HDC **fl_win32_gc** ()

  Returns the Windows-specific graphics context for the current window
- HGLRC **fl_win32_glcontext** (GLContext rc)

  Returns the Windows-specific GL rendering context corresponding to the given GLContext
- HWND **fl_win32_xid** (const Fl_Window *win)

  Returns the Windows-specific window reference corresponding to the given Fl_Window object.

14.213.1 Detailed Description

Definitions of functions specific to the Windows platform.

14.214 win32.H

Go to the documentation of this file.

```
//
// Windows platform header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2022 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

// Do not directly include this file, instead use <FL/platform.H>.  It will
// include this file if _WIN32 is defined.  This is to encourage
// portability of even the system-specific code...

#ifdef FL_DOXYGEN

extern HWND fl_win32_xid(const Fl_Window *win);
extern Fl_Window *fl_win32_find(HWND);
extern HGLRC fl_win32_glcontext(GLContext rc);
extern HDC fl_win32_gc();
extern HINSTANCE fl_win32_display();
extern HICON fl_image_to_HICON(const Fl_RGB_Image *image, bool is_icon, int hotx, int hoty);

#else

#ifndef FL_PLATFORM_H
#  error "Never use <FL/win32.H> directly; include <FL/platform.H> instead."
#endif // !FL_PLATFORM_H
```

```

#include <windows.h>
typedef HWND Window;

typedef struct HGLRC__ *HGLRC;
extern FL_EXPORT HGLRC fl_win32_glcontext(GLContext rc);
extern FL_EXPORT HWND fl_win32_xid(const Fl_Window *win);
extern FL_EXPORT Fl_Window *fl_win32_find(HWND);
extern FL_EXPORT HICON fl_image_to_HICON(const class Fl_RGB_Image *image, bool is_icon, int hotx, int hoty);

// this part is included only when compiling the FLTK library or if requested explicitly
#if defined(FL_LIBRARY) || defined(FL_INTERNALS)

// In some of the distributions, the gcc header files are missing some stuff:
#ifndef LPMINMAXINFO
#define LPMINMAXINFO MINMAXINFO*
#endif
#ifndef VK_LWIN
#define VK_LWIN 0x5B
#define VK_RWIN 0x5C
#define VK_APPS 0x5D
#endif

extern FL_EXPORT UINT fl_wake_msg;
extern FL_EXPORT char fl_override_redirect; // hack into Fl_Window::make_xid()
extern FL_EXPORT HPALETTE fl_palette; // non-zero only on 8-bit displays!
extern void fl_release_dc(HWND w, HDC dc);
extern FL_EXPORT void fl_save_dc( HWND w, HDC dc);

#endif // FL_LIBRARY || FL_INTERNALS

// most recent fl_color() or fl_rgbcolor() points at one of these:
extern FL_EXPORT struct Fl_XMap {
  COLORREF rgb; // this should be the type the RGB() macro returns
  HPEN pen;     // pen, 0 if none created yet
  int brush;    // ref to solid brush, 0 if none created yet
  int pwidth;   // the width of the pen, if present
} *fl_current_xmap;
inline COLORREF fl_RGB() {return fl_current_xmap->rgb;}
inline HPEN fl_pen() {return fl_current_xmap->pen;}
FL_EXPORT HBRUSH fl_brush(); // allocates a brush if necessary
FL_EXPORT HBRUSH fl_brush_action(int); // now does the real work

extern FL_EXPORT HINSTANCE fl_display;
extern FL_EXPORT HINSTANCE fl_win32_display();
extern FL_EXPORT HDC fl_gc;
extern FL_EXPORT HDC fl_win32_gc();
extern FL_EXPORT MSG fl_msg;
extern FL_EXPORT HDC fl_GetDC(Window);
extern FL_EXPORT HDC fl_makeDC(HBITMAP);

#endif // FL_DOXYGEN
```

14.215 win32_target.h

```
/*
 *  Define min. Windows version macros for the Fast Light Tool Kit (FLTK).
 *
 * Copyright 1998-2026 by Bill Spitzak and others.
 *
 * This library is free software. Distribution and use rights are outlined in
 * the file "COPYING" which should have been included with this file.  If this
 * file is missing or damaged, see the license at:
 *
 *     https://www.fltk.org/COPYING.php
 *
 * Please see the following page on how to report bugs and issues:
 *
 *     https://www.fltk.org/bugs.php
 */

#ifndef FL_WIN32_TARGET_VERSION
#  error "Please define FL_WIN32_TARGET_VERSION before including FL/win32_target.h"
#endif

/* _WIN32_WINNT */
#ifndef _WIN32_WINNT
#  define _WIN32_WINNT FL_WIN32_TARGET_VERSION
#elif _WIN32_WINNT < FL_WIN32_TARGET_VERSION
#  undef _WIN32_WINNT
#  define _WIN32_WINNT FL_WIN32_TARGET_VERSION
#endif

/* WINVER */
```

```
#ifndef WINVER
#  define WINVER FL_WIN32_TARGET_VERSION
#elif WINVER < FL_WIN32_TARGET_VERSION
#  undef WINVER
#  define WINVER FL_WIN32_TARGET_VERSION
#endif

// Note: we can't undefine the macro because this would break the build.
// #undef FL_WIN32_TARGET_VERSION
```

14.216 x.H

```
//
// *Deprecated* platform header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2018 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

// IMPORTANT: This file is deprecated since FLTK 1.4.0. DO NOT include it.
// FL/x.H will be removed in a future FLTK release.

// Please #include <FL/platform.H> instead if you really need it. See
// documentation in FL/platform.H to decide whether you need that file.

#if !defined(Fl_X_H) && !defined(FL_DOXYGEN)
#  define Fl_X_H
#  include <FL/platform.H>
#endif // !Fl_X_H
```

14.217 x11.H File Reference

Definitions of functions specific to the X11 platform.

Functions

- cairo_t * **fl_cairo_gc** ()

  Returns the Cairo-specific currently active graphics context (FLTK_GRAPHICS_CAIRO=On)
- Display * **fl_x11_display** ()

  Returns the X11 Display in use.
- Fl_Window * fl_x11_find (Window xid)

  Returns the Fl_Window corresponding to the given Window reference.
- GC fl_x11_gc ()

  Returns the X11-specific currently active graphics context.
- void fl_x11_use_display (Display *d)

  Have FLTK use a pre-established X11 connection.
- Window fl_x11_xid (const Fl_Window *win)

  Returns the Window reference for the given Fl_Window, or zero if not `shown()`.

14.217.1 Detailed Description

Definitions of functions specific to the X11 platform.

14.217.2 Function Documentation

14.217.2.1 fl_x11_find()

```
Fl_Window * fl_x11_find (
            Window xid )  [extern]
```

Returns the Fl_Window corresponding to the given Window reference.

14.217.2.2 fl_x11_gc()

```
GC fl_x11_gc ( ) [extern]
```

Returns the X11-specific currently active graphics context.

14.217.2.3 fl_x11_use_display()

```
void fl_x11_use_display (
            Display * d ) [extern]
```

Have FLTK use a pre-established X11 connection.

This function must be called before FLTK attempts to open its own X11 connection, that is, as long as fl_x11_display() returns NULL.

Parameters

d	the X11 Display∗ value representing a valid, pre-established X11 connection

14.217.2.4 fl_x11_xid()

```
Window fl_x11_xid (
            const Fl_Window * win ) [extern]
```

Returns the Window reference for the given Fl_Window, or zero if not `shown()`.

14.218 x11.H

Go to the documentation of this file.

```
00001 //
00002 // X11 platform header file for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 1998-2022 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file.  If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 //     https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 //     https://www.fltk.org/bugs.php
00015 //
00016
00017 // Do not directly include this file, instead use <FL/platform.H>.  It will
00018 // include this file if FLTK_USE_X11 is defined.  This is to encourage
00019 // portability of even the system-specific code...
00020
00021 #ifdef FL_DOXYGEN
00022
00028 extern Display *fl_x11_display();
00034 extern void fl_x11_use_display(Display *d);
00036 extern Window fl_x11_xid(const Fl_Window *win);
00038 extern Fl_Window *fl_x11_find(Window xid);
00040 extern GC fl_x11_gc();
00042 extern cairo_t* fl_cairo_gc();
00043 #else // ! FL_DOXYGEN
00044
00045 #ifndef FL_PLATFORM_H
00046 #  error "Never use <FL/x11.H> directly; include <FL/platform.H> instead."
00047 #endif // !FL_PLATFORM_H
00048
```

```
#include <FL/Enumerations.H>

#if defined(_ABIN32) || defined(_ABI64) // fix for broken SGI Irix X .h files
#  pragma set woff 3322
#endif
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#if defined(_ABIN32) || defined(_ABI64)
#  pragma reset woff 3322
#endif
#include <X11/Xatom.h>

typedef struct __GLXcontextRec *GLXContext;
extern GLXContext fl_x11_glcontext(GLContext rc);

// constant info about the X server connection:
extern FL_EXPORT Display *fl_display;
extern FL_EXPORT Display *fl_x11_display();
extern FL_EXPORT void fl_x11_use_display(Display *);
extern FL_EXPORT Window fl_x11_xid(const Fl_Window *win);
extern FL_EXPORT Fl_Window *fl_x11_find(Window);
extern FL_EXPORT int fl_screen;
extern FL_EXPORT XVisualInfo *fl_visual;
extern FL_EXPORT Colormap fl_colormap;

// drawing functions:
extern FL_EXPORT GC fl_gc;
#if FLTK_USE_CAIRO
  typedef struct _cairo cairo_t;
  extern FL_EXPORT cairo_t* fl_cairo_gc();
#endif
extern FL_EXPORT GC fl_x11_gc();
FL_EXPORT ulong fl_xpixel(Fl_Color i);
FL_EXPORT ulong fl_xpixel(uchar r, uchar g, uchar b);

// feed events into fltk:
FL_EXPORT int fl_handle(const XEvent&);

// you can use these in Fl::add_handler() to look at events:
extern FL_EXPORT const XEvent* fl_xevent;
extern FL_EXPORT ulong fl_event_time;

#if defined(FL_LIBRARY) || defined(FL_INTERNALS)
extern FL_EXPORT Window fl_message_window;
extern FL_EXPORT void *fl_xftfont;

// access to core fonts:
// This class provides a "smart pointer" that returns a pointer to an XFontStruct.
// The global variable fl_xfont can be called wherever a bitmap "core" font is
// needed, e.g. when rendering to a GL context under X11.
// With Xlib / X11 fonts, fl_xfont will return the current selected font.
// With XFT / X11 fonts, fl_xfont will attempt to return the bitmap "core" font most
// similar to (usually the same as) the current XFT font.
class FL_EXPORT Fl_XFont_On_Demand
{
public:
  Fl_XFont_On_Demand(XFontStruct* p = NULL) : ptr(p) { }
  Fl_XFont_On_Demand& operator=(const Fl_XFont_On_Demand& x)
  { ptr = x.ptr;  return *this; }
  Fl_XFont_On_Demand& operator=(XFontStruct* p)
  { ptr = p;  return *this; }
  XFontStruct* value();
  operator XFontStruct*() { return value(); }
  XFontStruct& operator*() { return *value(); }
  XFontStruct* operator->() { return value(); }
  bool operator==(const Fl_XFont_On_Demand& x) { return ptr == x.ptr; }
  bool operator!=(const Fl_XFont_On_Demand& x) { return ptr != x.ptr; }
private:
  XFontStruct *ptr;
};
extern FL_EXPORT Fl_XFont_On_Demand fl_xfont;

extern FL_EXPORT char fl_override_redirect; // hack into Fl_X::make_xid()

#endif // FL_LIBRARY || FL_INTERNALS

#endif // FL_DOXYGEN
```

14.219 cgdebug.h

```
//
// macOS Core Graphics debugging help for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2025 by Bill Spitzak and others.
//
```

```
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

// This file allows easier debugging of macOS Core Graphics
// code. This file is normally not included into any FLTK builds,
// but since it has proven to be tremendously useful in debugging
// the FLTK port to "Quartz", I decided to add this file in case
// more bugs show up.
//
// This header is activated by adding the following line to "config.h"
//
//   #include "src/cgdebug.h"
//
// Running "cmake" will remove this line from "config.h".
//
// When used erroneously, Core Graphics prints warnings to
// stderr. This is helpful, however it is not possible to
// associate a line number or source file with the warning message.
// This header file outputs a trace of CG calls, interweaving
// them with CG warnings.
//
// Matthias

#ifndef CGDEBUG
#define CGDEBUG

#include <stdio.h>
#include <Carbon/Carbon.h>

//+BitmapContextCreate
//+BitmapContextGetData
// ClipCGContextToRegion
// QDBeginCGContext
// QDEndCGContext

//+AddArc
//+AddLineToPoint
// ClipToRect
// ClosePath
//+ConcatCTM
//+DrawImage
// FillPath
// FillRect
// Flush
//+GetCTM
// MoveToPoint
//+Release
// RestoreGState
// SaveGState
//+ScaleCTM
//+SetLineCap
//+SetLineDash
//+SetLineJoin
//+SetLineWidth
//+SetRGBFillColor
//+SetRGBStrokeColor
//+SetShouldAntialias
//+SetTextMatrix
//+StrokePath
//+TranslateCTM

inline OSStatus dbgLocation(const char *file, int line)
{
  fprintf(stderr, "%s:%d ", file, line);
  return 0;
}

inline OSStatus dbgEndl()
{
  fprintf(stderr, "\n");
  return 0;
}


inline void dbgCGContextClipToRect(CGContextRef a, CGRect b)
{
  CGContextClipToRect(a, b);
}

```

```
#define CGContextClipToRect(a, b) { \
  fprintf(stderr, "%s:%d ", __FILE__, __LINE__); \
  dbgCGContextClipToRect(a, b); \
  fprintf(stderr, "\n"); }

inline void dbgCGContextFillRect(CGContextRef a, CGRect b)
{
  CGContextFillRect(a, b);
}

#define CGContextFillRect(a, b) { \
  fprintf(stderr, "%s:%d ", __FILE__, __LINE__); \
  dbgCGContextFillRect(a, b); \
  fprintf(stderr, "\n"); }

inline OSStatus dbgQDEndCGContext(CGrafPtr a, CGContextRef *b)
{
  return QDEndCGContext(a, b);
}

#define QDEndCGContext(a, b) ( \
  dbgLocation(__FILE__, __LINE__) + \
  dbgQDEndCGContext(a, b) + \
  dbgEndl() )

inline OSStatus dbgQDBeginCGContext(CGrafPtr a, CGContextRef *b)
{
  return QDBeginCGContext(a, b);
}

#define QDBeginCGContext(a, b) ( \
  dbgLocation(__FILE__, __LINE__) + \
  dbgQDBeginCGContext(a, b) + \
  dbgEndl() )

inline void dbgClipCGContextToRegion(CGContextRef a, const Rect *b, RgnHandle c)
{
  ClipCGContextToRegion(a, b, c);
}

#define ClipCGContextToRegion(a, b, c) { \
  fprintf(stderr, "%s:%d ", __FILE__, __LINE__); \
  dbgClipCGContextToRegion(a, b, c); \
  fprintf(stderr, "\n"); }

inline void dbgCGContextMoveToPoint(CGContextRef context, float x, float y)
{
  CGContextMoveToPoint(context, x, y);
}

#define CGContextMoveToPoint(a, b, c) { \
  fprintf(stderr, "%s:%d ", __FILE__, __LINE__); \
  dbgCGContextMoveToPoint(a, b, c); \
  fprintf(stderr, "\n"); }

inline void dbgCGContextFillPath(CGContextRef context)
{
  CGContextFillPath(context);
}

#define CGContextFillPath(a) { \
  fprintf(stderr, "%s:%d ", __FILE__, __LINE__); \
  dbgCGContextFillPath(a); \
  fprintf(stderr, "\n"); }

inline void dbgCGContextClosePath(CGContextRef context)
{
  CGContextClosePath(context);
}

#define CGContextClosePath(a) { \
  fprintf(stderr, "%s:%d ", __FILE__, __LINE__); \
  dbgCGContextClosePath(a); \
  fprintf(stderr, "\n"); }

inline void dbgCGContextFlush(CGContextRef context)
{
  CGContextFlush(context);
}

#define CGContextFlush(a) { \
  fprintf(stderr, "%s:%d ", __FILE__, __LINE__); \
  dbgCGContextFlush(a); \
  fprintf(stderr, "\n"); }

inline void dbgCGContextSaveGState(CGContextRef context)
{
```

```
  CGContextSaveGState(context);
}

#define CGContextSaveGState(a) { \
  fprintf(stderr, "%s:%d ", __FILE__, __LINE__); \
  dbgCGContextSaveGState(a); \
  fprintf(stderr, "\n"); }

inline void dbgCGContextRestoreGState(CGContextRef context)
{
  CGContextRestoreGState(context);
}

#define CGContextRestoreGState(a) { \
  fprintf(stderr, "%s:%d ", __FILE__, __LINE__); \
  dbgCGContextRestoreGState(a); \
  fprintf(stderr, "\n"); }


#endif // CGDEBUG
```

14.220 fastarrow.h

```
#define fastarrow_width 16
#define fastarrow_height 16
static const unsigned char fastarrow_bits[] = {
   0x00, 0x00, 0x00, 0x07, 0xe0, 0x07, 0xfc, 0x03, 0xff, 0xff, 0xfc, 0x03,
   0xe0, 0x07, 0x00, 0x07, 0xe0, 0x00, 0xe0, 0x07, 0xc0, 0x3f, 0xff, 0xff,
   0xc0, 0x3f, 0xe0, 0x07, 0xe0, 0x00, 0x00, 0x00};
```

14.221 Fl.cxx File Reference

Implementation of the member functions of class Fl.

```
#include "Fl_Private.H"
#include <FL/platform.H>
#include "Fl_Screen_Driver.H"
#include "Fl_Window_Driver.H"
#include "Fl_System_Driver.H"
#include "Fl_Timeout.h"
#include <FL/Fl_Window.H>
#include <FL/Fl_Tooltip.H>
#include <FL/fl_draw.H>
#include <stdlib.h>
#include "flstring.h"
```

Macros

- #define **FOREVER** 1e20

Functions

- bool **fl_clipboard_notify_empty** (void)
- void fl_close_display ()

  Closes the connection to the windowing system when that's possible.
- const char ∗ fl_filename_name (const char ∗name)

  Gets the file name from a path.
- Fl_Window ∗ fl_find (Window xid)

  Returns the Fl_Window that corresponds to the given window reference, or `NULL` if not found.
- void **fl_fix_focus** ()
- void fl_open_callback (void(∗cb)(const char ∗))

  Register a function called for each file dropped onto an application icon.
- void fl_open_display ()

  Opens the display.

- int **fl_send_system_handlers** (void ∗e)
- void **fl_throw_focus** (Fl_Widget ∗o)
- void **fl_trigger_clipboard_notify** (int source)
- Window **fl_xid_** (const Fl_Window ∗w)

Variables

- bool fl_disable_wayland = true

  Prevent the FLTK library from using its Wayland backend and force it to use its X11 backend.
- const char ∗ **fl_local_alt** = "Alt"

  string pointer used in shortcuts, you can change it to another language
- const char ∗ **fl_local_ctrl** = "Ctrl"

  string pointer used in shortcuts, you can change it to another language
- int(∗ **fl_local_grab**)(int)
- const char ∗ **fl_local_meta** = "Meta"

  string pointer used in shortcuts, you can change it to another language
- const char ∗ **fl_local_shift** = "Shift"

  string pointer used in shortcuts, you can change it to another language
- Fl_Widget ∗ **fl_oldfocus**
- Fl_Widget ∗ **fl_selection_requestor**

14.221.1 Detailed Description

Implementation of the member functions of class Fl.

14.221.2 Function Documentation

14.221.2.1 fl_close_display()

```
void fl_close_display ( )
```

Closes the connection to the windowing system when that's possible.
You do *not* need to call this to exit, and in fact it is faster to not do so. It may be useful to call this if you want your program to continue without a GUI. You cannot open the display again, and cannot call any FLTK functions.

Note

Requires #include <FL/platform.H>

14.221.2.2 fl_find()

```
Fl_Window * fl_find (
            Window xid )
```

Returns the Fl_Window that corresponds to the given window reference, or `NULL` if not found.

Deprecated Kept in the X11, Windows, and macOS platforms for compatibility with FLTK versions before 1.4. Please use fl_x11_find(Window), fl_wl_find(struct wld_window∗), fl_win32_find(HWND) or fl_mac_find(FLWindow∗) with FLTK 1.4.0 and above.

14.221.2.3 fl_open_display()

```
void fl_open_display ( )
```

Opens the display.
Automatically called by the library when the first window is show()'n. Does nothing if the display is already open.

Note

Requires #include <FL/platform.H>

14.221.3 Variable Documentation

14.221.3.1 fl_disable_wayland

```
bool fl_disable_wayland = true
```

Prevent the FLTK library from using its Wayland backend and force it to use its X11 backend.
Put this declaration somewhere in your source code outside the body of any function:

```
FL_EXPORT bool fl_disable_wayland = true;
```

This declaration makes sure that source code developed for FLTK 1.3, including X11-specific code, will build and run with FLTK 1.4 and its Wayland platform with a single line source code level change. This declaration has no effect on non-Wayland platforms. Don't add this declaration if you want the Wayland backend to be used when it's available.

Note

Please see also chapter 2.1 of README.Wayland.txt for further information on how to build your application to ensure that this declaration is effective.

Since

1.4.0

14.222 fl_arc.cxx File Reference

Utility functions for drawing arcs and circles.

```
#include <FL/fl_draw.H>
#include <FL/math.h>
```

14.222.1 Detailed Description

Utility functions for drawing arcs and circles.

14.223 fl_ask.cxx File Reference

Utility functions for common dialogs.

```
#include <FL/Fl.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Input_.H>
#include "flstring.h"
#include "Fl_Screen_Driver.H"
#include <FL/fl_ask.H>
#include "Fl_Message.h"
#include <stdio.h>
#include <stdarg.h>
```

Functions

- void fl_alert (const char *fmt,...)

  Shows an alert message dialog box.
- void fl_beep (int type)

  Emits a system beep.
- int fl_choice (const char *fmt, const char *b0, const char *b1, const char *b2,...)

  Shows a dialog displaying the printf style `fmt` message.
- int fl_choice_n (const char *fmt, const char *b0, const char *b1, const char *b2,...)

  Shows a dialog displaying the printf style `fmt` message.
- const char * fl_input (const char *fmt, const char *defstr,...)

Shows an input dialog displaying the `fmt` message with variable arguments.

- const char * fl_input (int maxchar, const char *fmt, const char *defstr,...)

  Shows an input dialog displaying the `fmt` message with variable arguments.
- std::string fl_input_str (int &ret, int maxchar, const char *fmt, const char *defstr,...)

  Shows an input dialog displaying the `fmt` message with variable arguments.
- std::string fl_input_str (int maxchar, const char *fmt, const char *defstr,...)

  Shows an input dialog displaying the `fmt` message with variable arguments.
- void fl_message (const char *fmt,...)

  Shows an information message dialog box.
- int fl_message_hotspot ()

  Gets whether or not to move the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password() to follow the mouse pointer.
- void fl_message_hotspot (int enable)

  Sets whether or not to move the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password() to follow the mouse pointer.
- Fl_Widget * fl_message_icon ()

  Gets the Fl_Box icon container of the current default dialog used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().
- void fl_message_icon_label (const char *str)

  Sets the icon label of the dialog window used in many common dialogs.
- void fl_message_position (const int x, const int y, const int center)

  Sets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().
- void fl_message_position (Fl_Widget *widget)

  Sets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().
- int fl_message_position (int *x, int *y)

  Gets the preferred position for the message box used in many common dialogs like fl_message(), fl_alert(), fl_ask(), fl_choice(), fl_input(), fl_password().
- void fl_message_title (const char *title)

  Sets the title of the dialog window used in many common dialogs.
- void fl_message_title_default (const char *title)

  Sets the default title of the dialog window used in many common dialogs.
- const char * fl_password (const char *fmt, const char *defstr,...)

  Shows an input dialog displaying the `fmt` message with variable arguments.
- const char * fl_password (int maxchar, const char *fmt, const char *defstr,...)

  Shows an input dialog displaying the `fmt` message with variable arguments.
- std::string fl_password_str (int &ret, int maxchar, const char *fmt, const char *defstr,...)

  Shows an input dialog displaying the `fmt` message with variable arguments.
- std::string fl_password_str (int maxchar, const char *fmt, const char *defstr,...)

  Shows an input dialog displaying the `fmt` message with variable arguments.

Variables

- const char * **fl_cancel** = "Cancel"

  string pointer used in common dialogs, you can change it to another language
- const char * **fl_close** = "Close"

  string pointer used in common dialogs, you can change it to another language
- Fl_Font **fl_message_font_** = FL_HELVETICA
- Fl_Fontsize **fl_message_size_** = -1
- const char * **fl_no** = "No"

  string pointer used in common dialogs, you can change it to another language
- const char * **fl_ok** = "OK"

string pointer used in common dialogs, you can change it to another language

- const char ∗ **fl_yes** = "Yes"

  string pointer used in common dialogs, you can change it to another language

14.223.1 Detailed Description

Utility functions for common dialogs.
This file defines the functions

- fl_alert()
- fl_beep()
- fl_message()
- fl_ask() // deprecated since 1.1.7 (2006), see comment in FL/fl_ask.H
- fl_choice()
- fl_input()
- fl_input_str()
- fl_password()
- fl_password_str()

and some more functions to change their behavior (positioning, window title, and more).
Since FLTK 1.4.0 a big part of these functions is implemented in class Fl_Message.

14.224 fl_boxtype.cxx File Reference

Drawing code for common box types.

```
#include "src/Fl_Private.H"
#include <FL/Fl_Widget.H>
#include <FL/fl_draw.H>
#include <config.h>
```

Macros

- #define **D1** BORDER_WIDTH
- #define **D2** (BORDER_WIDTH+BORDER_WIDTH)
- #define **fl_border_box** fl_rectbound

  allow consistent naming

Typedefs

- typedef struct BOXTABLE **boxtable_t**

Functions

- void **fl_border_frame** (int x, int y, int w, int h, Fl_Color c)

  Draws a frame of type FL_BORDER_FRAME.
- void **fl_diamond_down_box** (int x, int y, int w, int h, Fl_Color bgcolor)
- void **fl_diamond_focus** (Fl_Boxtype, int, int, int, int, Fl_Color, Fl_Color)
- void **fl_diamond_up_box** (int x, int y, int w, int h, Fl_Color bgcolor)
- void **fl_down_box** (int x, int y, int w, int h, Fl_Color c)

  Draws a box of type FL_DOWN_BOX.
- void **fl_down_frame** (int x, int y, int w, int h, Fl_Color)

Draws a frame of type FL_DOWN_FRAME.

- void fl_draw_box (Fl_Boxtype t, int x, int y, int w, int h, Fl_Color c)

  Draws a box using given type, position, size and color.
- void fl_draw_box_focus (Fl_Boxtype bt, int x, int y, int w, int h, Fl_Color fg, Fl_Color bg)

  Draws the focus rectangle inside a box using given type, position, size and color.
- void **fl_embossed_box** (int x, int y, int w, int h, Fl_Color c)

  Draws a box of type FL_EMBOSSED_BOX.
- void **fl_embossed_frame** (int x, int y, int w, int h, Fl_Color)

  Draws a frame of type FL_EMBOSSED_FRAME.
- void **fl_engraved_box** (int x, int y, int w, int h, Fl_Color c)

  Draws a box of type FL_ENGRAVED_BOX.
- void **fl_engraved_frame** (int x, int y, int w, int h, Fl_Color)

  Draws a frame of type FL_ENGRAVED_FRAME.
- void **fl_flat_box** (int x, int y, int w, int h, Fl_Color c)

  Draws a box of type FL_FLAT_BOX.
- void fl_frame (const char *s, int x, int y, int w, int h)

  Draws a series of line segments around the given box.
- void fl_frame2 (const char *s, int x, int y, int w, int h)

  Draws a series of line segments around the given box.
- void **fl_gleam_down_box** (int x, int y, int w, int h, Fl_Color c)
- void **fl_gleam_down_frame** (int x, int y, int w, int h, Fl_Color c)
- void **fl_gleam_thin_down_box** (int x, int y, int w, int h, Fl_Color c)
- void **fl_gleam_thin_up_box** (int x, int y, int w, int h, Fl_Color c)
- void **fl_gleam_up_box** (int x, int y, int w, int h, Fl_Color c)
- void **fl_gleam_up_frame** (int x, int y, int w, int h, Fl_Color c)
- const uchar * **fl_gray_ramp** ()
- void **fl_gtk_down_box** (int x, int y, int w, int h, Fl_Color c)
- void **fl_gtk_down_frame** (int x, int y, int w, int h, Fl_Color c)
- void **fl_gtk_round_down_box** (int x, int y, int w, int h, Fl_Color c)
- void **fl_gtk_round_up_box** (int x, int y, int w, int h, Fl_Color c)
- void **fl_gtk_thin_down_box** (int x, int y, int w, int h, Fl_Color c)
- void **fl_gtk_thin_down_frame** (int x, int y, int w, int h, Fl_Color c)
- void **fl_gtk_thin_up_box** (int x, int y, int w, int h, Fl_Color c)
- void **fl_gtk_thin_up_frame** (int x, int y, int w, int h, Fl_Color c)
- void **fl_gtk_up_box** (int x, int y, int w, int h, Fl_Color c)
- void **fl_gtk_up_frame** (int x, int y, int w, int h, Fl_Color c)
- void **fl_no_box** (int, int, int, int, Fl_Color)

  Draws a box of type FL_NO_BOX.
- void **fl_oval_box** (int x, int y, int w, int h, Fl_Color c)
- void **fl_oval_flat_box** (int x, int y, int w, int h, Fl_Color c)
- void **fl_oval_focus** (Fl_Boxtype bt, int x, int y, int w, int h, Fl_Color fg, Fl_Color bg)
- void **fl_oval_frame** (int x, int y, int w, int h, Fl_Color c)
- void **fl_oval_shadow_box** (int x, int y, int w, int h, Fl_Color c)
- void **fl_oxy_button_down_box** (int x, int y, int w, int h, Fl_Color col)
- void **fl_oxy_button_up_box** (int x, int y, int w, int h, Fl_Color col)
- void **fl_oxy_down_box** (int x, int y, int w, int h, Fl_Color col)
- void **fl_oxy_down_frame** (int x, int y, int w, int h, Fl_Color col)
- void **fl_oxy_round_down_box** (int x, int y, int w, int h, Fl_Color col)
- void **fl_oxy_round_up_box** (int x, int y, int w, int h, Fl_Color col)
- void **fl_oxy_thin_down_box** (int x, int y, int w, int h, Fl_Color col)
- void **fl_oxy_thin_down_frame** (int x, int y, int w, int h, Fl_Color col)
- void **fl_oxy_thin_up_box** (int x, int y, int w, int h, Fl_Color col)
- void **fl_oxy_thin_up_frame** (int x, int y, int w, int h, Fl_Color col)

- void **fl_oxy_up_box** (int x, int y, int w, int h, Fl_Color col)
- void **fl_oxy_up_frame** (int x, int y, int w, int h, Fl_Color col)
- void **fl_plastic_down_box** (int x, int y, int w, int h, Fl_Color c)
- void **fl_plastic_down_frame** (int x, int y, int w, int h, Fl_Color c)
- void **fl_plastic_down_round** (int x, int y, int w, int h, Fl_Color c)
- void **fl_plastic_thin_up_box** (int x, int y, int w, int h, Fl_Color c)
- void **fl_plastic_up_box** (int x, int y, int w, int h, Fl_Color c)
- void **fl_plastic_up_frame** (int x, int y, int w, int h, Fl_Color c)
- void **fl_plastic_up_round** (int x, int y, int w, int h, Fl_Color c)
- void fl_rectbound (int x, int y, int w, int h, Fl_Color bgcolor)

  Draws a bounded rectangle with a given position, size and color.
- void **fl_rflat_box** (int x, int y, int w, int h, Fl_Color c)
- void **fl_round_down_box** (int x, int y, int w, int h, Fl_Color bgcolor)
- void **fl_round_focus** (Fl_Boxtype, int, int, int, int, Fl_Color, Fl_Color)
- void **fl_round_up_box** (int x, int y, int w, int h, Fl_Color bgcolor)
- void **fl_rounded_box** (int x, int y, int w, int h, Fl_Color c)
- void **fl_rounded_focus** (Fl_Boxtype bt, int x, int y, int w, int h, Fl_Color fg, Fl_Color bg)
- void **fl_rounded_frame** (int x, int y, int w, int h, Fl_Color c)
- void **fl_rshadow_box** (int x, int y, int w, int h, Fl_Color c)
- void **fl_shadow_box** (int x, int y, int w, int h, Fl_Color c)
- void **fl_shadow_frame** (int x, int y, int w, int h, Fl_Color c)
- void **fl_thin_down_box** (int x, int y, int w, int h, Fl_Color c)

  Draws a box of type FL_THIN_DOWN_BOX.
- void **fl_thin_down_frame** (int x, int y, int w, int h, Fl_Color)

  Draws a frame of type FL_THIN_DOWN_FRAME.
- void **fl_thin_up_box** (int x, int y, int w, int h, Fl_Color c)

  Draws a box of type FL_THIN_UP_BOX.
- void **fl_thin_up_frame** (int x, int y, int w, int h, Fl_Color)

  Draws a frame of type FL_THIN_UP_FRAME.
- void **fl_up_box** (int x, int y, int w, int h, Fl_Color c)

  Draws a box of type FL_UP_BOX.
- void **fl_up_frame** (int x, int y, int w, int h, Fl_Color)

  Draws a frame of type FL_UP_FRAME.

14.224.1 Detailed Description

Drawing code for common box types.

14.224.2 Function Documentation

14.224.2.1 fl_rectbound()

```
void fl_rectbound (
            int x,
            int y,
            int w,
            int h,
            Fl_Color bgcolor )
```

Draws a bounded rectangle with a given position, size and color.
Equivalent to drawing a box of type FL_BORDER_BOX.

14.225 fl_cmap.h

```
//
// DO NOT EDIT THIS FILE !
//
// This file must be generated by "util/cmap.cxx".
// See instructions in this file.
//
// Copyright 1998-2026 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//
    0x00000000, //   0
    0xff000000, //   1
    0x00ff0000, //   2
    0xffff0000, //   3
    0x0000ff00, //   4
    0xff00ff00, //   5
    0x00ffff00, //   6
    0xffffff00, //   7
    0x55555500, //   8
    0xc6717100, //   9
    0x71c67100, //  10
    0x8e8e3800, //  11
    0x7171c600, //  12
    0x8e388e00, //  13
    0x388e8e00, //  14
    0x00008000, //  15
    0xa8a89800, //  16
    0xe8e8d800, //  17
    0x68685800, //  18
    0x98a8a800, //  19
    0xd8e8e800, //  20
    0x58686800, //  21
    0x9c9ca800, //  22
    0xdcdce800, //  23
    0x5c5c6800, //  24
    0x9ca89c00, //  25
    0xdce8dc00, //  26
    0x5c685c00, //  27
    0x90909000, //  28
    0xc0c0c000, //  29
    0x50505000, //  30
    0xa0a0a000, //  31
    0x00000000, //  32
    0x0d0d0d00, //  33
    0x1a1a1a00, //  34
    0x26262600, //  35
    0x31313100, //  36
    0x3d3d3d00, //  37
    0x48484800, //  38
    0x55555500, //  39
    0x5f5f5f00, //  40
    0x6a6a6a00, //  41
    0x75757500, //  42
    0x80808000, //  43
    0x8a8a8a00, //  44
    0x95959500, //  45
    0xa0a0a000, //  46
    0xaaaaaa00, //  47
    0xb5b5b500, //  48
    0xc0c0c000, //  49
    0xcbcbcb00, //  50
    0xd5d5d500, //  51
    0xe0e0e000, //  52
    0xeaeaea00, //  53
    0xf5f5f500, //  54
    0xffffff00, //  55
    0x00000000, //  56
    0x00240000, //  57
    0x00480000, //  58
    0x006d0000, //  59
    0x00910000, //  60
    0x00b60000, //  61
    0x00da0000, //  62
    0x00ff0000, //  63
    0x3f000000, //  64
    0x3f240000, //  65
    0x3f480000, //  66
```

```
    0x3f6d0000, //  67
    0x3f910000, //  68
    0x3fb60000, //  69
    0x3fda0000, //  70
    0x3fff0000, //  71
    0x7f000000, //  72
    0x7f240000, //  73
    0x7f480000, //  74
    0x7f6d0000, //  75
    0x7f910000, //  76
    0x7fb60000, //  77
    0x7fda0000, //  78
    0x7fff0000, //  79
    0xbf000000, //  80
    0xbf240000, //  81
    0xbf480000, //  82
    0xbf6d0000, //  83
    0xbf910000, //  84
    0xbfb60000, //  85
    0xbfda0000, //  86
    0xbfff0000, //  87
    0xff000000, //  88
    0xff240000, //  89
    0xff480000, //  90
    0xff6d0000, //  91
    0xff910000, //  92
    0xffb60000, //  93
    0xffda0000, //  94
    0xffff0000, //  95
    0x00003f00, //  96
    0x00243f00, //  97
    0x00483f00, //  98
    0x006d3f00, //  99
    0x00913f00, // 100
    0x00b63f00, // 101
    0x00da3f00, // 102
    0x00ff3f00, // 103
    0x3f003f00, // 104
    0x3f243f00, // 105
    0x3f483f00, // 106
    0x3f6d3f00, // 107
    0x3f913f00, // 108
    0x3fb63f00, // 109
    0x3fda3f00, // 110
    0x3fff3f00, // 111
    0x7f003f00, // 112
    0x7f243f00, // 113
    0x7f483f00, // 114
    0x7f6d3f00, // 115
    0x7f913f00, // 116
    0x7fb63f00, // 117
    0x7fda3f00, // 118
    0x7fff3f00, // 119
    0xbf003f00, // 120
    0xbf243f00, // 121
    0xbf483f00, // 122
    0xbf6d3f00, // 123
    0xbf913f00, // 124
    0xbfb63f00, // 125
    0xbfda3f00, // 126
    0xbfff3f00, // 127
    0xff003f00, // 128
    0xff243f00, // 129
    0xff483f00, // 130
    0xff6d3f00, // 131
    0xff913f00, // 132
    0xffb63f00, // 133
    0xffda3f00, // 134
    0xffff3f00, // 135
    0x00007f00, // 136
    0x00247f00, // 137
    0x00487f00, // 138
    0x006d7f00, // 139
    0x00917f00, // 140
    0x00b67f00, // 141
    0x00da7f00, // 142
    0x00ff7f00, // 143
    0x3f007f00, // 144
    0x3f247f00, // 145
    0x3f487f00, // 146
    0x3f6d7f00, // 147
    0x3f917f00, // 148
    0x3fb67f00, // 149
    0x3fda7f00, // 150
    0x3fff7f00, // 151
    0x7f007f00, // 152
    0x7f247f00, // 153
```

```
    0x7f487f00, // 154
    0x7f6d7f00, // 155
    0x7f917f00, // 156
    0x7fb67f00, // 157
    0x7fda7f00, // 158
    0x7fff7f00, // 159
    0xbf007f00, // 160
    0xbf247f00, // 161
    0xbf487f00, // 162
    0xbf6d7f00, // 163
    0xbf917f00, // 164
    0xbfb67f00, // 165
    0xbfda7f00, // 166
    0xbfff7f00, // 167
    0xff007f00, // 168
    0xff247f00, // 169
    0xff487f00, // 170
    0xff6d7f00, // 171
    0xff917f00, // 172
    0xffb67f00, // 173
    0xffda7f00, // 174
    0xffff7f00, // 175
    0x0000bf00, // 176
    0x0024bf00, // 177
    0x0048bf00, // 178
    0x006dbf00, // 179
    0x0091bf00, // 180
    0x00b6bf00, // 181
    0x00dabf00, // 182
    0x00ffbf00, // 183
    0x3f00bf00, // 184
    0x3f24bf00, // 185
    0x3f48bf00, // 186
    0x3f6dbf00, // 187
    0x3f91bf00, // 188
    0x3fb6bf00, // 189
    0x3fdabf00, // 190
    0x3fffbf00, // 191
    0x7f00bf00, // 192
    0x7f24bf00, // 193
    0x7f48bf00, // 194
    0x7f6dbf00, // 195
    0x7f91bf00, // 196
    0x7fb6bf00, // 197
    0x7fdabf00, // 198
    0x7fffbf00, // 199
    0xbf00bf00, // 200
    0xbf24bf00, // 201
    0xbf48bf00, // 202
    0xbf6dbf00, // 203
    0xbf91bf00, // 204
    0xbfb6bf00, // 205
    0xbfdabf00, // 206
    0xbfffbf00, // 207
    0xff00bf00, // 208
    0xff24bf00, // 209
    0xff48bf00, // 210
    0xff6dbf00, // 211
    0xff91bf00, // 212
    0xffb6bf00, // 213
    0xffdabf00, // 214
    0xffffbf00, // 215
    0x0000ff00, // 216
    0x0024ff00, // 217
    0x0048ff00, // 218
    0x006dff00, // 219
    0x0091ff00, // 220
    0x00b6ff00, // 221
    0x00daff00, // 222
    0x00ffff00, // 223
    0x3f00ff00, // 224
    0x3f24ff00, // 225
    0x3f48ff00, // 226
    0x3f6dff00, // 227
    0x3f91ff00, // 228
    0x3fb6ff00, // 229
    0x3fdaff00, // 230
    0x3fffff00, // 231
    0x7f00ff00, // 232
    0x7f24ff00, // 233
    0x7f48ff00, // 234
    0x7f6dff00, // 235
    0x7f91ff00, // 236
    0x7fb6ff00, // 237
    0x7fdaff00, // 238
    0x7fffff00, // 239
    0xbf00ff00, // 240
```

```
    0xbf24ff00, // 241
    0xbf48ff00, // 242
    0xbf6dff00, // 243
    0xbf91ff00, // 244
    0xbfb6ff00, // 245
    0xbfdaff00, // 246
    0xbfffff00, // 247
    0xff00ff00, // 248
    0xff24ff00, // 249
    0xff48ff00, // 250
    0xff6dff00, // 251
    0xff91ff00, // 252
    0xffb6ff00, // 253
    0xffdaff00, // 254
    0xffffff00  // 255
//
// End of src/fl_cmap.h - generated by util/cmap.cxx
//
```

14.226 fl_color.cxx File Reference

Color handling.

```
#include <FL/Fl.H>
#include <FL/Fl_Device.H>
#include <FL/Fl_Graphics_Driver.H>
#include "fl_cmap.h"
```

Functions

- Fl_Color fl_color_average (Fl_Color color1, Fl_Color color2, float weight)

  Returns the weighted average color between the two given colors.
- Fl_Color **fl_inactive** (Fl_Color c)

  Returns the inactive, dimmed version of the given color.

Variables

- unsigned fl_cmap [256]

14.226.1 Detailed Description

Color handling.

14.226.2 Variable Documentation

14.226.2.1 fl_cmap

```
unsigned fl_cmap[256]
```

Initial value:

```
= {

}
```

14.227 Fl_compose.cxx File Reference

Utility functions to support text input.

```
#include <FL/Fl.H>
#include "Fl_Screen_Driver.H"
```

14.227.1 Detailed Description

Utility functions to support text input.

14.228 fl_contrast.cxx File Reference

Color contrast handling.

```
#include <FL/Fl.H>
#include <math.h>
```

Macros

- #define **DEBUG_CONTRAST_LEGACY** 0

Functions

- Fl_Color fl_contrast (Fl_Color fg, Fl_Color bg, int context, int size)

  Returns a color that contrasts with the background color.
- void fl_contrast_function (Fl_Contrast_Function *f)

  Register a custom contrast function.
- int fl_contrast_level ()

  Get the contrast level (sensitivity) of the fl_contrast() method.
- void fl_contrast_level (int level)

  Set the contrast level (sensitivity) of the fl_contrast() method.
- int fl_contrast_mode ()

  Return the current contrast algorithm (mode).
- void fl_contrast_mode (int mode)

  Set the contrast algorithm (mode).
- double fl_lightness (Fl_Color color)

  Return the perceived lightness of a color.
- double fl_luminance (Fl_Color color)

  Return the raw / physical luminance of a color.
- unsigned **get_color** (Fl_Color i)

Variables

- unsigned **fl_cmap** [256]

14.228.1 Detailed Description

Color contrast handling.
Implementation of fl_contrast() and its variants.

14.229 fl_curve.cxx File Reference

Utility for drawing Bézier curves, adding the points to the current fl_begin/fl_vertex/fl_end path.

```
#include <FL/fl_draw.H>
#include <math.h>
```

14.229.1 Detailed Description

Utility for drawing Bézier curves, adding the points to the current fl_begin/fl_vertex/fl_end path.
Incremental math implementation: I very much doubt this is optimal! From Foley/vanDam page 511. If anybody has a better algorithm, please send it!

14.230 Fl_Double_Window.cxx File Reference

Fl_Double_Window implementation.

```
#include <FL/Fl.H>
#include <FL/platform.H>
#include <FL/Fl_Double_Window.H>
#include <FL/fl_draw.H>
#include "Fl_Window_Driver.H"
```

14.230.1 Detailed Description

Fl_Double_Window implementation.

14.231 Fl_get_system_colors.cxx File Reference

System color support.

```
#include <FL/Fl.H>
#include "Fl_Screen_Driver.H"
#include "Fl_System_Driver.H"
#include <FL/fl_draw.H>
#include <FL/platform.H>
#include <FL/math.h>
#include <FL/fl_utf8.h>
#include <FL/fl_string_functions.h>
#include "flstring.h"
#include <stdio.h>
#include <stdlib.h>
#include <FL/Fl_Pixmap.H>
#include <FL/Fl_Tiled_Image.H>
#include "tile.xpm"
```

Macros

- #define **D1** BORDER_WIDTH
- #define **D2** (BORDER_WIDTH+BORDER_WIDTH)

Functions

- void **fl_down_box** (int, int, int, int, Fl_Color)

  Draws a box of type FL_DOWN_BOX.
- void **fl_down_frame** (int, int, int, int, Fl_Color)

  Draws a frame of type FL_DOWN_FRAME.
- int fl_parse_color (const char *p, uchar &r, uchar &g, uchar &b)

  Parse a string containing a description of a color and write r, g, and b.
- void **fl_round_down_box** (int, int, int, int, Fl_Color)
- void **fl_round_focus** (Fl_Boxtype, int, int, int, int, Fl_Color, Fl_Color)
- void **fl_round_up_box** (int, int, int, int, Fl_Color)
- void **fl_thin_down_box** (int, int, int, int, Fl_Color)

  Draws a box of type FL_THIN_DOWN_BOX.
- void **fl_thin_down_frame** (int, int, int, int, Fl_Color)

  Draws a frame of type FL_THIN_DOWN_FRAME.
- void **fl_thin_up_box** (int, int, int, int, Fl_Color)

  Draws a box of type FL_THIN_UP_BOX.
- void **fl_thin_up_frame** (int, int, int, int, Fl_Color)

Draws a frame of type FL_THIN_UP_FRAME.

- void **fl_up_box** (int, int, int, int, Fl_Color)

  Draws a box of type FL_UP_BOX.

- void **fl_up_frame** (int, int, int, int, Fl_Color)

  Draws a frame of type FL_UP_FRAME.

Variables

- const char * **fl_bg** = NULL
- const char * **fl_bg2** = NULL
- const char * **fl_fg** = NULL

14.231.1 Detailed Description

System color support.

14.231.2 Function Documentation

14.231.2.1 fl_parse_color()

```
int fl_parse_color (
            const char * p,
            uchar & r,
            uchar & g,
            uchar & b )
```

Parse a string containing a description of a color and write r, g, and b.
This call is used by the Pixmap file format interpreter and by the command line arguments parser to set UI colors.
RGB color triplets usually start with a '#' character, but it can be omitted if it does not conflict with the later rules.
Color components are defined in hexadecimal notation with 1, 2, 3, or four hex digits per component, making color triplets 3, 6, 9, or 12 characters long. The interpreter is case insensitive. Valid code examples include "FF0000" for red, "#0F0" for green, and "000000004444" for a dark blue.
On the X11 platform, color values can also be given a color name like "red". The list of accepted color names is provided by the X11 server.
If none of the color interpretations work, fl_parse_color returns 0. The Pixmap reader interprets those as transparent, and are usually written as "None", "#transparent", or "bg".

Parameters

in	*p*	a C-string describing the color
out	*r,g,b*	the color components in the 0...255 range

Returns

0 if the color cannot be interpreted, 1 otherwise

14.232 Fl_Gl_Choice.H

```
00001 //
00002 // OpenGL definitions for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 1998-2018 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file.  If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 //     https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 //     https://www.fltk.org/bugs.php
00015 //
```

```

// Internal interface to set up OpenGL.
//
// A "Fl_Gl_Choice" is created from an OpenGL mode and holds information
// necessary to create a window (on X) and to create an OpenGL "context"
// (on both X and Win32).
//
// create_gl_context takes a window (necessary only on Win32) and an
// Fl_Gl_Choice and returns a new OpenGL context. All contexts share
// display lists with each other.
//
// On X another create_gl_context is provided to create it for any
// X visual.
//
// set_gl_context makes the given OpenGL context current and makes
// it draw into the passed window. It tracks the current one context
// to avoid calling the context switching code when the same context
// is used, though it is a mystery to me why the GLX/WGL libraries
// don't do this themselves...
//
// delete_gl_context destroys the context.
//
// This code is used by Fl_Gl_Window, gl_start(), and gl_visual()

#ifndef Fl_Gl_Choice_H
#define Fl_Gl_Choice_H

// Describes the platform-independent part of data needed to create a GLContext.
class Fl_Gl_Choice {
  friend class Fl_Gl_Window_Driver;
  int mode;
  const int *alist;
  Fl_Gl_Choice *next;
public:
  Fl_Gl_Choice(int m, const int *alistp, Fl_Gl_Choice *n) : mode(m), alist(alistp), next(n) {}
};

#endif // Fl_Gl_Choice_H
```

14.233 Fl_Gl_Window_Driver.H

```
//
// Definition of class Fl_Gl_Window_Driver, and of its platform-specific derived classes
// for the Fast Light Tool Kit (FLTK).
//
// Copyright 2016-2018 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef Fl_Gl_Window_Driver_H
#define Fl_Gl_Window_Driver_H

#include <FL/Fl_Gl_Window.H>
#include <FL/gl.h> // for GLint

class Fl_Gl_Choice;
class Fl_Font_Descriptor;

/* The constructor of each Fl_Gl_Window object creates also an object from a
 platform-specific derived class from this class.
 */
class Fl_Gl_Window_Driver {
protected:
  GLint current_prog;
  Fl_Gl_Window *pWindow;
public:
  static Fl_Window* cached_window;
  static int nContext;
  static GLContext *context_list;
  static Fl_Gl_Choice *first;
  static int copy;
  static float gl_scale;
  static GLContext gl_start_context;
  Fl_Gl_Choice* g() {return pWindow->g;}
  void g(Fl_Gl_Choice *c) {pWindow->g = c;}
  int mode() {return pWindow->mode_;}
```

```
  void mode(int m) { pWindow->mode_ = m;}
  const int *alist() {return pWindow->alist;}
  void alist(const int *l) { pWindow->alist = l;}
  void* overlay() {return pWindow->overlay;}
  void draw_overlay() {pWindow->draw_overlay();}

  Fl_Gl_Window_Driver(Fl_Gl_Window *win) : pWindow(win) {current_prog=0;}
  virtual ~Fl_Gl_Window_Driver() {}
  static Fl_Gl_Window_Driver *newGlWindowDriver(Fl_Gl_Window *w);
  static Fl_Gl_Window_Driver *global();
  virtual float pixels_per_unit() {return 1;}
  virtual void before_show(int&) {}
  virtual void after_show() {}
  virtual void invalidate();
  virtual int mode_(int /*m*/, const int * /*a*/) {return 0;}
  virtual void make_current_before() {}
  virtual void make_current_after() {}
  virtual void swap_buffers() {}
  virtual void resize(int /*is_a_resize*/, int /*w*/, int /*h*/) {}
  virtual char swap_type();
  virtual void swap_interval(int) { }
  virtual int swap_interval() const { return -1; }
  virtual int flush_begin(char&) {return 0;}
  virtual void gl_hide_before(void *&) {} // the default implementation may be enough
  static Fl_Gl_Choice *find_begin(int m, const int *alistp);
  static void add_context(GLContext ctx);
  static void del_context(GLContext ctx);
  // Return one of these structures for a given gl mode.
  // The second argument is a glX attribute list, and is used if mode is zero.
  // This is not supported on Win32:
  virtual Fl_Gl_Choice *find(int /*mode*/, const int * /*alistp*/) {return NULL;}
  virtual GLContext create_gl_context(Fl_Window*, const Fl_Gl_Choice*) {return 0;}
  virtual void set_gl_context(Fl_Window*, GLContext) {}
  virtual void delete_gl_context(GLContext) {}
  virtual void make_overlay(void* &o);
  virtual void hide_overlay() {} // the default implementation may be enough
  virtual void make_overlay_current() {}
  virtual void redraw_overlay() {}
  virtual int can_do_overlay() {return 0;}
  virtual void waitGL() {} // support for gl_finish() function
  virtual void gl_visual(Fl_Gl_Choice*); // support for Fl::gl_visual() function
  virtual void gl_start() {} // support for gl_start() function
  virtual void* GetProcAddress(const char *procName); // support for glutGetProcAddress()
  virtual void draw_string_legacy(const char* str, int n); // support for gl_draw()
  void draw_string_legacy_get_list(const char* str, int n); // support for gl_draw()
  static void draw_string_legacy_glut(const char* str, int n); // support for gl_draw()
  virtual void get_list(Fl_Font_Descriptor*, int) {} // support for gl_draw() without textures
  virtual void gl_bitmap_font(Fl_Font_Descriptor *) {} // support for gl_font() without textures
  virtual int overlay_color(Fl_Color) {return 0;} // support for gl_color() with HAVE_GL_OVERLAY
  static void draw_string_with_texture(const char* str, int n); // cross-platform
  // support for gl_draw(). The cross-platform version may be enough.
  virtual char *alpha_mask_for_string(const char *str, int n, int w, int h, Fl_Fontsize fs);
  virtual int genlistsize() { return 0; } // support for gl_draw()
  virtual Fl_Font_Descriptor** fontnum_to_fontdescriptor(int fnum);
  virtual Fl_RGB_Image* capture_gl_rectangle(int x, int y, int w, int h);
  static inline Fl_Gl_Window_Driver* driver(const Fl_Gl_Window *win) {return win->pGlWindowDriver;}
  // true means the platform uses glScissor() to make sure GL subwindows
  // don't leak outside their parent window
  virtual bool need_scissor() { return false; }
  virtual void switch_to_GL1();
  virtual void switch_back();
};

#endif /* Fl_Gl_Window_Driver_H */
```

14.234 Fl_Graphics_Driver.cxx File Reference

Implementation of class Fl_Graphics_Driver.

```
#include <config.h>
#include <FL/Fl_Graphics_Driver.H>
#include "Fl_Scalable_Graphics_Driver.H"
```

Variables

- Fl_Graphics_Driver * **fl_graphics_driver**

  Points to the driver that currently receives all graphics requests.

14.234.1 Detailed Description

Implementation of class Fl_Graphics_Driver.

14.235 Fl_Grid.cxx File Reference

Implements the Fl_Grid container widget.

```
#include <FL/Fl_Grid.H>
#include <FL/fl_draw.H>
```

14.235.1 Detailed Description

Implements the Fl_Grid container widget.

Since

1.4.0

14.236 Fl_Image_Reader.h

```
00001 //
00002 // Internal (Image) Reader class for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 2020-2021 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file.  If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 //     https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 //     https://www.fltk.org/bugs.php
00015 //
00016
00017 /*
00018   This internal (undocumented) class reads data chunks from a file or from
00019   memory in LSB-first byte order.
00020
00021   This class is used in Fl_GIF_Image and Fl_BMP_Image to avoid code
00022   duplication and may be extended to be used in similar cases. Future
00023   options might be to read data in MSB-first byte order or to add more
00024   methods.
00025 */
00026
00027 #ifndef FL_IMAGE_READER_H
00028 #define FL_IMAGE_READER_H
00029
00030 #include <stdio.h>
00031
00032 class Fl_Image_Reader {
00033 public:
00034   // Create the reader.
00035   Fl_Image_Reader()
00036     : is_file_(0)
00037     , is_data_(0)
00038     , file_(0L)
00039     , data_(0L)
00040     , start_(0L)
00041     , end_((const unsigned char *)(-1L))
00042     , name_(0L)
00043     , error_(0) {}
00044
00045   // Initialize the reader to access the file system, filename is copied
00046   // and stored.
00047   int open(const char *filename);
00048
00049   // Initialize the reader for memory access, name is copied and stored
00050   int open(const char *imagename, const unsigned char *data, const size_t datasize);
00051   // Deprecated, DO NOT USE!
00052   int open(const char *imagename, const unsigned char *data);
00053
00054   // Close and destroy the reader
00055   ~Fl_Image_Reader();
00056
```

```
  // Read a single byte from memory or a file
  unsigned char read_byte();

  // Read a 16-bit unsigned integer, LSB-first
  unsigned short read_word();

  // Read a 32-bit unsigned integer, LSB-first
  unsigned int read_dword();

  // Read a 32-bit signed integer, LSB-first
  int read_long() { return (int)read_dword(); }

  // Move the current read position to a byte offset from the beginning
  // of the file or the original start address in memory
  void seek(unsigned int n);

  // Get the current file or memory offset from the beginning
  // of the file or the original start address in memory
  long tell() const;

  // Get the current EOF or error status of the file or data block
  int error() const { return error_; }

  // return the name or filename for this reader
  const char *name() const { return name_; }

  // skip a given number of bytes
  void skip(unsigned int n) { seek((unsigned int)tell() + n); }

  // return 1 if we are reading from a file, 0 otherwise
  char is_file() const { return is_file_; }

  // return 1 if we are reading from a block of memory, 0 otherwise
  char is_data() const { return is_data_; }

  // return the start address of the data block if reading from memory, otherwise nullptr
  const unsigned char *data_start() const { return start_; }

  // return the size of the data block if reading from memoryand the size of
  // the block is known, otherwise 0
  size_t data_size() const {
    if (is_data_ && end_ && end_ != (const unsigned char *)(-1L)) {
      return (size_t)(end_ - start_);
    } else {
      return 0;
    }
  }

private:
  // open() sets this if we read from a file
  char is_file_;
  // open() sets this if we read from memory
  char is_data_;
  // a pointer to the opened file
  FILE *file_;
  // a pointer to the current byte in memory
  const unsigned char *data_;
  // a pointer to the start of the image data
  const unsigned char *start_;
  // a pointer to the end of image data if reading from memory, otherwise undefined
  // note: currently (const unsigned char *)(-1L) if end of memory is not available
  // ... which means "unlimited"
  const unsigned char *end_;
  // a copy of the name associated with this reader
  char *name_;
  // a flag to store EOF or error status
  int error_;
};

#endif // FL_IMAGE_READER_H
```

14.237 Fl_Message.h

```
//
// Common dialog header file for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2022 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
```

```
//
//     https://www.fltk.org/bugs.php
//

#ifndef _src_Fl_Message_h_
#define _src_Fl_Message_h_

#include <FL/Fl_Window.H>
#include <FL/Fl_Box.H>
#include <FL/fl_ask.H>

class Fl_Button;
class Fl_Input;

/* Note: Do not FL_EXPORT this class, it's for internal use only */

class Fl_Message_Box : public Fl_Box {
public:
  Fl_Message_Box(int X, int Y, int W, int H)
    : Fl_Box(X, Y, W, H) {}
  int handle(int e) FL_OVERRIDE;
}; // class Fl_Message_Box

/* Note: Do not FL_EXPORT this class, it's for internal use only */

class Fl_Message {

  // static variables and methods

private:
  static Fl_Box *message_icon_; // returned by Fl_Message::message_icon()

  static const char *message_title_;
  static const char *message_title_default_;

  // icon label for next dialog (STR #2762)
  static const char *message_icon_label_;

  // Note: since Fl_Message objects are destroyed before fl_input()
  // and fl_password() return their input text, we *need* to store
  // the text in an internal (static) buffer. :-(

  static char *input_buffer_; // points to the allocated text buffer
  static int input_size_;     // size of allocated text buffer

  // the callback for all buttons:
  static void button_cb_(Fl_Widget *w, void *d);

  // the window callback:
  static void window_cb_(Fl_Widget *w, void *d);

  // resize to make text and buttons fit
  void resizeform();

public:
  static Fl_Box *message_icon();
  static void message_title(const char *title);
  static void message_title_default(const char *title);
  static void icon_label(const char *str);

  static void message_position(const int x, const int y, const int center) {
    form_x_ = x;
    form_y_ = y;
    form_position_ = center ? 2 : 1;
  }

  static void message_position(Fl_Widget *widget) {
    int xo, yo;
    Fl_Window *win = widget->top_window_offset(xo, yo);
    form_x_ = xo + widget->w() / 2;
    form_y_ = yo + widget->h() / 2;
    if (win) {
      form_x_ += win->x();
      form_y_ += win->y();
    }
    form_position_ = 2;
  }

  static int message_position(int *x, int *y) {
    if (x)
      *x = form_position_ ? form_x_ : -1;
    if (y)
      *y = form_position_ ? form_y_ : -1;
    return form_position_;
  }

  static void message_hotspot(int enable) { enable_hotspot_ = enable ? 1 : 0; }
```

```

  static int message_hotspot() { return enable_hotspot_; }

  int window_closed() const {
    return window_closed_;
  }

  // member variables and methods

private:
  Fl_Window *window_;
  Fl_Message_Box *message_;
  Fl_Box *icon_;
  Fl_Button *button_[3];
  Fl_Input *input_;
  int retval_;
  int window_closed_;

  // static (private) variables

  static int enable_hotspot_;
  static int form_x_;
  static int form_y_;
  static int form_position_;

public:
  // Constructor
  Fl_Message(const char *iconlabel);
  ~Fl_Message() { delete window_; }

  int innards(const char *fmt, va_list ap, const char *b0, const char *b1, const char *b2);

  const char *input_innards(const char *fmt, va_list ap, const char *defstr, uchar type, int maxchar =
-1, bool str = false);
};

#endif // _src_Fl_Message_h_
```

14.238 Fl_Native_File_Chooser_Kdialog.H

```
//
// FLTK native file chooser widget : KDE version
//
// Copyright 2021-2022 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef FL_KDIALOG_NATIVE_FILE_CHOOSER_H
#define FL_KDIALOG_NATIVE_FILE_CHOOSER_H 1

#include <FL/Fl_Native_File_Chooser.H>
#include <string>

class Fl_Kdialog_Native_File_Chooser_Driver : public Fl_Native_File_Chooser_FLTK_Driver {
  friend class Fl_Native_File_Chooser;
  friend class Fl_Zenity_Native_File_Chooser_Driver;
  struct fnfc_pipe_struct {
    char *all_files;
    int fd;
  };
  static void fnfc_fd_cb(int fd, fnfc_pipe_struct *data);
  char **_pathnames;
  int _tpathnames;
  char *_directory;
  char *_preset_file;
  char *_title;
  static bool did_find_kdialog;
  static bool have_looked_for_kdialog;
  Fl_Kdialog_Native_File_Chooser_Driver(int val);
  ~Fl_Kdialog_Native_File_Chooser_Driver();
  int count() const FL_OVERRIDE;
  const char *filename() const FL_OVERRIDE;
  const char *filename(int i) const FL_OVERRIDE;
  virtual void build_command(std::string& command);
  int show() FL_OVERRIDE;
  char *parse_filter(const char *f);
```

```
  const char *filter() const FL_OVERRIDE;
  void filter(const char *f) FL_OVERRIDE;
  int filters() const FL_OVERRIDE;
  void preset_file(const char *val) FL_OVERRIDE;
  const char *preset_file() const FL_OVERRIDE;
  void directory(const char *val) FL_OVERRIDE;
  const char *directory() const FL_OVERRIDE;
  void title(const char *val) FL_OVERRIDE;
  const char *title() const FL_OVERRIDE;
  void shell_quote(std::string& s);
};

#endif // FL_KDIALOG_NATIVE_FILE_CHOOSER_H
```

14.239 Fl_Native_File_Chooser_Zenity.H

```
//
// FLTK native file chooser widget : Zenity version
//
// Copyright 2021-2022 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef FL_ZENITY_NATIVE_FILE_CHOOSER_H
#define FL_ZENITY_NATIVE_FILE_CHOOSER_H 1

#include "Fl_Native_File_Chooser_Kdialog.H"

class Fl_Zenity_Native_File_Chooser_Driver : public Fl_Kdialog_Native_File_Chooser_Driver {
  friend class Fl_Native_File_Chooser;
  static bool did_find_zenity;
  static bool have_looked_for_zenity;
  Fl_Zenity_Native_File_Chooser_Driver(int val);
  void append_filter(std::string& command);
  void build_command(std::string& command) FL_OVERRIDE;
};

#endif // FL_ZENITY_NATIVE_FILE_CHOOSER_H
```

14.240 fl_oxy.h

```
//
// "Oxy" Scheme drawing routines for the Fast Light Tool Kit (FLTK).
//
// Copyright 2011 by Dmitrij K. aka "kdiman"
// Copyright 2012-2022 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//   https://www.fltk.org/COPYING.php
//
// Please report all bugs and problems on the following page:
//
//   https://www.fltk.org/str.php
//

#ifndef fl_oxy_h
#define fl_oxy_h

#include <FL/Fl.H>

// draw an arrow GUI element for the 'oxy' scheme
//
// bb   bounding box
// t    arrow type
// o    orientation
// c    arrow color

extern FL_EXPORT void oxy_arrow(Fl_Rect bb,
                                Fl_Arrow_Type t, Fl_Orientation o,
                                Fl_Color col);
```

```
00033
00034 #endif // fl_oxy_h
```

14.241 Fl_Paged_Device.cxx File Reference

implementation of class Fl_Paged_Device.

```
#include <FL/Fl_Paged_Device.H>
#include <FL/Fl.H>
#include <FL/fl_draw.H>
```

14.241.1 Detailed Description

implementation of class Fl_Paged_Device.

14.242 fl_plastic.cxx File Reference

Implementation of the `'plastic'` scheme.

```
#include <FL/Fl.H>
#include <FL/Fl_Scheme.H>
#include <FL/fl_draw.H>
#include <cassert>
```

Functions

- const uchar * **fl_gray_ramp** ()
- void **fl_plastic_down_box** (int x, int y, int w, int h, Fl_Color c)
- void **fl_plastic_down_frame** (int x, int y, int w, int h, Fl_Color c)
- void **fl_plastic_down_round** (int x, int y, int w, int h, Fl_Color c)
- void **fl_plastic_thin_up_box** (int x, int y, int w, int h, Fl_Color c)
- void **fl_plastic_up_box** (int x, int y, int w, int h, Fl_Color c)
- void **fl_plastic_up_frame** (int x, int y, int w, int h, Fl_Color c)
- void **fl_plastic_up_round** (int x, int y, int w, int h, Fl_Color c)
- Fl_Color **shade_color** (uchar gc, Fl_Color bc)

14.242.1 Detailed Description

Implementation of the `'plastic'` scheme.

14.243 Fl_Private.H File Reference

Fl::Private namespace.

```
#include <FL/fl_config.h>
#include <FL/Fl.H>
```

Namespaces

- namespace Fl

  Fl is the FLTK global namespace containing state information and global methods for the current application.

Functions

- cairo_t * Fl::Private::cairo_make_current (void *gc, int W, int H)

  Creates a Cairo context from a `gc` and the given size.
- void **Fl::Private::run_checks** ()
- void **Fl::Private::run_idle** ()
- void Fl::Private::set_idle_ (Fl_Old_Idle_Handler cb)

  Sets an idle callback (internal use only).

Variables

- int **Fl::Private::box_border_radius_max_** = 15
- int **Fl::Private::box_shadow_width_** = 3
- Fl_Cairo_State **Fl::Private::cairo_state_**

  current Cairo context information
- int **Fl::Private::draw_GL_text_with_textures_** = 1
- int **Fl::Private::e_x_down**
- int **Fl::Private::e_y_down**
- void(* Fl::Private::idle_)()

  The currently executing idle callback function: DO NOT USE THIS DIRECTLY!
- unsigned char **Fl::Private::options_** [OPTION_LAST]
- unsigned char **Fl::Private::options_read_**
- int **Fl::Private::program_should_quit_** = 0
- int **Fl::Private::selection_to_clipboard_**
- int **Fl::Private::use_high_res_GL_** = 1

14.243.1 Detailed Description

Fl::Private namespace.

14.243.2 Function Documentation

14.243.2.1 cairo_make_current()

```
cairo_t * Fl::Private::cairo_make_current (
            void * gc,
            int W,
            int H )  [extern]
```

Creates a Cairo context from a `gc` and the given size.

Note

Only available if CMake FLTK_OPTION_CAIRO_WINDOW is enabled.

14.243.2.2 set_idle_()

```
void Fl::Private::set_idle_ (
            Fl_Old_Idle_Handler cb )  [inline]
```

Sets an idle callback (internal use only).
This method is now private and is used to store the idle callback. The old, public set_idle() method was deprecated since 1.3.x and is no longer available in FLTK 1.5.
See documentation: use Fl::add_idle() instead.

14.243.3 Variable Documentation

14.243.3.1 idle_

```
void(* Fl::Private::idle_)() ( )  [extern]
```

The currently executing idle callback function: DO NOT USE THIS DIRECTLY!
This is now used as part of a higher level system allowing multiple idle callback functions to be called.

See also

add_idle(), remove_idle()

14.244 Fl_Private.H

Go to the documentation of this file.

```
00001 //
00002 // Private header file for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Copyright 2025-2026 by Bill Spitzak and others.
00005 //
00006 // This library is free software. Distribution and use rights are outlined in
00007 // the file "COPYING" which should have been included with this file.  If this
00008 // file is missing or damaged, see the license at:
00009 //
00010 //     https://www.fltk.org/COPYING.php
00011 //
00012 // Please see the following page on how to report bugs and issues:
00013 //
00014 //     https://www.fltk.org/bugs.php
00015 //
00016
00021 #ifndef Fl_Private_H
00022 #define Fl_Private_H
00023
00024 #include <FL/fl_config.h> // build configuration
00025 #include <FL/Fl.H>
00026
00030 namespace Fl {
00031 namespace Private {
00032
00033 FL_EXPORT extern int use_high_res_GL_;
00034 FL_EXPORT extern int draw_GL_text_with_textures_;
00035 FL_EXPORT extern int box_shadow_width_;
00036 FL_EXPORT extern int box_border_radius_max_;
00037 FL_EXPORT extern int selection_to_clipboard_;
00038
00039 FL_EXPORT extern unsigned char options_[OPTION_LAST];
00040 FL_EXPORT extern unsigned char options_read_;
00041 FL_EXPORT extern int program_should_quit_; // non-zero means the program was asked to cleanly
      terminate
00042
00050 FL_EXPORT extern void (*idle_)();
00051
00052 FL_EXPORT extern void run_idle();
00053 FL_EXPORT extern void run_checks();
00054
00064 FL_EXPORT inline void set_idle_(Fl_Old_Idle_Handler cb) { idle_ = cb; }
00065
00066 #ifdef FLTK_HAVE_CAIRO
00067
00068 #if 0 // this non-public function appears not to be used anywhere in FLTK
00069   FL_EXPORT extern cairo_t *cairo_make_current(void *gc);
00070 #endif
00071 FL_EXPORT extern cairo_t *cairo_make_current(void *gc, int W, int H);
00072 FL_EXPORT extern Fl_Cairo_State cairo_state_;
00073
00074 #endif // FLTK_HAVE_CAIRO
00075
00076 // Global pen position at pen down event
00077 extern int e_x_down;
00078 extern int e_y_down;
00079
00080 } // namespace Private
00081 } // namespace Fl
00082
00083 #endif // !Fl_Private_H
```

14.245 fl_rect.cxx File Reference

Drawing and clipping routines for rectangles.

```
#include <FL/platform.H>
#include <FL/Fl_Graphics_Driver.H>
```

14.245.1 Detailed Description

Drawing and clipping routines for rectangles.

14.246 Fl_Scalable_Graphics_Driver.H

```
00001 //
00002 // Declaration of Fl_Scalable_Graphics_Driver, Fl_Font_Descriptor, Fl_Fontdesc
00003 // for the Fast Light Tool Kit (FLTK).
00004 //
00005 // Copyright 2025 by Bill Spitzak and others.
00006 //
00007 // This library is free software. Distribution and use rights are outlined in
00008 // the file "COPYING" which should have been included with this file.  If this
00009 // file is missing or damaged, see the license at:
00010 //
00011 //     https://www.fltk.org/COPYING.php
00012 //
00013 // Please see the following page on how to report bugs and issues:
00014 //
00015 //     https://www.fltk.org/bugs.php
00016 //
00017
00018 #ifndef FL_SCALABLE_GRAPHICS_DRIVER_H
00019 #define FL_SCALABLE_GRAPHICS_DRIVER_H
00020
00021 #include <FL/Fl_Graphics_Driver.H>
00022
00023 #ifndef FL_DOXYGEN
00024
00025
00026 /* Class Fl_Scalable_Graphics_Driver is platform-independent.
00027  It supports the scaling of all graphics coordinates by a
00028  float factor helpful to support HiDPI displays.
00029  This class does :
00030  - compute scaled coordinates
00031  - scale the cached offscreen of image objects
00032  - scale the pixel arrays used when performing direct image draws
00033  - call the member functions of a platform-specific,
00034  Fl_Scalable_Graphics_Driver-derived class that do the drawings with adequately
00035  scaled coordinates. These member functions are named with the _unscaled suffix.
00036  - scale and unscale the clipping region.
00037
00038  This class is presently used by the X11 and Windows platforms.
00039  */
00040 class FL_EXPORT Fl_Scalable_Graphics_Driver : public Fl_Graphics_Driver {
00041   Fl_Fontsize fontsize_; // scale-independent font size value
00042 public:
00043   static int floor(int x, float s);
00044   inline int floor(int x) { return Fl_Scalable_Graphics_Driver::floor(x, scale()); }
00045 protected:
00046   int line_style_;
00047   int line_width_;
00048   bool is_solid_;
00049   Fl_Scalable_Graphics_Driver();
00050   virtual Fl_Region scale_clip(float f);
00051   void unscale_clip(Fl_Region r);
00052   void point(int x, int y) FL_OVERRIDE;
00053   virtual void point_unscaled(float x, float y);
00054   void rect(int x, int y, int w, int h) FL_OVERRIDE;
00055   void rectf(int x, int y, int w, int h) FL_OVERRIDE;
00056   virtual void rect_unscaled(int x, int y, int w, int h);
00057   virtual void rectf_unscaled(int x, int y, int w, int h);
00058   void line(int x, int y, int x1, int y1) FL_OVERRIDE;
00059   virtual void line_unscaled(int x, int y, int x1, int y1);
00060   void line(int x, int y, int x1, int y1, int x2, int y2) FL_OVERRIDE;
00061   virtual void line_unscaled(int x, int y, int x1, int y1, int x2, int y2);
00062   void xyline(int x, int y, int x1) FL_OVERRIDE;
00063   virtual void xyline_unscaled(int x, int y, int x1);
00064   void xyline(int x, int y, int x1, int y2) FL_OVERRIDE {Fl_Graphics_Driver::xyline(x, y, x1, y2);}
00065   void xyline(int x, int y, int x1, int y2, int x3) FL_OVERRIDE {Fl_Graphics_Driver::xyline(x, y, x1,
      y2, x3);}
00066   void yxline(int x, int y, int y1) FL_OVERRIDE;
```

```
  virtual void yxline_unscaled(int x, int y, int y1);
  void yxline(int x, int y, int y1, int x2) FL_OVERRIDE {Fl_Graphics_Driver::yxline(x, y, y1, x2);}
  void yxline(int x, int y, int y1, int x2, int y3) FL_OVERRIDE {Fl_Graphics_Driver::yxline(x, y, y1, x2, y3);}
  void loop(int x0, int y0, int x1, int y1, int x2, int y2) FL_OVERRIDE;
  virtual void loop_unscaled(int x0, int y0, int x1, int y1, int x2, int y2);
  void loop(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3) FL_OVERRIDE;
  virtual void loop_unscaled(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3);
  void polygon(int x0, int y0, int x1, int y1, int x2, int y2) FL_OVERRIDE;
  virtual void polygon_unscaled(int x0, int y0, int x1, int y1, int x2, int y2);
  void polygon(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3) FL_OVERRIDE;
  virtual void polygon_unscaled(int x0, int y0, int x1, int y1, int x2, int y2, int x3, int y3);
  void circle(double x, double y, double r) FL_OVERRIDE;
  virtual void ellipse_unscaled(double xt, double yt, double rx, double ry);
  void font(Fl_Font face, Fl_Fontsize size) FL_OVERRIDE;
  Fl_Font font() FL_OVERRIDE;
  virtual void font_unscaled(Fl_Font face, Fl_Fontsize size);
  double width(const char *str, int n) FL_OVERRIDE;
  double width(unsigned int c) FL_OVERRIDE;
  virtual double width_unscaled(const char *str, int n);
  virtual double width_unscaled(unsigned int c);
  Fl_Fontsize size() FL_OVERRIDE;
  virtual Fl_Fontsize size_unscaled();
  void text_extents(const char *str, int n, int &dx, int &dy, int &w, int &h) FL_OVERRIDE;
  virtual void text_extents_unscaled(const char *str, int n, int &dx, int &dy, int &w, int &h);
  int height() FL_OVERRIDE;
  int descent() FL_OVERRIDE;
  virtual int height_unscaled();
  virtual int descent_unscaled();
  void draw(const char *str, int n, int x, int y) FL_OVERRIDE;
  virtual void draw_unscaled(const char *str, int n, int x, int y);
  void draw(int angle, const char *str, int n, int x, int y) FL_OVERRIDE;
  virtual void draw_unscaled(int angle, const char *str, int n, int x, int y);
  void draw(const char *str, int nChars, float x, float y) FL_OVERRIDE;
  void rtl_draw(const char* str, int n, int x, int y) FL_OVERRIDE;
  virtual void rtl_draw_unscaled(const char* str, int n, int x, int y);
  void arc(double x, double y, double r, double start, double end) FL_OVERRIDE;
  void arc(int x, int y, int w, int h, double a1, double a2) FL_OVERRIDE;
  virtual void arc_unscaled(int x, int y, int w, int h, double a1, double a2);
  void pie(int x, int y, int w, int h, double a1, double a2) FL_OVERRIDE;
  virtual void pie_unscaled(int x, int y, int w, int h, double a1, double a2);
  void draw_circle(int x, int y, int d, Fl_Color c) FL_OVERRIDE;
  void line_style(int style, int width=0, char* dashes=0) FL_OVERRIDE;
  virtual void line_style_unscaled(int style, int width, char* dashes);
  void draw_image_rescale(void *buf, Fl_Draw_Image_Cb cb, int X, int Y, int W, int H, int D, int L, bool mono);
  virtual void draw_image_unscaled(const uchar* buf, int X,int Y,int W,int H, int D=3, int L=0);
  virtual void draw_image_unscaled(Fl_Draw_Image_Cb cb, void* data, int X,int Y,int W,int H, int D=3);
  void draw_image(const uchar* buf, int X,int Y,int W,int H, int D=3, int L=0) FL_OVERRIDE;
  void draw_image(Fl_Draw_Image_Cb cb, void* data, int X,int Y,int W,int H, int D=3) FL_OVERRIDE;
  virtual void draw_image_mono_unscaled(const uchar* buf, int x, int y, int w, int h, int d, int l);
  void draw_image_mono(const uchar* buf, int X,int Y,int W,int H, int D=1, int L=0) FL_OVERRIDE;
  virtual void draw_image_mono_unscaled(Fl_Draw_Image_Cb cb, void* data, int X,int Y,int W,int H, int D=1);
  void draw_image_mono(Fl_Draw_Image_Cb cb, void* data, int X,int Y,int W,int H, int D=1) FL_OVERRIDE;

  void transformed_vertex(double xf, double yf) FL_OVERRIDE;
  void vertex(double x, double y) FL_OVERRIDE;
  float override_scale() FL_OVERRIDE;
  void restore_scale(float) FL_OVERRIDE;
  virtual void *change_pen_width(int lwidth);
  virtual void reset_pen_width(void *data);
};


/*
 Platforms usually define a derived class called Fl_XXX_Font_Descriptor
 containing extra platform-specific data/functions.
 This is a class for an actual system font, with junk to
 help choose it and info on character sizes.  Each Fl_Fontdesc has a
 linked list of these.  These are created the first time each system
 font/size combination is used.
 */
class Fl_Font_Descriptor {
public:
  Fl_Font_Descriptor *next;
  Fl_Fontsize size;
  Fl_Font_Descriptor(const char* fontname, Fl_Fontsize size);
  virtual FL_EXPORT ~Fl_Font_Descriptor() {}
  int ascent, descent;
  unsigned int listbase;// base of display list, 0 = none
};


struct Fl_Fontdesc {
  const char *name;
  char fontname[128];  // "Pretty" font name
```

```
  Fl_Font_Descriptor *first;  // linked list of sizes of this style
};

#endif // FL_DOXYGEN

#endif // FL_SCALABLE_GRAPHICS_DRIVER_H
```

14.247 Fl_Scheme.cxx File Reference

Implementation of the Fl_Scheme class.

```
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Scheme.H>
#include <FL/fl_string_functions.h>
#include <stdlib.h>
```

14.247.1 Detailed Description

Implementation of the Fl_Scheme class.
This class is intentionally not fully documented and is subject to change in future FLTK versions.

Note

Do not rely on details of this class.

Since

1.4.0

14.248 Fl_Screen_Driver.H

```
//
// All screen related calls in a driver style class.
//
// Copyright 1998-2026 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef FL_SCREEN_DRIVER_H
#define FL_SCREEN_DRIVER_H

#include <FL/fl_types.h>
#include <FL/Fl.H> // for Fl_Timeout_Handler
#include <FL/Fl_Text_Editor.H>

namespace Fl {
// Initialize screen driver and details of available screens.
extern void init_screens();     // recompute screen number and dimensions
} // namespace Fl

// TODO: add text composition?
// TODO: add Fl::display
// TODO: add copy/paste, drag/drop?
// TODO: get key/get mouse?
// TODO: system colors/colormaps
// TODO: system menu?
// TODO: native filechooser
// TODO: native message boxes
// TODO: read screen to image
// TODO: application shortcuts

class Fl_Window;
class Fl_RGB_Image;
class Fl_Group;
```

```
class Fl_Input;
class Fl_System_Driver;

class Fl_Screen_Driver {

protected:
  Fl_Screen_Driver();
  virtual ~Fl_Screen_Driver();

  static const int MAX_SCREENS = 16;

  int num_screens;
  static float fl_intersection(int x1, int y1, int w1, int h1,
                               int x2, int y2, int w2, int h2);

public:
  static int keyboard_screen_scaling; // true means ctrl/+/-/0/ resize windows
  static char bg_set;
  static char bg2_set;
  static char fg_set;
  static Fl_System_Driver *system_driver;
  // These flags are useful after calling XParseGeometry(). They indicate which of its
  // arguments contain meaningful data upon return.
  static const int fl_NoValue;
  static const int fl_WidthValue;
  static const int fl_HeightValue;
  static const int fl_XValue;
  static const int fl_YValue;
  static const int fl_XNegative;
  static const int fl_YNegative;
  // Next 2 are used when transient scale windows are implemented as popups
  static Fl_Window *transient_scale_parent;
  static void del_transient_window(void *);
  // key_table and key_table_size are used in fl_shortcut to translate key names
  struct Keyname {
    unsigned int key;
    const char* name;
  } *key_table;
  int key_table_size;

  virtual float scale(int) { return 1; }
  virtual void scale(int /*n*/, float /*f*/) {}
  static Fl_Screen_Driver *newScreenDriver();
  // implement to process the -display argument and support the DISPLAY env var
  virtual void display(const char *) { }
  // default implementation should be enough
  virtual int XParseGeometry(const char* string, int* x, int* y, unsigned int* width, unsigned int*
height);
  // the default implementation is most probably enough
  virtual void own_colormap() {}
  // the default implementation of shortcut_add_key_name() is in src/fl_shortcut.cxx
  virtual const char *shortcut_add_key_name(unsigned key, char *p, char *buf, const char **);
  // whether a platform uses additional code in Fl_Menu::handle_part1(int e)
  virtual int need_menu_handle_part1_extra() {return 0;}
  // whether a platform uses additional code in Fl_Menu::handle(int e)
  virtual int need_menu_handle_part2() {return 0;}
  // implement functions telling whether a key is pressed
  virtual int event_key(int) {return 0;}
  virtual int get_key(int) {return 0;}
  virtual int visual(int flags);
  // --- screen configuration
  virtual void init() {}
  virtual int x() { return 0; }
  virtual int y() { return 0; }
  virtual int w() { return 800; } // default, FL_OVERRIDE in driver!
  virtual int h() { return 600; } // default, FL_OVERRIDE in driver!
  virtual int screen_count();
  void screen_xywh(int &X, int &Y, int &W, int &H, int mx, int my);
  virtual void screen_xywh(int &X, int &Y, int &W, int &H, int /*n*/) {
    X = 0;
    Y = 0;
    W = 800;
    H = 600;
  }
  void screen_xywh(int &X, int &Y, int &W, int &H, int mx, int my, int mw, int mh);
  virtual bool screen_boundaries_known() { return true; }
  virtual int screen_num(int x, int y);
  virtual int screen_num(int x, int y, int w, int h);
  virtual void screen_dpi(float &h, float &v, int n = 0) { // FL_OVERRIDE in driver!
    h = 72;
    v = 72;
    (void)n;
  }
  void screen_work_area(int &X, int &Y, int &W, int &H, int mx, int my);
  virtual void screen_work_area(int &X, int &Y, int &W, int &H, int n) {
    screen_xywh(X, Y, W, H, n);
  }
```

```
// --- audible output
virtual void beep(int) {}
// --- global events
virtual void flush() {}                                    // must FL_OVERRIDE
virtual void grab(Fl_Window *) {}
// --- global colors
/* the default implementation of parse_color() may be enough */
virtual int parse_color(const char *p, uchar &r, uchar &g, uchar &b);
virtual void get_system_colors();
/* the default implementation of get_system_scheme() may be enough */
virtual const char *get_system_scheme();

static int secret_input_character;
/* Implement to indicate whether complex text input may involve marked text.
  When it does, has_marked_text returns non zero.
*/
virtual int has_marked_text() const { return 0; }
// implement so text-editing widgets support dead keys
virtual int compose(int &del) {
  del = 0;
  return 0;
}
// default implementation may be enough
virtual void compose_reset();
// implement to support drag-n-drop. use_selection = 1 means the GUI is welcome to display
// the selected text during the D&D operation
virtual int dnd(int use_selection = 0) { (void)use_selection; return 0; }
// null means no platform-specific key bindings for Fl_Text_Editor
Fl_Text_Editor::Key_Binding *text_editor_extra_key_bindings;
// default implementation may be enough
virtual int text_display_can_leak() const { return 0; }

// if no keyboard is connected on a touch or pen device, the system on-screen keyboard is
// requested
virtual void request_keyboard() {}
// we no longer need the on-screen keyboard; it's up to the system to hide it
virtual void release_keyboard() {}

/* Member function read_win_rectangle() supports public functions
  fl_read_image() and fl_capture_window() which capture pixel data from
  a window (or also from an offscreen buffer with fl_read_image).

  If 'may_capture_subwins' is true, an implementation may or may not capture
  also the content of subwindows embedded in 'win'. If subwindows were captured,
  *'did_capture_subwins' is returned set to true. If read_win_rectangle()
  is called with 'may_capture_subwins' set to true, 'did_capture_subwins' should
  be set before the call to the address of a boolean set to false.
  The implementation of this virtual function for the macOS platform has the
  capability of capturing subwindows when asked for.

  A platform may also use its read_win_rectangle() implementation to capture
  window decorations (e.g., title bar). In that case, it is called by
  Fl_XXX_Window_Driver::capture_titlebar_and_borders().

  win is the window to capture from, or NULL to capture from the current offscreen
*/
virtual Fl_RGB_Image *read_win_rectangle(int /*X*/, int /*Y*/, int /*w*/, int /*h*/, Fl_Window *,
                                         bool may_capture_subwins = false,
                                         bool *did_capture_subwins = NULL) {
  (void)may_capture_subwins;
  (void)did_capture_subwins;
  return NULL;
}
static void write_image_inside(Fl_RGB_Image *to, Fl_RGB_Image *from, int to_x, int to_y);
static Fl_RGB_Image *traverse_to_gl_subwindows(Fl_Group *g, int x, int y, int w, int h,
                                               Fl_RGB_Image *full_img);
static size_t convert_crlf(char *s, size_t len);
// optional platform-specific key handling for Fl_Input widget
// the default implementation may be enough
virtual int input_widget_handle_key(int key, unsigned mods, unsigned shift, Fl_Input *input);
// implement to support Fl::get_mouse()
virtual int get_mouse(int &/*x*/, int &/*y*/) { return 0; }
// optional methods to enable/disable input methods for complex scripts
virtual void enable_im() {}
virtual void disable_im() {}
// calls open_display_platform() and then does platform-independent work
void open_display();
// implement to open access to the display
virtual void open_display_platform() {}
// optional method to close display access
virtual void close_display() {}
// compute dimensions of an Fl_Offscreen
virtual void offscreen_size(Fl_Offscreen, int &/*width*/, int &/*height*/) {}

void rescale_all_windows_from_screen(int screen, float f, float old_f);
static void transient_scale_display(float f, int nscreen);
// need export to fltk_gl.so because used in glut_compatibility.cxx
```

```
  static FL_EXPORT int scale_handler(int event);
  virtual void desktop_scale_factor() {}
  void use_startup_scale_factor();
  enum APP_SCALING_CAPABILITY {
    NO_APP_SCALING = 0,
    SYSTEMWIDE_APP_SCALING,
    PER_SCREEN_APP_SCALING
  };
  virtual APP_SCALING_CAPABILITY rescalable() { return NO_APP_SCALING; }
  // supports Fl_Window::default_icons()
  virtual void default_icons(const Fl_RGB_Image *icons[], int count);
  // implement to support copy-to-clipboard
  virtual void copy(const char * /*stuff*/, int /*len*/, int /*clipboard*/, const char * /*type*/) {}
  // implement to support paste-from-clipboard
  virtual void paste(Fl_Widget &, int /*clipboard*/, const char * /*type*/) {}
  // implement to support paste-from-clipboard
  virtual int clipboard_contains(const char * /*type*/) {return 0;}
  //  implement to support paste-from-clipboard
  virtual void clipboard_notify_change() {}
  // next 3 are related to Input Methods
  virtual void set_spot(int font, int size, int X, int Y, int W, int H, Fl_Window *win);
  virtual void reset_spot();
  virtual void set_status(int X, int Y, int W, int H);
  virtual float base_scale(int numscreen);
};

#endif // !FL_SCREEN_DRIVER_H

```

14.249 Fl_Sys_Menu_Bar_Driver.H

```
//
// system menu bar widget for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2017 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef Fl_Sys_Menu_Bar_Driver_H
#define Fl_Sys_Menu_Bar_Driver_H

#if !defined(FL_DOXYGEN)

#include <FL/Fl_Sys_Menu_Bar.H>

class Fl_Sys_Menu_Bar_Driver {
  friend class Fl_Sys_Menu_Bar;
public:
  static Fl_Sys_Menu_Bar::window_menu_style_enum window_menu_style_;
  static Fl_Sys_Menu_Bar_Driver *driver_; // to be assigned with a unique object of this class or of a derived class
  Fl_Sys_Menu_Bar *bar;
  Fl_Sys_Menu_Bar_Driver();
  virtual ~Fl_Sys_Menu_Bar_Driver();
  virtual void update() {}
  virtual void draw() { bar->Fl_Menu_Bar::draw();}
  virtual void about(Fl_Callback *, void *) {}
  virtual int add(const char* label, int shortcut, Fl_Callback *cb, void *user_data, int flags) {
    return bar->Fl_Menu_Bar::add(label, shortcut, cb, user_data, flags);
  }
  virtual int add(const char* str) { return bar->Fl_Menu_Bar::add(str); }
  virtual int insert(int index, const char* label, int shortcut, Fl_Callback *cb, void *user_data, int flags) {
    return bar->Fl_Menu_Bar::insert(index, label, shortcut, cb, user_data, flags);
  }
  virtual void menu(const Fl_Menu_Item *m) { bar->Fl_Menu_Bar::menu(m); }
  virtual void shortcut(int i, int s) { bar->Fl_Menu_Bar::shortcut(i, s); }
  virtual void setonly(Fl_Menu_Item *item) { bar->Fl_Menu_Bar::setonly(item); }
  virtual void clear() { bar->Fl_Menu_Bar::clear(); }
  virtual int clear_submenu(int index) { return bar->Fl_Menu_Bar::clear_submenu(index);}
  virtual void remove(int index) { bar->Fl_Menu_Bar::remove(index); }
  virtual void replace(int index, const char *name) { bar->Fl_Menu_Bar::replace(index, name); }
  virtual void mode(int i, int fl) { bar->Fl_Menu_Bar::mode(i, fl); }
  virtual void create_window_menu() {}
  virtual void play_menu(const Fl_Menu_Item *) {}
  static Fl_Sys_Menu_Bar::window_menu_style_enum window_menu_style() { return window_menu_style_; }
```

```
  static void window_menu_style(Fl_Sys_Menu_Bar::window_menu_style_enum style) { window_menu_style_ = style; }
};

#endif // !defined(FL_DOXYGEN)

#endif // Fl_Sys_Menu_Bar_Driver_H
```

14.250 Fl_System_Driver.H

```
//
// A base class for platform specific system calls
// for the Fast Light Tool Kit (FLTK).
//
// Copyright 2010-2026 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

/* Class hierarchy
 + Fl_System_Driver
 | + Fl_Posix_System_Driver
 | | + Fl_Unix_System_Driver
 | | + Fl_Darwin_System_Driver
 | + Fl_WinAPI_System_Driver
 */

#ifndef FL_SYSTEM_DRIVER_H
#define FL_SYSTEM_DRIVER_H

#include <FL/Fl.H>
#include <FL/Fl_Export.H>
#include <FL/filename.H>
#include <FL/Fl_Preferences.H>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <time.h>

class Fl_File_Icon;
class Fl_File_Browser;
class Fl_Pixmap;
class Fl_Widget;
class Fl_Sys_Menu_Bar_Driver;

class Fl_System_Driver {
  friend Fl_System_Driver *Fl::system_driver();
protected:
  // implement once for each platform
  static Fl_System_Driver *newSystemDriver();
  Fl_System_Driver();
  int filename_relative_(char *to, int tolen, const char *from, const char *base, bool case_sensitive);

  // -- Awake handler stuff --
public:
  static Fl_Awake_Handler *awake_ring_;
  static void **awake_data_;
  static int awake_ring_size_;
  static int awake_ring_head_;
  static int awake_ring_tail_;
  static int push_awake_handler(Fl_Awake_Handler, void*, bool once);
  static int pop_awake_handler(Fl_Awake_Handler&, void*&);
  static bool awake_ring_empty();

public:
  virtual ~Fl_System_Driver();

  // To save initialisation time, don't make these virtual.
  static const char *shift_name() { return fl_local_shift; }
  static const char *meta_name() { return fl_local_meta; }
  static const char *alt_name() { return fl_local_alt; }
  static const char *control_name() { return fl_local_ctrl; }

  // implement if the system adds unwanted program argument(s)
  virtual int single_arg(const char *) { return 0; }
```

```
  // implement if the system adds unwanted program argument pair(s)
  virtual int arg_and_value(const char * /*name*/, const char * /*value*/) { return 0; }
  static void warning(const char* format, ...);
  // implement to set the default effect of Fl::warning()
  virtual void warning(const char* format, va_list args);
  static void error(const char* format, ...);
  // implement to set the default effect of Fl::error()
  virtual void error(const char* format, va_list args);
  static void fatal(const char* format, ...);
  // implement to set the default effect of Fl::error()
  virtual void fatal(const char* format, va_list args);

  // implement these to support cross-platform file operations
  virtual char *utf2mbcs(const char *s) {return (char*)s;}
  virtual char *getenv(const char*) {return NULL;}
  virtual int putenv(const char *) {return -1;}
  virtual int open(const char* /*f*/, int /*oflags*/, int /*pmode*/) {return -1;}

  // implement these to support cross-platform string operations
  // Note: the default implementation ignores the 'binary' argument.
  // Some platforms (notably Windows) may use this argument.
  virtual int open_ext(const char* f, int /*binary*/, int oflags, int pmode) {
    return this->open(f, oflags, pmode);
  }
  virtual FILE *fopen(const char* f, const char *mode);
  virtual int system(const char*) {return -1;}
  virtual int execvp(const char * /*file*/, char *const * /*argv*/) {return -1;}
  virtual int chmod(const char* /*f*/, int /*mode*/) {return -1;}
  virtual int access(const char* /*f*/, int /*mode*/) { return -1;}
  virtual int flstat(const char* /*f*/, struct stat *) { return -1;}
  virtual char *getcwd(char* /*b*/, int /*l*/) {return NULL;}
  virtual int chdir(const char*) {return -1;}
  virtual int unlink(const char*) {return -1;}
  virtual int mkdir(const char* /*f*/, int /*mode*/) {return -1;}
  virtual int rmdir(const char*) {return -1;}
  virtual int rename(const char* /*f*/, const char * /*n*/) {return -1;}

  // Windows commandline argument conversion to UTF-8.
  // Default implementation: no-op, overridden only on Windows
  virtual int args_to_utf8(int argc, char ** &argv) { return argc; }

  // the default implementation of these utf8... functions should be enough
  virtual unsigned utf8towc(const char* src, unsigned srclen, wchar_t* dst, unsigned dstlen);
  virtual unsigned utf8fromwc(char* dst, unsigned dstlen, const wchar_t* src, unsigned srclen);
  virtual int utf8locale() {return 1;}
  virtual unsigned utf8to_mb(const char* src, unsigned srclen, char* dst, unsigned dstlen);
  virtual unsigned utf8from_mb(char* dst, unsigned dstlen, const char* src, unsigned srclen);
  // implement to shield fprintf() from locale changes in decimal point
  virtual int clocale_vprintf(FILE *output, const char *format, va_list args);
  virtual int clocale_vsnprintf(char *output, size_t output_size, const char *format, va_list args);
  virtual int clocale_vsscanf(const char *input, const char *format, va_list args);
  // implement scandir-like function
  virtual int filename_list(const char * /*d*/, dirent ***,
                            int (* /*sort*/)(struct dirent **, struct dirent **),
                            char *errmsg=NULL, int errmsg_sz=0) {
    (void)errmsg; (void)errmsg_sz;
    return -1;
  }
  // the default implementation of filename_expand() may be enough
  virtual int filename_expand(char *to, int tolen, const char *from);
  // to implement
  virtual const char *getpwnam(const char *) {return NULL;}
  // the default implementation of filename_relative() is in src/filename_absolute.cxx and may be enough
  virtual int filename_relative(char *to, int tolen, const char *from, const char *base);
  // the default implementation of filename_absolute() is in src/filename_absolute.cxx and may be enough
  virtual int filename_absolute(char *to, int tolen, const char *from, const char *base);
  // the default implementation of filename_isdir() is in src/filename_isdir.cxx and may be enough
  virtual int filename_isdir(const char* n);
  // the default implementation of filename_isdir_quick() is in src/filename_isdir.cxx and may be enough
  virtual int filename_isdir_quick(const char* n);
  // the default implementation of filename_ext() is in src/filename_ext.cxx and may be enough
  virtual const char *filename_ext(const char *buf);
  // implement to support fl_filename_name()
  virtual const char *filename_name(const char *buf) {return buf;}
  // implement to support fl_open_uri()
  virtual int open_uri(const char * /*uri*/, char * /*msg*/, int /*msglen*/) {return 0;}
  // the default implementation of use_recent_tooltip_fix() may be enough
  virtual int use_recent_tooltip_fix() {return 0;}
  // the default implementation of need_test_shortcut_extra() may be enough
  virtual int need_test_shortcut_extra() {return 0;}
  // implement to support Fl_File_Browser::load()
  virtual int file_browser_load_filesystem(Fl_File_Browser *, char * /*filename*/, int /*lname*/, Fl_File_Icon *) {return 0;}
  // the default implementation of file_browser_load_directory() should be enough
```

```
  virtual int file_browser_load_directory(const char *directory, char *filename, size_t name_size,
                                          dirent ***pfiles, Fl_File_Sort_F *sort,
                                          char *errmsg=NULL, int errmsg_sz=0);
  // implement to support Fl_Preferences
  virtual void newUUID(char *uuidBuffer) { uuidBuffer[0] = 0; }
  // implement to support Fl_Preferences
  virtual char *preference_rootnode(Fl_Preferences *, Fl_Preferences::Root,
                                    const char * /*vendor*/,
                                    const char * /*application*/) {return NULL;}
  // the default implementation of preferences_need_protection_check() may be enough
  virtual int preferences_need_protection_check() {return 0;}
  // implement to support Fl_Plugin_Manager::load()
  virtual void *load(const char *) {return NULL;}
  // the default implementation is most probably enough
  virtual void png_extra_rgba_processing(unsigned char * /*array*/, int /*w*/, int /*h*/) {}
  // the default implementation is most probably enough
  virtual const char *next_dir_sep(const char *start) { return strchr(start, '/');}
  // implement to support threading
  virtual void awake(void*) {}
  virtual int lock() {return 1;}
  virtual void unlock() {}
  virtual void* thread_message() {return NULL;}
  // implement to support Fl_File_Icon
  virtual int file_type(const char *filename);
  // implement to return the user's home directory name
  virtual const char *home_directory_name() { return ""; }
  // the default implementation is most probably enough
  virtual const char *filesystems_label() { return "File Systems"; }
  // return TRUE means \ same as / in file names
  virtual int backslash_as_slash() {return 0;}
  // return TRUE means : indicates a drive letter in file names
  virtual int colon_is_drive() {return 0;}
  // return TRUE means that files whose name begins with dot are hidden
  virtual int dot_file_hidden() {return 0;}
  // return TRUE when file names are case insensitive
  virtual int case_insensitive_filenames() {return 0;}
  // the implementations of local_to_latin1() and latin1_to_local() are in fl_encoding_latin1.cxx
  virtual const char *local_to_latin1(const char *t, int n);
  virtual const char *latin1_to_local(const char *t, int n);
  // the implementations of local_to_mac_roman() and mac_roman_to_local() are in
fl_encoding_mac_roman.cxx
  virtual const char *local_to_mac_roman(const char *t, int n);
  virtual const char *mac_roman_to_local(const char *t, int n);
  // draw default tree view expando button
  virtual void tree_draw_expando_button(int x, int y, bool state, bool active);
  // the default implementation of tree_connector_style() is in Fl_Tree_Prefs.cxx and can be enough
  virtual int tree_connector_style();
  virtual void add_fd(int fd, int when, Fl_FD_Handler cb, void* = 0);
  virtual void add_fd(int fd, Fl_FD_Handler cb, void* = 0);
  virtual void remove_fd(int, int when);
  virtual void remove_fd(int);
  // the default implementation of open_callback() may be enough
  virtual void open_callback(void (*)(const char *));
  // The default implementation may be enough.
  virtual void gettime(time_t *sec, int *usec);
  virtual Fl_Sys_Menu_Bar_Driver *sys_menu_bar_driver() { return NULL; }
  virtual void lock_ring() {}
  virtual void unlock_ring() {}
  virtual double wait(double);                          // must FL_OVERRIDE
  virtual int ready() { return 0; }                     // must FL_OVERRIDE
  virtual int close_fd(int) {return -1;} // to close a file descriptor
};

#endif // FL_SYSTEM_DRIVER_H
```

14.251 Fl_Timeout.cxx File Reference

```
#include <config.h>
#include "Fl_Timeout.h"
#include "Fl_System_Driver.H"
#include <stdio.h>
#include <math.h>
```

14.252 Fl_Timeout.h File Reference

Fl_Timeout handling.

```
#include <FL/Fl.H>
```

Classes

- class Fl_Timeout

  The internal class Fl_Timeout handles all timeout related functions.

Macros

- #define **FL_TIMEOUT_DEBUG** 0

14.252.1 Detailed Description

Fl_Timeout handling.
This file contains implementations of:

- Fl::add_timeout()
- Fl::repeat_timeout()
- Fl::has_timeout()
- Fl::remove_timeout()
- Fl::remove_next_timeout()

and related methods of class Fl_Timeout.

14.253 Fl_Timeout.h

Go to the documentation of this file.

```
00001 //
00002 // Header for timeout support functions for the Fast Light Tool Kit (FLTK).
00003 //
00004 // Author: Albrecht Schlosser
00005 // Copyright 2021-2024 by Bill Spitzak and others.
00006 //
00007 // This library is free software. Distribution and use rights are outlined in
00008 // the file "COPYING" which should have been included with this file.  If this
00009 // file is missing or damaged, see the license at:
00010 //
00011 //     https://www.fltk.org/COPYING.php
00012 //
00013 // Please see the following page on how to report bugs and issues:
00014 //
00015 //     https://www.fltk.org/bugs.php
00016 //
00017
00018 #ifndef _src_Fl_Timeout_h_
00019 #define _src_Fl_Timeout_h_
00020
00021 #include <FL/Fl.H>
00022
00023 #define FL_TIMEOUT_DEBUG 0        // 1 = include debugging features, 0 = no
00024
00056 class Fl_Timeout {
00057
00058 protected:
00059
00060   Fl_Timeout *next;             // ** Link to next timeout
00061   Fl_Timeout_Handler callback;  // the user's callback
00062   void *data;                   // the user's callback data
00063   double time;                  // delay until timeout
00064   int skip;                     // skip "new" (inserted) timers (issue #450)
00065
00066   // constructor
00067   Fl_Timeout() {
00068     next = 0;
00069     callback = 0;
00070     data = 0;
00071     time = 0;
00072     skip = 0;
```

```
  }

  // destructor
  ~Fl_Timeout() = default;

  // get a new timer entry from the pool or allocate a new one
  static Fl_Timeout *get(double time, Fl_Timeout_Handler cb, void *data);

  // insert this timer into the active timer queue, sorted by expiration time
  void insert();

  // remove this timer from the active timer queue and
  // add it to the "current" timer stack
  void make_current();

  // remove this timer from the current timer stack and
  // add it to the list of free timers
  void release();

  double delay() {
    return time;
  }

  void delay(double t) {
    time = t;
  }

public:
  // Returns whether the given timeout is active.
  static int has_timeout(Fl_Timeout_Handler cb, void *data);

  // Add or remove timeouts

  static void add_timeout(double time, Fl_Timeout_Handler cb, void *data);
  static void repeat_timeout(double time, Fl_Timeout_Handler cb, void *data);
  static void remove_timeout(Fl_Timeout_Handler cb, void *data);
  static int remove_next_timeout(Fl_Timeout_Handler cb, void *data = NULL, void **data_return = NULL);
  static std::vector<Fl::TimeoutData> timeout_list();

  // Elapse timeouts, i.e. calculate new delay time of all timers.
  // This does not call the timer callbacks.
  static void elapse_timeouts();

  // Elapse timeouts and call timer callbacks.
  static void do_timeouts();

  // Return the delay in seconds until the next timer expires.
  static double time_to_wait(double ttw);

#if FL_TIMEOUT_DEBUG
  // Write some statistics to stdout
  static void debug(int level = 1);
#endif

protected:

  static Fl_Timeout *current();

  static Fl_Timeout *first_timeout;

  static Fl_Timeout *free_timeout;

  static Fl_Timeout *current_timeout;   // list of "current" timeouts

}; // class Fl_Timeout

#endif // _src_Fl_Timeout_h_
```

14.254 fl_vertex.cxx File Reference

Portable drawing code for drawing arbitrary shapes with simple 2D transformations.

```
#include <FL/Fl_Graphics_Driver.H>
#include <FL/Fl.H>
#include <FL/math.h>
#include <stdlib.h>
```

14.254.1 Detailed Description

Portable drawing code for drawing arbitrary shapes with simple 2D transformations.

14.255 Fl_Window_Driver.H

```
00001 //
00002 // A base class for platform specific window handling code
00003 // for the Fast Light Tool Kit (FLTK).
00004 //
00005 // Copyright 2010-2026 by Bill Spitzak and others.
00006 //
00007 // This library is free software. Distribution and use rights are outlined in
00008 // the file "COPYING" which should have been included with this file.  If this
00009 // file is missing or damaged, see the license at:
00010 //
00011 //     https://www.fltk.org/COPYING.php
00012 //
00013 // Please see the following page on how to report bugs and issues:
00014 //
00015 //     https://www.fltk.org/bugs.php
00016 //
00017
00028 #ifndef FL_WINDOW_DRIVER_H
00029 #define FL_WINDOW_DRIVER_H
00030
00031 #include <FL/Fl_Export.H>
00032 #include <FL/Fl_Window.H>
00033 #include <FL/Fl_Overlay_Window.H>
00034
00035 #include <stdlib.h>
00036
00037 class Fl_X;
00038 class Fl_Image;
00039 class Fl_RGB_Image;
00040 class Fl_Image_Surface;
00041
00052 class Fl_Window_Driver
00053 {
00054   friend class Fl_Window;
00055 private:
00056   static bool is_a_rescale_; // true when a top-level window is being rescaled
00057
00058 protected:
00059   Fl_Window *pWindow;
00060   int screen_num_; // number of screen where window is mapped
00061 public:
00062   Fl_Window_Driver(Fl_Window *);
00063   virtual ~Fl_Window_Driver();
00064   static Fl_Window_Driver *newWindowDriver(Fl_Window *);
00065   static fl_uintptr_t xid(const Fl_Window *win);
00066   static Fl_Window *find(fl_uintptr_t xid);
00067   int wait_for_expose_value;
00068   Fl_Image_Surface *other_xid; // offscreen bitmap (overlay and double-buffered windows)
00069   int screen_num();
00070   void screen_num(int n) { screen_num_ = n; }
00071
00072
00073   // --- frequently used accessors to public window data
00075   int x() const { return pWindow->x(); }
00077   int y() const { return pWindow->y(); }
00079   int w() const { return pWindow->w(); }
00081   int h() const { return pWindow->h(); }
00083   int border() const { return pWindow->border(); }
00085   int visible() const { return pWindow->visible(); }
00087   int visible_r() const { return pWindow->visible_r(); }
00089   int shown() const { return pWindow->shown(); }
00091   Fl_Group *parent() const { return pWindow->parent(); }
00092
00093   // --- accessors to private window data
00094   int is_resizable() { return pWindow->is_resizable(); }
00095   void is_a_rescale(bool b) { is_a_rescale_ = b;}
00096   int fullscreen_screen_top();
00097   int fullscreen_screen_bottom();
00098   int fullscreen_screen_left();
00099   int fullscreen_screen_right();
00100   int* no_fullscreen_x() { return &pWindow->no_fullscreen_x; }
00101   int* no_fullscreen_y() { return &pWindow->no_fullscreen_y; }
00102   int* no_fullscreen_w() { return &pWindow->no_fullscreen_w; }
00103   int* no_fullscreen_h() { return &pWindow->no_fullscreen_h; }
00104   int force_position();
00105   void force_position(int c);
00106   void x(int X);
00107   void y(int Y);
00108   void current(Fl_Window *c);
00109   char show_iconic() { return Fl_Window::show_next_window_iconic(); }
00110   void show_iconic(char c) { Fl_Window::show_next_window_iconic(c); }
00111   void flx(Fl_X *x) { pWindow->flx_ = x; }
00112   Fl_Cursor cursor_default() { return pWindow->cursor_default; }
00113   void destroy_double_buffer();
00115   Fl_Window *overlay() {
```

```
    return pWindow->as_overlay_window() ? pWindow->as_overlay_window()->overlay_ : NULL;
  }
  void overlay(Fl_Window *o) {
    if (pWindow->as_overlay_window()) pWindow->as_overlay_window()->overlay_ = o;
  }

  void resize_after_scale_change(int ns, float old_f, float new_f);
  void set_popup_window() { pWindow->set_flag(Fl_Window::POPUP); }
  bool popup_window() const {return pWindow->flags() & Fl_Window::POPUP;}

  // --- window data
  virtual int decorated_w() { return w(); } // default, should be overidden by driver
  virtual int decorated_h() { return h(); }
  virtual const Fl_Image* shape() { return NULL; }

  // --- window management
  virtual void take_focus();
  virtual void flush(); // the default implementation may be enough
  virtual void flush_double();
  virtual void flush_overlay();
  virtual void draw_begin();
  virtual void draw_end();
  void draw();
  virtual void make_current();
  virtual void label(const char *name, const char *mininame);

  virtual void makeWindow() {}
  virtual void wait_for_expose();
  virtual void show();
  virtual void resize(int /*X*/, int /*Y*/, int /*W*/, int /*H*/) {}
  virtual void hide() {}
  int hide_common();
  virtual void map() {}
  virtual void unmap() {}
  virtual void fullscreen_on() {}
  virtual void fullscreen_off(int /*X*/, int /*Y*/, int /*W*/, int /*H*/) {}
  virtual void fullscreen_screens(bool /*on_off*/) {}
  virtual void maximize();
  virtual void un_maximize();
  virtual bool maximize_needs_hide() { return false; }
  void is_maximized(bool b) { pWindow->is_maximized_(b); }
  virtual void use_border();
  virtual void size_range();
  virtual void iconize() {}
  virtual void decoration_sizes(int *top, int *left,  int *right, int *bottom) {
    *top = *left = *right = *bottom = 0;
  }
  virtual void show_with_args_begin() {}
  virtual void show_with_args_end(int /*argc*/, char ** /*argv*/) {}
  virtual int can_do_overlay();
  virtual void redraw_overlay();

  // --- window cursor stuff
  virtual int set_cursor(Fl_Cursor);
  virtual int set_cursor(const Fl_RGB_Image*, int, int);

  // --- window shape stuff
  void shape_pixmap_(Fl_Image* pixmap); // platform-independent, support function
  virtual void shape(const Fl_Image*) {}
  virtual void shape_alpha_(Fl_Image*, int /*offset*/) {}

  // --- window icon stuff
  virtual void icons(const Fl_RGB_Image* /*icons*/[], int /*count*/) {}
  virtual const void *icon() const {return NULL;}
  virtual void icon(const void *) {}
  virtual void free_icons() {}

  // --- window printing/drawing helper
  virtual void capture_titlebar_and_borders(Fl_RGB_Image*& top, Fl_RGB_Image*& left,
                                            Fl_RGB_Image*& bottom, Fl_RGB_Image*& right);
  virtual int scroll(int /*src_x*/, int /*src_y*/, int /*src_w*/, int /*src_h*/,
                     int /*dest_x*/, int /*dest_y*/,
                     void (*)(void*, int,int,int,int), void*) { return 0; }
  static inline Fl_Window_Driver* driver(const Fl_Window *win) {return win->pWindowDriver;}

  // --- support for menu windows
  // The default implementation of next 2 virtual members is enough if the
  // position of a window in a screen is known. Next static members may be useful
  // when that's not the case, as with Wayland.
  virtual void reposition_menu_window(int x, int y);
  virtual void menu_window_area(int &X, int &Y, int &W, int &H, int nscreen = -1);
  static bool to_menutitle(Fl_Window*);
  static Fl_Window *menu_parent(int *display_height = NULL);
  static Fl_Window *menu_leftorigin(Fl_Window*);
  static Fl_Window *menu_title(Fl_Window*);
  static int menu_itemheight(Fl_Window*);
  static int menu_bartitle(Fl_Window*);
```

```
  static int menu_selected(Fl_Window*);
  static int *menu_offset_y(Fl_Window*);
  static bool is_floating_title(Fl_Window *);
  static void scroll_to_selected_item(Fl_Window *);
  // non-NULL when an Fl_Menu_Button is being pulled down
  static class Fl_Menu_Button *current_menu_button;

  virtual fl_uintptr_t os_id() { return 0; }
  virtual void allow_expand_outside_parent() {}
};

#endif // FL_WINDOW_DRIVER_H
```

14.256 fl_write_jpeg.cxx File Reference

JPEG image support functions.

```
#include <config.h>
#include <FL/Fl_JPEG_Image.H>
#include <FL/Fl_RGB_Image.H>
#include <FL/fl_utf8.h>
#include <stdio.h>
#include <stdlib.h>
```

Functions

- int fl_write_jpeg (const char *filename, const char *pixels, int w, int h, int d, int ld)

  Write raw image data to a JPEG image file.
- int fl_write_jpeg (const char *filename, const unsigned char *pixels, int w, int h, int d, int ld)

  Write raw image data to a JPEG image file.
- int fl_write_jpeg (const char *filename, Fl_RGB_Image *img)

  Write an RGB image to a JPEG image file.

14.256.1 Detailed Description

JPEG image support functions.

14.256.2 Function Documentation

14.256.2.1 fl_write_jpeg() [1/3]

```
int fl_write_jpeg (
            const char * filename,
            const char * pixels,
            int w,
            int h,
            int d,
            int ld )
```

Write raw image data to a JPEG image file.
For further restrictions and return values please see fl_write_jpeg(const char *filename, Fl_RGB_Image *img).

Parameters

in	*filename*	Output filename, extension should be '.jpg' or '.jpeg'
in	*pixels*	Image data
in	*w*	Image data width
in	*h*	Image data height
in	*d*	Image depth: 1 = GRAY, 2 = GRAY+alpha, 3 = RGB, 4 = RGBA
in	*ld*	Line delta: default (0) = w * d

Returns

success (0) or error code: negative values are errors

See also

fl_write_jpeg(const char *filename, Fl_RGB_Image *img)

14.256.2.2 fl_write_jpeg() [2/3]

```
int fl_write_jpeg (
            const char * filename,
            const unsigned char * pixels,
            int w,
            int h,
            int d,
            int ld )
```

Write raw image data to a JPEG image file.

See also

fl_write_jpeg(const char *filename, const char *pixels, int w, int h, int d, int ld)

14.256.2.3 fl_write_jpeg() [3/3]

```
int fl_write_jpeg (
            const char * filename,
            Fl_RGB_Image * img )
```

Write an RGB image to a JPEG image file.
Create a JPEG image file from an RGB image (Fl_RGB_Image).
The image file is always written with the original image size data_w() and data_h(), even if the image has been scaled.
Image depth 1 (gray), 2 (gray + alpha), 3 (RGB), and 4 (RGBA) are supported. For images with alpha channel (depth 2 or 4), the alpha component is ignored and only the color data is written since JPEG does not support transparency.

Note

Error handling is limited to basic error detection: library availability and file opening errors.

Parameters

in	*filename*	Output filename, extension should be '.jpg' or '.jpeg'
in	*img*	RGB image to be written

Returns

success (0) or error code: negative values are errors

Return values

0	success, file has been written
-1	jpeg library not available
-2	file open error
-3	invalid image depth (must be 1, 2, 3, or 4)
-4	memory allocation error

See also

fl_write_jpeg(const char ∗, const char ∗, int, int, int, int)

14.257 fl_write_png.cxx File Reference

PNG image support functions.

```
#include <config.h>
#include <FL/Fl_PNG_Image.H>
#include <FL/Fl_RGB_Image.H>
#include <FL/fl_string_functions.h>
#include <FL/fl_utf8.h>
#include <stdio.h>
#include <time.h>
```

Functions

- int fl_write_png (const char ∗filename, const char ∗pixels, int w, int h, int d, int ld)

  Write raw image data to a PNG image file.
- int fl_write_png (const char ∗filename, const unsigned char ∗pixels, int w, int h, int d, int ld)

  Write raw image data to a PNG image file.
- int fl_write_png (const char ∗filename, Fl_RGB_Image ∗img)

  Write an RGB(A) image to a PNG image file.

14.257.1 Detailed Description

PNG image support functions.

14.257.2 Function Documentation

14.257.2.1 fl_write_png() [1/3]

```
int fl_write_png (
            const char * filename,
            const char * pixels,
            int w,
            int h,
            int d,
            int ld )
```

Write raw image data to a PNG image file.
This is a very basic and restricted function to create a PNG image file from raw image data, e.g. a screenshot.
The image data must be aligned w/o gaps after each row (ld = 0 or ld = w ∗ d) or `ld` must be the total length of each row, i.e. w ∗ d + gapsize. If ld == 0 then ld = w ∗ d is assumed.
The total data size must be (w ∗ d + gapsize) ∗ h = ld' ∗ h where ld' = w ∗ d if ld == 0.
For further restrictions and return values please see fl_write_png(const char ∗filename, Fl_RGB_Image ∗img).

Parameters

in	*filename*	Output filename, extension should be '.png'
in	*pixels*	Image data
in	*w*	Image data width
in	*h*	Image data height
in	*d*	Image depth: 1 = GRAY, 2 = GRAY + alpha, 3 = RGB, 4 = RGBA
in	*ld*	Line delta: default (0) = w ∗ d

Returns

success (0) or error code, see ...

See also

fl_write_png(const char *filename, Fl_RGB_Image *img)

14.257.2.2 fl_write_png() [2/3]

```
int fl_write_png (
            const char * filename,
            const unsigned char * pixels,
            int w,
            int h,
            int d,
            int ld )
```

Write raw image data to a PNG image file.

See also

fl_write_png(const char *filename, const char *pixels, int w, int h, int d, int ld)

14.257.2.3 fl_write_png() [3/3]

```
int fl_write_png (
            const char * filename,
            Fl_RGB_Image * img )
```

Write an RGB(A) image to a PNG image file.
This is a very basic and restricted function to create a PNG image file from an RGB image (Fl_RGB_Image).
The image data must be aligned w/o gaps, i.e. ld() **MUST** be zero or equal to data_w() * data_h().
The image file is always written with the original image size data_w() and data_h(), even if the image has been scaled.
Image depth 1 (gray), 2 (gray + alpha channel), 3 (RGB) and 4 (RGBA) are supported.

Note

Currently there is no error handling except for errors when opening the file. This may be changed in the future.

Parameters

in	*filename*	Output filename, extension should be '.png'
in	*img*	RGB image to be written

Returns

success (0) or error code: negative values are errors

Return values

0	success, file has been written
-1	png or zlib library not available
-2	file open error

See also

fl_write_png(const char *, int, int, int, const unsigned char *)

14.258 Fl_XColor.H

```
//
// X-specific color definitions for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef FL_DOXYGEN
#include <config.h>
#include <FL/Enumerations.H>

// one of these for each color in fltk's "colormap":
// if overlays are enabled, another one for the overlay
struct Fl_XColor {
  unsigned char r,g,b;  // actual color used by X
  unsigned char mapped; // true when XAllocColor done
  unsigned long pixel;  // the X pixel to use
};
extern Fl_XColor fl_xmap[/*overlay*/][256];

// mask & shifts to produce xcolor for truecolor visuals:
extern unsigned char fl_redmask, fl_greenmask, fl_bluemask;
extern int fl_redshift, fl_greenshift, fl_blueshift, fl_extrashift;
#endif // FL_DOXYGEN
```

14.259 flstring.h

```
/*
 * Common string header file for the Fast Light Tool Kit (FLTK).
 * Internal use only (see "important note" below).
 *
 * Copyright 1998-2026 by Bill Spitzak and others.
 *
 * This library is free software. Distribution and use rights are outlined in
 * the file "COPYING" which should have been included with this file.  If this
 * file is missing or damaged, see the license at:
 *
 *     https://www.fltk.org/COPYING.php
 *
 * Please see the following page on how to report bugs and issues:
 *
 *     https://www.fltk.org/bugs.php
 */

 /*
 * Important note: this header file includes '<config.h>' !
 *
 * This header MUST NOT be included in public headers (i.e. in 'FL/') and
 * SHOULD NOT be included in test and demo programs (i.e. in 'test/' or
 * 'examples/') because it includes '<config.h>'.
 */

#ifndef flstring_h
#  define flstring_h

#  include <FL/Fl_Export.H>
#  include <config.h>
#  include <stdio.h>
#  include <stdarg.h>
#  include <string.h>
#  ifdef HAVE_STRINGS_H
#    include <strings.h>
#  endif /* HAVE_STRINGS_H */
#  include <FL/fl_string_functions.h>

/*
 * Apparently Unixware defines "index" to strchr (!) rather than
```

```
 * providing a proper entry point or not providing the (obsolete)
 * BSD function.  Make sure index is not defined...
 */

#  ifdef index
#    undef index
#  endif /* index */

/*
 * Visual C++ 2005 incorrectly displays a warning about the use of
 * POSIX APIs on Windows, which is supposed to be POSIX compliant...
 * Some of these functions are also defined in ISO C99...
 */

#  if defined(_WIN32) && !defined(__CYGWIN__) && !defined(__MINGW32__)
#    define strcasecmp(s,t)     _stricmp((s), (t))
#    define strncasecmp(s,t,n)  _strnicmp((s), (t), (n))
#  endif /* _WIN32 && ... */

#  ifdef __cplusplus
extern "C" {
#  endif /* __cplusplus */

FL_EXPORT extern int fl_snprintf(char *, size_t, const char *, ...);
#  ifndef HAVE_SNPRINTF
#    define snprintf fl_snprintf
#  endif /* !HAVE_SNPRINTF */

FL_EXPORT extern int fl_vsnprintf(char *, size_t, const char *, va_list ap);
#  ifndef HAVE_VSNPRINTF
#    define vsnprintf fl_vsnprintf
#  endif /* !HAVE_VSNPRINTF */

/*
 * strlcpy() and strlcat() are some really useful BSD string functions
 * that work the way strncpy() and strncat() *should* have worked.
 */

FL_EXPORT extern size_t fl_strlcat(char *, const char *, size_t);
#  ifndef HAVE_STRLCAT
#    define strlcat fl_strlcat
#  endif /* !HAVE_STRLCAT */

/* promoted to <FL/fl_string_functions.h> */
/* FL_EXPORT extern size_t fl_strlcpy(char *, const char *, size_t); */
#  ifndef HAVE_STRLCPY
#    define strlcpy fl_strlcpy
#  endif /* !HAVE_STRLCPY */

/*
 * Locale independent ASCII string compare function,
 * does not introduce locale issues as with strcasecmp()
 */
FL_EXPORT extern int fl_ascii_strcasecmp(const char *s, const char *t);


// Special character class detection and conversion functions for FLTK:
//
// The following functions replace system functions which are locale dependent
// and/or work only with pure ASCII characters (range: 0x00 .. 0x7f).
// Using input (bytes) outside this range results in undefined behavior and may
// even crash (e.g. Visual Studio in Debug mode).
// Note that UTF-8 sequences using 'char' variables would be sign-extended to
// large negative values which are invalid.
//
// Note: These functions are not UTF-8 aware and are intended to fix bad behavior,
//   but they don't fix wrong semantics. In the future we should check all text
//   handling functions for UTF-8 awareness.
//   These functions are locale independent and work as if locale 'C' or Posix
//   was active.
//
// These functions are intentionally declared and defined in this header in the
// `src' folder which is hidden from user code. They should not be used in demo
// programs in `test' and `examples' folders.
// These functions must not be "exported" (don't use FL_EXPORT).
//
// AlbrechtS, June 2026

/*
  This function can be used to replace isalnum(int) in FLTK.

  This function is \b NOT UTF-8 aware and \b should only be used where only ASCII
  checks are needed.

  \note isalnum() can only be used correctly on ASCII characters (bytes) in the
    range 0 .. 127. Everything else is locale dependent or results in undefined
    behavior.
```

```

  \param[in]  ch  input character
*/
inline int fl_ascii_isalnum(int ch) {
  return (ch >= 'a' && ch <= 'z') ||
         (ch >= 'A' && ch <= 'Z') ||
         (ch >= '0' && ch <= '9');
}

/*
  This function can be used to replace isalpha(int) in FLTK.

  This function is \b NOT UTF-8 aware and \b should only be used where only ASCII
  checks are needed.

  \note isalpha() can only be used correctly on ASCII characters (bytes) in the
    range 0 .. 127. Everything else is locale dependent or results in undefined
    behavior.

  \param[in]  ch  input character
*/
inline int fl_ascii_isalpha(int ch) {
  return (ch >= 'a' && ch <= 'z') ||
         (ch >= 'A' && ch <= 'Z');
}

/*
  This function can be used to replace isprint(int) in FLTK.

  This function is \b NOT UTF-8 aware and \b should only be used where only ASCII
  checks are needed.

  \note isprint() can only be used correctly on ASCII characters (bytes) in the
    range 0 .. 127. Everything else is locale dependent or results in undefined
    behavior.

  \param[in]  ch  input character
*/
inline int fl_ascii_isprint(int ch) {
  if (ch < 32 || ch > 0x7e) return 0;
  return 1;
}

/*
  This function can be used to replace ispunct(int) in FLTK.

  This function is \b NOT UTF-8 aware and \b should only be used where only ASCII
  checks are needed.

  \note ispunct() can only be used correctly on ASCII characters (bytes) in the
    range 0 .. 127. Everything else is locale dependent or results in undefined
    behavior.

  \param[in]  ch  input character
*/
inline int fl_ascii_ispunct(int ch) {
  return (ch >= 33 && ch <= 47) ||
         (ch >= 58 && ch <= 64) ||
         (ch >= 91 && ch <= 96) ||
         (ch >= 123 && ch <= 126);
}

/*
  This function can be used to replace isspace(int) in FLTK.

  This function is \b NOT UTF-8 aware and \b should only be used where only ASCII
  checks are needed.

  \note isspace() can only be used correctly on ASCII characters (bytes) in the
    range 0 .. 127. Everything else is locale dependent or results in undefined
    behavior.

  \param[in]  ch  input character
*/
inline int fl_ascii_isspace(int ch) {
  if (ch < 0 || ch > 32) return 0;
  return (ch == ' '  || // space
          ch == '\f' || // formfeed
          ch == '\n' || // newline
          ch == '\r' || // carriage return
          ch == '\t' || // horizontal tab
          ch == '\v');  // vertical tab
}

/*
  This function can be used to replace toupper(int) in FLTK.

```

```
  This function is \b NOT UTF-8 aware and \b should only be used where only ASCII
  checks are needed.

  \note toupper() can only be used correctly on ASCII characters (bytes) in the
    range 0 .. 127. Everything else is locale dependent or results in undefined
    behavior.

  \param[in]  ch  input character
*/
inline int fl_ascii_toupper(int ch) {
  if (ch >= 'a' && ch <= 'z')
    return ch - ('a' - 'A');
  return ch;
}

/*
  This function can be used to replace tolower(int) in FLTK.

  This function is \b NOT UTF-8 aware and \b should only be used where only ASCII
  checks are needed.

  \note tolower() can only be used correctly on ASCII characters (bytes) in the
    range 0 .. 127. Everything else is locale dependent or results in undefined
    behavior.

  \param[in]  ch  input character
*/
inline int fl_ascii_tolower(int ch) {
  if (ch >= 'A' && ch <= 'Z')
    return ch + ('a' - 'A');
  return ch;
}

/*
  This function can be used to replace isxdigit(int) in FLTK.

  This function is \b NOT UTF-8 aware and \b should only be used where only ASCII
  checks are needed.

  \note isxdigit() can only be used correctly on ASCII characters (bytes) in the
    range 0 .. 127. Everything else is locale dependent or results in undefined
    behavior.

  \param[in]  ch  input character
*/
inline int fl_ascii_isxdigit(int ch) {
  return (ch >= '0' && ch <= '9') ||
         (ch >= 'a' && ch <= 'f') ||
         (ch >= 'A' && ch <= 'F');
}

/*
  This function can be used to replace isdigit(int) in FLTK.

  This function is \b NOT UTF-8 aware and \b should only be used where only ASCII
  checks are needed.

  \note isdigit() can only be used correctly on ASCII characters (bytes) in the
    range 0 .. 127. Everything else is locale dependent or results in undefined
    behavior.

  \param[in]  ch  input character
*/
inline int fl_ascii_isdigit(int ch) {
  return (ch >= '0' && ch <= '9');
}


/*
  This function can be used to replace isupper(int) in FLTK.

  This function is \b NOT UTF-8 aware and \b should only be used where only ASCII
  checks are needed.

  \note isupper() can only be used correctly on ASCII characters (bytes) in the
    range 0 .. 127. Everything else is locale dependent or results in undefined
    behavior.

  \param[in]  ch  input character
*/
inline int fl_ascii_isupper(int ch) {
  return (ch >= 'A' && ch <= 'Z');
}

/*
  This function can be used to replace islower(int) in FLTK.

```

```
  This function is \b NOT UTF-8 aware and \b should only be used where only ASCII
  checks are needed.

  \note islower() can only be used correctly on ASCII characters (bytes) in the
    range 0 .. 127. Everything else is locale dependent or results in undefined
    behavior.

  \param[in]  ch  input character
*/
inline int fl_ascii_islower(int ch) {
  return (ch >= 'a' && ch <= 'z');
}
#  ifdef __cplusplus
}
#  endif /* __cplusplus */

#endif /* !flstring_h */
```

14.260 freeglut_teapot_data.h

```
/*
 * freeglut_teapot_data.h
 *
 * The freeglut library teapot data include file.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 * copy of this software and associated documentation files (the "Software"),
 * to deal in the Software without restriction, including without limitation
 * the rights to use, copy, modify, merge, publish, distribute, sublicense,
 * and/or sell copies of the Software, and to permit persons to whom the
 * Software is furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included
 * in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
 * OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.  IN NO EVENT SHALL
 * PAWEL W. OLSZTA BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
 * IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
 * CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

#ifndef  FREEGLUT_TEAPOT_DATA_H
#define  FREEGLUT_TEAPOT_DATA_H

/*
 * Original teapot code copyright follows:
 */

/*
 * (c) Copyright 1993, Silicon Graphics, Inc.
 *
 * ALL RIGHTS RESERVED
 *
 * Permission to use, copy, modify, and distribute this software
 * for any purpose and without fee is hereby granted, provided
 * that the above copyright notice appear in all copies and that
 * both the copyright notice and this permission notice appear in
 * supporting documentation, and that the name of Silicon
 * Graphics, Inc. not be used in advertising or publicity
 * pertaining to distribution of the software without specific,
 * written prior permission.
 *
 * THE MATERIAL EMBODIED ON THIS SOFTWARE IS PROVIDED TO YOU
 * "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EXPRESS, IMPLIED OR
 * OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF
 * MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.  IN NO
 * EVENT SHALL SILICON GRAPHICS, INC.  BE LIABLE TO YOU OR ANYONE
 * ELSE FOR ANY DIRECT, SPECIAL, INCIDENTAL, INDIRECT OR
 * CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES WHATSOEVER,
 * INCLUDING WITHOUT LIMITATION, LOSS OF PROFIT, LOSS OF USE,
 * SAVINGS OR REVENUE, OR THE CLAIMS OF THIRD PARTIES, WHETHER OR
 * NOT SILICON GRAPHICS, INC.  HAS BEEN ADVISED OF THE POSSIBILITY
 * OF SUCH LOSS, HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * ARISING OUT OF OR IN CONNECTION WITH THE POSSESSION, USE OR
 * PERFORMANCE OF THIS SOFTWARE.
 *
 * US Government Users Restricted Rights
 *
 * Use, duplication, or disclosure by the Government is subject to
 * restrictions set forth in FAR 52.227.19(c)(2) or subparagraph
 * (c)(1)(ii) of the Rights in Technical Data and Computer
 * Software clause at DFARS 252.227-7013 and/or in similar or
 * successor clauses in the FAR or the DOD or NASA FAR
```

```
 * Supplement.  Unpublished-- rights reserved under the copyright
 * laws of the United States.  Contractor/manufacturer is Silicon
 * Graphics, Inc., 2011 N.  Shoreline Blvd., Mountain View, CA
 * 94039-7311.
 *
 * OpenGL(TM) is a trademark of Silicon Graphics, Inc.
 */

/*
 * Rim, body, lid, and bottom data must be reflected in x and y;
 * handle and spout data across the y axis only.
 */
static int patchdata[][16] =
{
    { 102, 103, 104, 105,   4,   5,   6,   7,   8,   9,  10,  11,  12,  13,  14,  15 }, /* rim    */
    {  12,  13,  14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,  26,  27 }, /* body   */
    {  24,  25,  26,  27,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,  39,  40 },
    {  96,  96,  96,  96,  97,  98,  99, 100, 101, 101, 101, 101,   0,   1,   2,   3 }, /* lid    */
    {   0,   1,   2,   3, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117 },
    { 118, 118, 118, 118, 124, 122, 119, 121, 123, 126, 125, 120,  40,  39,  38,  37 }, /* bottom */
    {  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,  52,  53,  54,  55,  56 }, /* handle */
    {  53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,  28,  65,  66,  67 },
    {  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,  78,  79,  80,  81,  82,  83 }, /* spout  */
    {  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,  91,  92,  93,  94,  95 }
};

static double cpdata[][3] =
{
    {0.2, 0, 2.7}, {0.2, -0.112, 2.7}, {0.112, -0.2, 2.7}, {0,
    -0.2, 2.7}, {1.3375, 0, 2.53125}, {1.3375, -0.749, 2.53125},
    {0.749, -1.3375, 2.53125}, {0, -1.3375, 2.53125}, {1.4375,
    0, 2.53125}, {1.4375, -0.805, 2.53125}, {0.805, -1.4375,
    2.53125}, {0, -1.4375, 2.53125}, {1.5, 0, 2.4}, {1.5, -0.84,
    2.4}, {0.84, -1.5, 2.4}, {0, -1.5, 2.4}, {1.75, 0, 1.875},
    {1.75, -0.98, 1.875}, {0.98, -1.75, 1.875}, {0, -1.75,
    1.875}, {2, 0, 1.35}, {2, -1.12, 1.35}, {1.12, -2, 1.35},
    {0, -2, 1.35}, {2, 0, 0.9}, {2, -1.12, 0.9}, {1.12, -2,
    0.9}, {0, -2, 0.9}, {-2, 0, 0.9}, {2, 0, 0.45}, {2, -1.12,
    0.45}, {1.12, -2, 0.45}, {0, -2, 0.45}, {1.5, 0, 0.225},
    {1.5, -0.84, 0.225}, {0.84, -1.5, 0.225}, {0, -1.5, 0.225},
    {1.5, 0, 0.15}, {1.5, -0.84, 0.15}, {0.84, -1.5, 0.15}, {0,
    -1.5, 0.15}, {-1.6, 0, 2.025}, {-1.6, -0.3, 2.025}, {-1.5,
    -0.3, 2.25}, {-1.5, 0, 2.25}, {-2.3, 0, 2.025}, {-2.3, -0.3,
    2.025}, {-2.5, -0.3, 2.25}, {-2.5, 0, 2.25}, {-2.7, 0,
    2.025}, {-2.7, -0.3, 2.025}, {-3, -0.3, 2.25}, {-3, 0,
    2.25}, {-2.7, 0, 1.8}, {-2.7, -0.3, 1.8}, {-3, -0.3, 1.8},
    {-3, 0, 1.8}, {-2.7, 0, 1.575}, {-2.7, -0.3, 1.575}, {-3,
    -0.3, 1.35}, {-3, 0, 1.35}, {-2.5, 0, 1.125}, {-2.5, -0.3,
    1.125}, {-2.65, -0.3, 0.9375}, {-2.65, 0, 0.9375}, {-2,
    -0.3, 0.9}, {-1.9, -0.3, 0.6}, {-1.9, 0, 0.6}, {1.7, 0,
    1.425}, {1.7, -0.66, 1.425}, {1.7, -0.66, 0.6}, {1.7, 0,
    0.6}, {2.6, 0, 1.425}, {2.6, -0.66, 1.425}, {3.1, -0.66,
    0.825}, {3.1, 0, 0.825}, {2.3, 0, 2.1}, {2.3, -0.25, 2.1},
    {2.4, -0.25, 2.025}, {2.4, 0, 2.025}, {2.7, 0, 2.4}, {2.7,
    -0.25, 2.4}, {3.3, -0.25, 2.4}, {3.3, 0, 2.4}, {2.8, 0,
    2.475}, {2.8, -0.25, 2.475}, {3.525, -0.25, 2.49375},
    {3.525, 0, 2.49375}, {2.9, 0, 2.475}, {2.9, -0.15, 2.475},
    {3.45, -0.15, 2.5125}, {3.45, 0, 2.5125}, {2.8, 0, 2.4},
    {2.8, -0.15, 2.4}, {3.2, -0.15, 2.4}, {3.2, 0, 2.4}, {0, 0,
    3.15}, {0.8, 0, 3.15}, {0.8, -0.45, 3.15}, {0.45, -0.8,
    3.15}, {0, -0.8, 3.15}, {0, 0, 2.85}, {1.4, 0, 2.4}, {1.4,
    -0.784, 2.4}, {0.784, -1.4, 2.4}, {0, -1.4, 2.4}, {0.4, 0,
    2.55}, {0.4, -0.224, 2.55}, {0.224, -0.4, 2.55}, {0, -0.4,
    2.55}, {1.3, 0, 2.55}, {1.3, -0.728, 2.55}, {0.728, -1.3,
    2.55}, {0, -1.3, 2.55}, {1.3, 0, 2.4}, {1.3, -0.728, 2.4},
    {0.728, -1.3, 2.4}, {0, -1.3, 2.4}, {0, 0, 0}, {1.425,
    -0.798, 0}, {1.5, 0, 0.075}, {1.425, 0, 0}, {0.798, -1.425,
    0}, {0, -1.5, 0.075}, {0, -1.425, 0}, {1.5, -0.84, 0.075},
    {0.84, -1.5, 0.075}
};

static double tex[2][2][2] =
{
    { {0.0, 0.0}, {1.0, 0.0} },
    { {0.0, 1.0}, {1.0, 1.0} }
};


#endif  /* FREEGLUT_TEAPOT_DATA_H */

```

14.261 mediumarrow.h

```
#define mediumarrow_width 16
#define mediumarrow_height 16
```

```
00003 static const unsigned char mediumarrow_bits[] = {
00004    0x40, 0x00, 0x60, 0x00, 0x70, 0x00, 0x78, 0x00, 0xfc, 0x3f, 0x78, 0x00,
00005    0x70, 0x00, 0x60, 0x02, 0x40, 0x06, 0x00, 0x0e, 0x00, 0x1e, 0xfc, 0x3f,
00006    0x00, 0x1e, 0x00, 0x0e, 0x00, 0x06, 0x00, 0x02};
```

14.262 numericsort.c File Reference

```
#include "flstring.h"
#include <stdlib.h>
#include <string.h>
#include <FL/platform_types.h>
#include <FL/filename.H>
#include <FL/fl_utf8.h>
```

Functions

- int fl_casenumericsort (struct dirent **A, struct dirent **B)

  Compares directory entries alphanumerically (case-insensitive).
- int fl_numericsort (struct dirent **A, struct dirent **B)

  Compares directory entries alphanumerically (case-sensitive).

14.262.1 Function Documentation

14.262.1.1 fl_casenumericsort()

```
int fl_casenumericsort (
            struct dirent ** A,
            struct dirent ** B )
```

Compares directory entries alphanumerically (case-insensitive).

Note

This comparison is UTF-8 aware.

See also

fl_numericsort()

14.262.1.2 fl_numericsort()

```
int fl_numericsort (
            struct dirent ** A,
            struct dirent ** B )
```

Compares directory entries alphanumerically (case-sensitive).
Numbers are compared without sign, i.e. "-" is not taken as a sign of following numerical values. The following list of files would be in ascending order (examples are ASCII and numbers only for simplicity):

1. 1zzz.txt
2. 2xxx.txt
3. 19uuu.txt
4. 100aaa.txt
5. file1z.txt
6. file5a.txt
7. file5z.txt

8. file30z.txt
9. file200a.txt
10. temp+5.txt ('+' is lexically lower than '-')
11. temp-5.txt ('-' is not a sign)
12. temp-100.txt (100 is bigger than 5, no sign)

Parameters

in	*A*	first directory entry
in	*B*	second directory entry

Returns

comparison result (-1, 0, or +1)

Return values

-1	A < B
0	A == B
+1	A > B

Note

This comparison is UTF-8 aware.

See also

fl_casenumericsort()

14.263 print_button.h

```
//
// Header for "Print Window" functions for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2022 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

#ifndef _SRC_FL_PRINT_BUTTON_H_
#define _SRC_FL_PRINT_BUTTON_H_

#include <FL/Fl_Window.H>

// These are all internal functions, do not FL_EXPORT these functions!
// These functions are mplemented in src/print_button.cxx

// Create and initialize the "Print/copy front window" dialog window

int fl_create_print_window();

// Print a window or copy its contents to the clipboard.

int fl_print_or_copy_window(Fl_Window *win, bool grab_decoration, int mode);

#endif // _SRC_FL_PRINT_BUTTON_H_
```

14.264 print_panel.h

```
//
// Print panel for the Fast Light Tool Kit (FLTK).
//
// Copyright 1998-2010 by Bill Spitzak and others.
//
// This library is free software. Distribution and use rights are outlined in
// the file "COPYING" which should have been included with this file.  If this
// file is missing or damaged, see the license at:
//
//     https://www.fltk.org/COPYING.php
//
// Please see the following page on how to report bugs and issues:
//
//     https://www.fltk.org/bugs.php
//

//
// This is a temporary file.  It is only for development and will
// probably be removed later.
//

#ifndef print_panel_h
#define print_panel_h
#include <FL/Fl.H>
#include <FL/Fl_Double_Window.H>
#include <FL/Fl_Group.H>
#include <FL/Fl_Choice.H>
#include <FL/Fl_Button.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Round_Button.H>
#include <FL/Fl_Input.H>
#include <FL/Fl_Spinner.H>
#include <FL/Fl_Check_Button.H>
#include <FL/Fl_Return_Button.H>
#include <FL/Fl_Progress.H>
enum printing_style {SystemV, BSD};
static Fl_Double_Window* make_print_panel();
static void print_cb(Fl_Return_Button *, void *);
static printing_style print_load();
static void print_update_status();
#endif
```

14.265 slowarrow.h

```
#define slowarrow_width 16
#define slowarrow_height 16
static const unsigned char slowarrow_bits[] = {
   0x40, 0x00, 0x40, 0x00, 0x60, 0x00, 0x60, 0x00, 0xf0, 0x0f, 0x60, 0x00,
   0x60, 0x00, 0x40, 0x02, 0x40, 0x02, 0x00, 0x06, 0x00, 0x06, 0xf0, 0x0f,
   0x00, 0x06, 0x00, 0x06, 0x00, 0x02, 0x00, 0x02};
```

14.266 utf8_internal.h

```
/*
 * Internal UTF-8 header file for the Fast Light Tool Kit (FLTK).
 *
 * Copyright 1998-2016 by Bill Spitzak and others.
 *
 * This library is free software. Distribution and use rights are outlined in
 * the file "COPYING" which should have been included with this file.  If this
 * file is missing or damaged, see the license at:
 *
 *     https://www.fltk.org/COPYING.php
 *
 * Please see the following page on how to report bugs and issues:
 *
 *     https://www.fltk.org/bugs.php
 */

/*
  ----------------
  Note to editors:
  ----------------

  This file may only contain common, platform-independent function
  declarations used internally in FLTK. It may be #included everywhere
  in source files in the library, but not in public header files.
*/

#ifndef _SRC__FL_UTF8_H
```

```
#define _SRC__FL_UTF8_H

#  ifdef __cplusplus
extern "C" {
#  endif

unsigned short
XUtf8IsNonSpacing(
        unsigned int ucs);

unsigned short
XUtf8IsRightToLeft(
        unsigned int ucs);


int
XUtf8Tolower(
        int ucs);

int
XUtf8Toupper(
        int ucs);


#  ifdef __cplusplus
}
#  endif

#endif /* _SRC__FL_UTF8_H */
```

14.267 vsnprintf.c File Reference

Portable vsnprintf() implementation.

```
#include <stdio.h>
#include "flstring.h"
```

Functions

- int **fl_snprintf** (char ∗str, size_t size, const char ∗fmt,...)
- int fl_vsnprintf (char ∗buffer, size_t bufsize, const char ∗format, va_list ap)

  FLTK's platform independent wrapper for the vsnprintf() C library function.

14.267.1 Detailed Description

Portable vsnprintf() implementation.

14.267.2 Function Documentation

14.267.2.1 fl_vsnprintf()

```
int fl_vsnprintf (
            char * buffer,
            size_t bufsize,
            const char * format,
            va_list ap )
```

FLTK's platform independent wrapper for the vsnprintf() C library function.
This function guarantees:

- access to vsnprintf(), even on systems that don't have it (FLTK's own built-in code is used)
- Guarantees NUL termination. Even if string expands larger than the buffer, a terminating NUL is included, unlike some implementations of vsnprintf(), notably Microsoft Visual Studio (pre-2015), which can leave the string unterminated when truncated.

If the build environment for FLTK has vsnprintf(), fl_vsnprintf() is just a wrapper around the compiler's provided function. Otherwise, if the function is NOT available, FLTK's own built-in version is provided.
The FLTK built in provides these style options:

- %[-+#']
- ∗ – padding width
- .∗ – precision width
- Data types: h, l, ll, L
- Floating point formats: E, G, e, f, g
- Integer formats: B, X, b, d, i, o, u, x
- Pointer format: p
- String/char: c, s, n

14.268 Xutf8.h

```
/*
 * Author: Jean-Marc Lienher ( http://oksid.ch )
 * Copyright 2000-2010 by O'ksi'D.
 *
 * This library is free software. Distribution and use rights are outlined in
 * the file "COPYING" which should have been included with this file.  If this
 * file is missing or damaged, see the license at:
 *
 *     https://www.fltk.org/COPYING.php
 *
 * Please see the following page on how to report bugs and issues:
 *
 *     https://www.fltk.org/bugs.php
 */

#if ! ( defined(_Xutf8_h) || defined(FL_DOXYGEN) )
#define _Xutf8_h

#  ifdef __cplusplus
extern "C" {
#  endif

#include <X11/X.h>
#include <X11/Xlib.h>
#include <X11/Xlocale.h>
#include <X11/Xutil.h>
#include <FL/Fl_Export.H>

typedef struct {
        int nb_font;
        char **font_name_list;
        int *encodings;
        XFontStruct **fonts;
        Font fid;
        int ascent;
        int descent;
        int *ranges;
} XUtf8FontStruct;

XUtf8FontStruct *
XCreateUtf8FontStruct (
        Display         *dpy,
        const char      *base_font_name_list);

void
XUtf8DrawString(
        Display                 *display,
        Drawable                d,
        XUtf8FontStruct  *font_set,
        GC                      gc,
        int                     x,
        int                     y,
        const char              *string,
        int                     num_bytes);

void
XUtf8_measure_extents(
        Display                 *display,
        Drawable                d,
        XUtf8FontStruct  *font_set,
        GC                      gc,
        int                     *xx,
        int                     *yy,
```

```
        int                     *ww,
        int                     *hh,
        const char              *string,
        int                     num_bytes);

void
XUtf8DrawRtlString(
        Display                 *display,
        Drawable                d,
        XUtf8FontStruct  *font_set,
        GC                      gc,
        int                     x,
        int                     y,
        const char              *string,
        int                     num_bytes);

void
XUtf8DrawImageString(
        Display         *display,
        Drawable        d,
        XUtf8FontStruct         *font_set,
        GC              gc,
        int             x,
        int             y,
        const char      *string,
        int             num_bytes);

int
XUtf8TextWidth(
        XUtf8FontStruct  *font_set,
        const char              *string,
        int                     num_bytes);
int
XUtf8UcsWidth(
        XUtf8FontStruct  *font_set,
        unsigned int            ucs);

FL_EXPORT int
fl_XGetUtf8FontAndGlyph(
        XUtf8FontStruct  *font_set,
        unsigned int            ucs,
        XFontStruct     **fnt,
        unsigned short  *id);

void
XFreeUtf8FontStruct(
        Display                 *dpy,
        XUtf8FontStruct         *font_set);


int
XConvertUtf8ToUcs(
        const unsigned char     *buf,
        int                     len,
        unsigned int            *ucs);

int
XConvertUcsToUtf8(
        unsigned int            ucs,
        char                    *buf);

int
XUtf8CharByteLen(
        const unsigned char     *buf,
        int                     len);

int
XCountUtf8Char(
        const unsigned char *buf,
        int len);

int
XFastConvertUtf8ToUcs(
        const unsigned char     *buf,
        int                     len,
        unsigned int            *ucs);

long
XKeysymToUcs(
        KeySym  keysym);

#ifdef X_HAVE_UTF8_STRING
#define XUtf8LookupString Xutf8LookupString
#else
int
XUtf8LookupString(
    XIC                 ic,
```

```
    XKeyPressedEvent*   event,
    char*               buffer_return,
    int                 bytes_buffer,
    KeySym*             keysym,
    Status*             status_return);
#endif

#  ifdef __cplusplus
}
#  endif

#endif
```

Index

_FL_EMBOSSED_LABEL
Enumerations.H, 2082
_FL_ENGRAVED_LABEL
Enumerations.H, 2082
_FL_ICON_LABEL
Enumerations.H, 2082
_FL_IMAGE_LABEL
Enumerations.H, 2082
_FL_MULTI_LABEL
Enumerations.H, 2082
_FL_SHADOW_LABEL
Enumerations.H, 2082
_RESERVED_1
Fl_Terminal, 1650
_RESERVED_2
Fl_Terminal, 1650
__fl_attr
fl_attr.h, 2114
_remove
Fl_Browser, 502
~Cell
Fl_Grid::Cell, 441
~Fl_Anim_GIF_Image
Fl_Anim_GIF_Image, 461
~Fl_Double_Window
Fl_Double_Window, 690
~Fl_EPS_File_Surface
Fl_EPS_File_Surface, 694
~Fl_Group
Fl_Group, 884
~Fl_Help_View
Fl_Help_View, 908
~Fl_Input_
Fl_Input_, 1006
~Fl_Native_File_Chooser
Fl_Native_File_Chooser, 1183
~Fl_Plugin_Manager
Fl_Plugin_Manager, 1254
~Fl_Preferences
Fl_Preferences, 1287
~Fl_SVG_File_Surface
Fl_SVG_File_Surface, 1539
~Fl_Scroll
Fl_Scroll, 1415
~Fl_Shared_Image
Fl_Shared_Image, 1461
~Fl_Table
Fl_Table, 1579
~Fl_Table_Row
Fl_Table_Row, 1604
~Fl_Terminal
Fl_Terminal, 1652
~Fl_Text_Display
Fl_Text_Display, 1709
~Fl_Widget
Fl_Widget, 1950
~Fl_Window
Fl_Window, 2001

A0
Fl_Paged_Device, 1233
A1
Fl_Paged_Device, 1233
A2
Fl_Paged_Device, 1233
A3
Fl_Paged_Device, 1233
A4
Fl_Paged_Device, 1234
A5
Fl_Paged_Device, 1234
A6
Fl_Paged_Device, 1234
A7
Fl_Paged_Device, 1234
A8
Fl_Paged_Device, 1234
A9
Fl_Paged_Device, 1234
abi_check
Fl, 416
abi_version
Fl, 416
about
Fl_Sys_Menu_Bar, 1560
absolute_top_line_number
Fl_Text_Display, 1709
activate
Fl_Tree_Item, 1872
Fl_Widget, 1950
active
Fl_Widget, 1950
active_r
Fl_Widget, 1950
add
Fl_Browser, 502
Fl_Chart, 578
Fl_Check_Browser, 594
Fl_File_Icon, 720

Fl_Input_Choice, 1033
Fl_Menu_, 1079
Fl_Menu_Item, 1120
Fl_Shared_Image, 1461
Fl_Sys_Menu_Bar, 1560, 1561
Fl_Tree, 1835
Fl_Tree_Item, 1872
Fl_Tree_Item_Array, 1888
add_check
Fl, 416
add_clipboard_notify
Selection & Clipboard functions, 281
add_color
Fl_File_Icon, 720
add_extra
Fl_File_Chooser, 716
add_fd
Fl, 416
add_handler
Events handling functions, 252
Fl_Shared_Image, 1461
add_idle
Fl, 417
add_key_binding
Fl_Text_Editor, 1756
add_modify_callback
Fl_Text_Buffer, 1678
add_scheme_name
Fl_Scheme, 1393
add_system_handler
Events handling functions, 253
add_timeout
Fl, 417
Fl_Timeout, 1789
add_vertex
Fl_File_Icon, 720, 721
Adding and Extending Widgets, 83
addPlugin
Fl_Plugin_Manager, 1254
address
Fl_Text_Buffer, 1679
Advanced FLTK, 100
align
Fl_Widget, 1951
alloc_size
Fl_Flex, 765
allow_expand_outside_parent
Fl_Window, 2001
ALWAYS_ON
Fl_Browser_, 530
angle1
Fl_Dial, 675
animate
Fl_GIF_Image, 815
ansi
Fl_Terminal, 1652
ANY_DOWN
Pen and tablet event handling, 268
api_version
Fl, 418
append
Fl_Input_, 1006
Fl_Terminal, 1653
Fl_Text_Buffer, 1679
append_ascii
Fl_Terminal, 1653
append_utf8
Fl_Terminal, 1653
appendfile
Fl_Text_Buffer, 1679
apply_undo
Fl_Input_, 1006
arg
Fl, 418
args
Fl, 419
args_to_utf8
Fl, 420
argument
Fl_Menu_Item, 1120
Fl_Widget, 1951
array
Fl_Group, 885
Fl_RGB_Image, 1366
Fl_Table, 1580
arrow_widths
Fl_Counter, 664
as_double_window
Fl_Double_Window, 690
Fl_Window, 2001
as_gl_window
Fl_Gl_Window, 830
Fl_Widget, 1951
as_group
Fl_Group, 885
Fl_Widget, 1952
as_overlay_window
Fl_Overlay_Window, 1219
Fl_Window, 2001
as_shared_image
Fl_Image, 970
Fl_Shared_Image, 1461
as_svg_image
Fl_RGB_Image, 1364
Fl_SVG_Image, 1545
as_window
Fl_Widget, 1952
Fl_Window, 2001
atclose
Windows handling functions, 280
ATTR_BGCOLOR
Fl_Text_Display, 1708
ATTR_BGCOLOR_EXT
Fl_Text_Display, 1708
ATTR_BGCOLOR_EXT_
Fl_Text_Display, 1708

ATTR_GRAMMAR
Fl_Text_Display, 1708
ATTR_LINES_MASK
Fl_Text_Display, 1708
ATTR_SPELLING
Fl_Text_Display, 1708
ATTR_STRIKE_THROUGH
Fl_Text_Display, 1708
ATTR_UNDERLINE
Fl_Text_Display, 1708
Attrib
Fl_Terminal, 1650
AUTO_DELETE_USER_DATA
Fl_Widget, 1949
auto_resize_children
Fl_Tree, 1836
autosize
Fl_Chart, 578
awake
Multithreading support functions, 343
awake_once
Multithreading support functions, 344

b
Fl_Color_Chooser, 648
Fl_Rect, 1341
B0
Fl_Paged_Device, 1234
B1
Fl_Paged_Device, 1234
B10
Fl_Paged_Device, 1234
B2
Fl_Paged_Device, 1234
B3
Fl_Paged_Device, 1234
B4
Fl_Paged_Device, 1234
B5
Fl_Paged_Device, 1234
B6
Fl_Paged_Device, 1234
B7
Fl_Paged_Device, 1234
B8
Fl_Paged_Device, 1234
B9
Fl_Paged_Device, 1234
background
Fl, 421
background2
Fl, 421
BARREL_PRESSURE
Pen and tablet event handling, 269
bbox
Fl_Browser_, 531
Fl_Scroll, 1416
begin
Fl_Group, 885
begin_document
Fl_PDF_File_Surface, 1240
begin_job
Fl_Paged_Device, 1234
Fl_PDF_File_Surface, 1240
Fl_PostScript_File_Device, 1275
Fl_Printer, 1304
begin_page
Fl_Paged_Device, 1235
Fl_PDF_File_Surface, 1241
Fl_PostScript_File_Device, 1275
Fl_Printer, 1304
belowmouse
Events handling functions, 253
BG_XTERM
Fl_Terminal, 1650
bind_deimage
Fl_Widget, 1953
bind_image
Fl_Widget, 1953, 1954
bline_length
Fl_Browser, 503
BLOCK_CURSOR
Fl_Text_Display, 1708
BOLD
Fl_Terminal, 1650
border
Fl_Window, 2002
BOTH
Fl_Browser_, 530
BOTH_ALWAYS
Fl_Browser_, 530
bottomline
Fl_Browser, 503
bounds
Fl_Chart, 579
Fl_Group, 885
Fl_Slider, 1508
box
Fl_Terminal, 1654
Fl_Widget, 1954
box_bg
Fl, 421
box_border_radius_max
Fl, 422
box_color
Fl, 422
box_dh
Fl, 422
box_dw
Fl, 422
box_dx
Fl, 422
box_dy
Fl, 423
box_shadow_width
Fl, 423
BROWSE_DIRECTORY

Fl_Native_File_Chooser, 1182
BROWSE_FILE
Fl_Native_File_Chooser, 1182
BROWSE_MULTI_DIRECTORY
Fl_Native_File_Chooser, 1182
BROWSE_MULTI_FILE
Fl_Native_File_Chooser, 1182
BROWSE_SAVE_DIRECTORY
Fl_Native_File_Chooser, 1182
BROWSE_SAVE_FILE
Fl_Native_File_Chooser, 1182
buffer
Fl_Text_Display, 1709
buffer_modified_cb
Fl_Text_Display, 1710
buffer_predelete_cb
Fl_Text_Display, 1710
BUTTON0
Pen and tablet event handling, 268
BUTTON1
Pen and tablet event handling, 268
BUTTON2
Pen and tablet event handling, 268
BUTTON3
Pen and tablet event handling, 268
BUTTON_PUSH
Pen and tablet event handling, 268
BUTTON_RELEASE
Pen and tablet event handling, 268
byte_at
Fl_Text_Buffer, 1679

C_LOCALE
Fl_Preferences, 1284
Cairo Support Functions and Classes, 345
cairo_autolink_context, 346
cairo_cc, 346
cairo_flush, 346
cairo_make_current, 347
cairo_autolink_context
Cairo Support Functions and Classes, 346
cairo_cc
Cairo Support Functions and Classes, 346
cairo_flush
Cairo Support Functions and Classes, 346
cairo_make_current
Cairo Support Functions and Classes, 347
Fl_Private.H, 2379
calc_dimensions
Fl_Tree, 1837
calc_item_height
Fl_Tree_Item, 1873
calc_last_char
Fl_Text_Display, 1710
calc_line_starts
Fl_Text_Display, 1711
calc_tree
Fl_Tree, 1837
callback
Fl_Menu_Item, 1120, 1121
Fl_Table, 1580
Fl_Widget, 1954, 1955
Callback Function Typedefs, 276
Fl_Event_Dispatch, 277
Fl_Idle_Handler, 277
Fl_Old_Idle_Handler, 277
Fl_Timeout_Handler, 277
callback_col
Fl_Table, 1581
callback_context
Fl_Table, 1581
callback_item
Fl_Tree, 1837
callback_reason
Events handling functions, 253
Fl_Tree, 1837, 1838
callback_row
Fl_Table, 1581
can_do
Fl_Gl_Window, 831
can_do_overlay
Fl_Gl_Window, 831
can_redo
Fl_Input_, 1006
Fl_Text_Buffer, 1680
can_undo
Fl_Input_, 1007
Fl_Text_Buffer, 1680
canUndo
Fl_Text_Buffer, 1680
canvas
Fl_Anim_GIF_Image, 461
canvas_h
Fl_Anim_GIF_Image, 462
canvas_w
Fl_Anim_GIF_Image, 462
CARET_CURSOR
Fl_Text_Display, 1708
cc
Fl_Cairo_State, 555
cell
Fl_Grid, 865, 866
cgdebug.h, 2354
CHANGED
Fl_Widget, 1949
Pen and tablet event handling, 268
changed
Fl_Widget, 1956
char_at
Fl_Text_Buffer, 1680
CharFlags
Fl_Terminal, 1650
check
Fl, 423
checkbox
Fl_Menu_Item, 1121
child

Fl_Group, 886
Fl_Table, 1581
Fl_Tree_Item, 1873
children
Fl_Table, 1581
CLEAR
Fl_Preferences, 1284
clear
Fl_Browser, 503
Fl_Button, 551
Fl_Group, 886
Fl_Menu_, 1081
Fl_Pack, 1229
Fl_Sys_Menu_Bar, 1561
Fl_Table, 1581
Fl_Table_Row, 1604
Fl_Terminal, 1654
Fl_Tree, 1838
Fl_Tree_Item_Array, 1888
clear_active
Fl_Widget, 1956
clear_border
Fl_Window, 2002
clear_changed
Fl_Widget, 1956
clear_children
Fl_Tree, 1838
clear_damage
Fl_Widget, 1956
clear_layout
Fl_Grid, 866
clear_modal_states
Fl_Window, 2002
clear_output
Fl_Widget, 1957
clear_rect
Fl_Text_Display, 1711
clear_screen
Fl_Terminal, 1654
clear_screen_home
Fl_Terminal, 1654
clear_submenu
Fl_Menu_, 1082
Fl_Sys_Menu_Bar, 1561
clear_tab_positions
Fl_Tabs, 1620
clear_visible
Fl_Widget, 1957
clear_visible_focus
Fl_Widget, 1957
clear_widget_pointer
Safe widget deletion support functions, 373
client_area
Fl_Tabs, 1620
CLIP_CHILDREN
Fl_Widget, 1949
clip_children
Fl_Group, 887
clipboard_contains
Selection & Clipboard functions, 281
close
Fl_EPS_File_Surface, 694
Fl_SVG_File_Surface, 1539
Fl_Tree, 1838, 1839
closedeicon
Fl_Tree_Prefs, 1893
closeicon
Fl_Tree, 1839
Fl_Tree_Prefs, 1893
col_gap
Fl_Grid, 866
col_header
Fl_Table, 1581
col_resize
Fl_Table, 1581
col_resize_min
Fl_Table, 1582
col_to_x
Fl_Text_Display, 1711
col_weight
Fl_Grid, 867
col_width
Fl_Grid, 868
Fl_Table, 1582
col_width_all
Fl_Table, 1582
color
Fl_Terminal, 1655
Fl_Tooltip, 1813
Fl_Widget, 1957, 1958
Color & Font functions, 291
fl_color, 293, 294
fl_color_average, 294
fl_contrast, 294
fl_contrast_function, 296
fl_contrast_level, 296
fl_contrast_mode, 297
fl_descent, 298
fl_font, 299
fl_height, 299
fl_latin1_to_local, 299
fl_lightness, 300
fl_local_to_latin1, 300
fl_local_to_mac_roman, 300
fl_luminance, 301
fl_mac_roman_to_local, 301
fl_show_colormap, 301
fl_size, 302
fl_text_extents, 302, 303
fl_width, 303, 304
free_color, 304
get_color, 304
get_font, 305
get_font_name, 305
get_font_sizes, 305
set_color, 305, 306

set_font, 306
set_fonts, 306
color_average
Fl_Anim_GIF_Image, 462
Fl_Image, 970
Fl_Pixmap, 1247
Fl_RGB_Image, 1364
Fl_Shared_Image, 1461
Fl_SVG_Image, 1546
Fl_Tiled_Image, 1785
COLUMN
Fl_Flex, 763
column_char
Fl_Browser, 503
column_widths
Fl_Browser, 504
Common Dialog Classes and Functions, 376
error, 392
fatal, 392
fl_alert, 378
fl_beep, 379
fl_choice, 379
fl_choice_n, 380
fl_color_chooser, 381, 382
fl_dir_chooser, 383
fl_file_chooser, 383
fl_file_chooser_callback, 384
fl_file_chooser_ok_label, 384
fl_input, 385
fl_input_str, 386
fl_message, 386
fl_message_hotspot, 387
fl_message_icon, 387
fl_message_icon_label, 388
fl_message_position, 388, 389
fl_message_title, 390
fl_message_title_default, 390
fl_password, 390, 391
fl_password_str, 391, 392
warning, 393
Common Widgets and Attributes, 16
compact
Fl_Button, 551
compare
Fl_Shared_Image, 1462
compose
Events handling functions, 254
compose_reset
Events handling functions, 254
connectorstyle
Fl_Tree, 1840
Constants and Enumerations, 111
contains
Fl_Widget, 1958
context
Fl_Gl_Window, 831
CONTEXT_CELL
Fl_Table, 1579
CONTEXT_COL_HEADER
Fl_Table, 1579
CONTEXT_ENDPAGE
Fl_Table, 1579
CONTEXT_NONE
Fl_Table, 1579
CONTEXT_RC_RESIZE
Fl_Table, 1579
CONTEXT_ROW_HEADER
Fl_Table, 1579
CONTEXT_STARTPAGE
Fl_Table, 1579
CONTEXT_TABLE
Fl_Table, 1579
context_valid
Fl_Gl_Window, 831
Coordinates and Layout Widgets, 28
COPIED_LABEL
Fl_Widget, 1949
COPIED_TOOLTIP
Fl_Widget, 1949
copy
Fl_Anim_GIF_Image, 462
Fl_Bitmap, 473
Fl_Help_View, 908
Fl_Image, 970, 971
Fl_Input_, 1007
Fl_Menu_, 1082
Fl_Pixmap, 1247
Fl_RGB_Image, 1364
Fl_Shared_Image, 1462, 1463
Fl_SVG_Image, 1546
Fl_Text_Buffer, 1681
Fl_Tiled_Image, 1786
Selection & Clipboard functions, 281
copy_
Fl_Shared_Image, 1463
copy_cuts
Fl_Input_, 1007
copy_label
Fl_Widget, 1958
copy_tooltip
Fl_Widget, 1958
CORE
Fl_Preferences, 1284
CORE_READ_OK
Fl_Preferences, 1299
CORE_SYSTEM
Fl_Preferences, 1284
CORE_SYSTEM_L
Fl_Preferences, 1284
CORE_USER
Fl_Preferences, 1284
CORE_USER_L
Fl_Preferences, 1284
CORE_WRITE_OK
Fl_Preferences, 1299
count

Fl_Image, 971
Fl_Native_File_Chooser, 1183
count_displayed_characters
Fl_Text_Buffer, 1681
count_lines
Fl_Text_Buffer, 1681
Fl_Text_Display, 1711
CR_TO_LF
Fl_Terminal, 1651
CREATE
Fl_File_Chooser, 716
create_window_menu
Fl_Sys_Menu_Bar, 1561
current
Fl_Group, 887
Fl_Timeout, 1790
Fl_Tooltip, 1813
Fl_Window, 2002
current_
Fl_Window, 2021
current_timeout
Fl_Timeout, 1794
cursor
Fl_Tile, 1776
Fl_Window, 2002, 2003
cursor2rowcol
Fl_Table, 1582
cursor_col
Fl_Terminal, 1655
cursor_color
Fl_Input_, 1007
Fl_Text_Display, 1712
Fl_Value_Input, 1915
cursor_cr
Fl_Terminal, 1655
cursor_down
Fl_Terminal, 1655
cursor_right
Fl_Terminal, 1655
cursor_row
Fl_Terminal, 1656
cursor_style
Fl_Text_Display, 1712
cursor_up
Fl_Terminal, 1656
custom_application_menu_items
Fl_Mac_App_Menu, 1069
cut
Fl_Input_, 1008

d
Fl_Image, 972
damage
Fl_Widget, 1959
damage_zone
Fl_Table, 1582
data
Fl_Browser, 504, 505
Fl_Image, 972
deactivate
Fl_Menu_Item, 1121
Fl_Tree_Item, 1873
Fl_Widget, 1960
debug
Fl_Grid, 868
DEBUG_FLAG
Fl_Anim_GIF_Image, 460
decorated_h
Fl_Window, 2003
decorated_w
Fl_Window, 2003
default_atclose
Windows handling functions, 278
default_callback
Fl_Widget, 1960
default_cursor
Fl_Window, 2004
default_icon
Fl_Window, 2004
default_icons
Fl_Window, 2004, 2005
default_size_range
Fl_Window, 2005
default_xclass
Fl_Window, 2006
deimage
Fl_Widget, 1960, 1961
DEIMAGE_BOUND
Fl_Widget, 1949
deimage_bound
Fl_Widget, 1961
delay
Fl_Anim_GIF_Image, 463
Fl_Tooltip, 1813
delete_child
Fl_Group, 887
Fl_Scroll, 1416
delete_entry
Fl_Preferences, 1288
delete_group
Fl_Preferences, 1288
delete_rows
Fl_Terminal, 1656
delete_widget
Safe widget deletion support functions, 373
deleted
Fl_Widget_Tracker, 1987
deleting
Fl_Browser_, 531
deparent
Fl_Tree_Item, 1873
Fl_Tree_Item_Array, 1888
Deprecated List, 221
depth
Fl_Tree_Item, 1874
desaturate
Fl_Anim_GIF_Image, 463

Fl_Image, 972
Fl_Pixmap, 1248
Fl_RGB_Image, 1365
Fl_Shared_Image, 1463
Fl_SVG_Image, 1546
Fl_Tiled_Image, 1786
deselect
Fl_Browser_, 531
Fl_Tree, 1840
deselect_all
Fl_Tree, 1841
Fl_Tree_Item, 1874
Designing a Simple Text Editor, 38
DETECTED
Pen and tablet event handling, 268, 269
Developer info for bundled libs, 207
Developer Information, 211
Development of the FLTK library, 175
DIM
Fl_Terminal, 1650
DIM_CURSOR
Fl_Text_Display, 1708
direction
Fl_Timer, 1802
DIRECTORY
Fl_File_Chooser, 716
directory
Fl_Help_View, 908
Fl_Native_File_Chooser, 1183
dirty
Fl_Preferences, 1289
disable
Fl_Tooltip, 1813
disable_im
Events handling functions, 254
DISPLAY
Pen and tablet event handling, 269
display
Fl, 423
Fl_Browser, 505
Fl_Browser_, 532
Fl_Tree, 1841
display_columns
Fl_Terminal, 1656
display_insert
Fl_Text_Display, 1712
display_needs_recalc
Fl_Text_Display, 1713
display_rows
Fl_Terminal, 1656
displayed
Fl_Browser, 505
Fl_Browser_, 532
Fl_Tree, 1841
dnd
Selection & Clipboard functions, 282
dnd_text_ops
Fl, 424
do_callback
Fl_Menu_Item, 1121, 1122
Fl_Table, 1582
Fl_Widget, 1961, 1962
do_widget_deletion
Safe widget deletion support functions, 374
DONT_RESIZE_CANVAS
Fl_Anim_GIF_Image, 460
DONT_SET_AS_IMAGE
Fl_Anim_GIF_Image, 460
DONT_START
Fl_Anim_GIF_Image, 460
down_box
Fl_Button, 552
Fl_File_Input, 734
Fl_Menu_, 1082
drag_intersection
Fl_Tile, 1776
DRAW
Pen and tablet event handling, 268
draw
Fl_Adjuster, 453
Fl_Anim_GIF_Image, 463
Fl_Bitmap, 474
Fl_Box, 487
Fl_Browser_, 532
Fl_Button, 553
Fl_Cairo_Window, 569
Fl_Chart, 579
Fl_Choice, 617
Fl_Clock_Output, 636
Fl_Counter, 664
Fl_Dial, 675, 676
Fl_File_Icon, 721
Fl_File_Input, 734
Fl_Flex, 765
Fl_FormsBitmap, 787
Fl_FormsPixmap, 795
Fl_FormsText, 802
Fl_Free, 810
Fl_Gl_Window, 831
Fl_Glut_Window, 853
Fl_Grid, 869
Fl_Group, 888
Fl_Help_View, 909
Fl_Image, 972, 973
Fl_Input, 995
Fl_Input_Choice, 1034
Fl_Label, 1051
Fl_Light_Button, 1059
Fl_Menu_Bar, 1102
Fl_Menu_Button, 1114
Fl_Menu_Item, 1122
Fl_Pack, 1230
Fl_Pixmap, 1248
Fl_Positioner, 1270
Fl_Progress, 1315
Fl_Return_Button, 1358

Fl_RGB_Image, 1365
Fl_Roller, 1375
Fl_Scroll, 1416
Fl_Scrollbar, 1430
Fl_Shared_Image, 1464
Fl_Shortcut_Button, 1475
Fl_Slider, 1508
Fl_Spinner, 1521
Fl_SVG_Image, 1547
Fl_Sys_Menu_Bar, 1561
Fl_Table, 1583
Fl_Tabs, 1621
Fl_Terminal, 1657
Fl_Text_Display, 1713
Fl_Tiled_Image, 1786
Fl_Timer, 1802
Fl_Tree, 1842
Fl_Tree_Item, 1874
Fl_Value_Input, 1915
Fl_Value_Output, 1926
Fl_Value_Slider, 1937
Fl_Widget, 1963
Fl_Widget_Surface, 1984
Fl_Window, 2007
Fl_Wizard, 2030
draw_barchart
Fl_Chart, 579
draw_begin
Fl_Gl_Window, 832
draw_box_active
Fl, 424
draw_buff
Fl_Terminal, 1657
draw_cell
Fl_Table, 1583
draw_child
Fl_Group, 888
draw_children
Fl_Group, 888
draw_cursor
Fl_Text_Display, 1713
draw_decorated_window
Fl_Widget_Surface, 1984
draw_empty
Fl_Image, 973
draw_end
Fl_Gl_Window, 832
draw_filler_
Fl_Pack, 1230
draw_focus
Fl_Widget, 1963
draw_GL_text_with_textures
Fl, 424
draw_grid
Fl_Grid, 869
draw_horbarchart
Fl_Chart, 580
draw_horizontal_connector
Fl_Tree_Item, 1874
draw_item_content
Fl_Tree_Item, 1875
draw_label
Fl_Widget, 1964
draw_line_numbers
Fl_Text_Display, 1713
draw_linechart
Fl_Chart, 580
draw_overlay
Fl_Glut_Window, 853
Fl_Overlay_Window, 1219
draw_piechart
Fl_Chart, 581
draw_range
Fl_Text_Display, 1713
draw_row
Fl_Terminal, 1657
draw_row_bg
Fl_Terminal, 1657
draw_scaled
Fl_Image, 973
draw_string
Fl_Text_Display, 1714
draw_tab
Fl_Tabs, 1621
draw_text
Fl_Text_Display, 1714
draw_ticks
Fl_Slider, 1508
draw_vertical_connector
Fl_Tree_Item, 1876
draw_vline
Fl_Text_Display, 1714
drawbgcolor
Fl_Tree_Item, 1876
drawfgcolor
Fl_Tree_Item, 1876
Drawing functions, 307
fl_add_symbol, 313
fl_antialias, 314
fl_arc, 314, 315
fl_begin_complex_polygon, 315
fl_begin_offscreen, 315
fl_begin_points, 316
fl_can_do_alpha_blending, 316
FL_CAP_FLAT, 312
FL_CAP_ROUND, 312
FL_CAP_SQUARE, 312
fl_capture_window, 316
fl_circle, 316
fl_clip_box, 317
fl_clip_region, 317, 318
fl_copy_offscreen, 318
fl_create_offscreen, 318
fl_cursor, 319
fl_curve, 319
FL_DASH, 312

FL_DASHDOT, 312
FL_DASHDOTDOT, 312
fl_delete_offscreen, 320
FL_DOT, 312
fl_draw, 320, 321
fl_draw_arrow, 321
fl_draw_box, 322
fl_draw_box_focus, 322
fl_draw_check, 322
fl_draw_circle, 323
fl_draw_image, 323, 324
fl_draw_image_mono, 325
fl_draw_pixmap, 325
fl_draw_radio, 326
fl_draw_symbol, 326
fl_expand_text, 327
fl_focus_rect, 327
fl_frame, 327
fl_frame2, 328
fl_gap, 328
FL_JOIN_BEVEL, 312
FL_JOIN_MITER, 312
FL_JOIN_ROUND, 312
fl_line_style, 328
fl_load_matrix, 329
fl_measure, 330
fl_measure_pixmap, 330
fl_mult_matrix, 331
fl_not_clipped, 331
fl_old_shortcut, 331
fl_overlay_clear, 332
fl_overlay_rect, 332
fl_override_scale, 333
fl_pie, 333
fl_polygon, 334
fl_pop_clip, 334
fl_push_clip, 334
fl_push_matrix, 334
fl_read_image, 335
fl_rect, 335
fl_rectf, 336
fl_remove_symbol, 336
fl_rescale_offscreen, 337
fl_reset_spot, 337
fl_restore_scale, 337
fl_rotate, 337
fl_rounded_rect, 338
fl_rounded_rectf, 338
fl_scale, 338
fl_scroll, 338
fl_set_spot, 339
fl_set_status, 339
fl_shortcut_label, 339, 340
FL_SOLID, 312
fl_transform_dx, 341
fl_transform_dy, 341
fl_transform_x, 341
fl_transform_y, 341
fl_transformed_vertex, 341
fl_translate, 342
FL_UNIFORM_WIDTH, 312
fl_vertex, 342
Drawing Things in FLTK, 53
drawtext
Fl_Input_, 1008, 1009
DRIVER_AVAILABLE
Pen and tablet event handling, 269
driver_traits
Pen and tablet event handling, 269
dvalue
Fl_Input_, 1009

elapse_timeouts
Fl_Timeout, 1790
empty_vlines
Fl_Text_Display, 1715
enable
Fl_Tooltip, 1813
enable_im
Events handling functions, 254
enabled
Fl_Tooltip, 1813
end
Fl_Flex, 765
Fl_Group, 888
Fl_Text_Selection, 1763
end_current
Fl_PostScript_File_Device, 1276
Fl_Surface_Device, 1535
end_job
Fl_Paged_Device, 1235
Fl_PDF_File_Surface, 1241
Fl_PostScript_File_Device, 1276
Fl_Printer, 1305
end_page
Fl_Paged_Device, 1235
Fl_PDF_File_Surface, 1241
Fl_PostScript_File_Device, 1276
Fl_Printer, 1305
ENTER
Pen and tablet event handling, 268
enter_area
Fl_Tooltip, 1814
entries
Fl_Preferences, 1289
entry
Fl_Preferences, 1289
entry_exists
Fl_Preferences, 1289
Enumerations.H, 2062, 2085
_FL_EMBOSSED_LABEL, 2082
_FL_ENGRAVED_LABEL, 2082
_FL_ICON_LABEL, 2082
_FL_IMAGE_LABEL, 2082
_FL_MULTI_LABEL, 2082
_FL_SHADOW_LABEL, 2082
FL_ACTIVATE, 2080

FL_ALIGN_LEFT, 2085
FL_ALIGN_TOP, 2085
FL_APP_ACTIVATE, 2082
FL_APP_DEACTIVATE, 2082
FL_ARROW_CHOICE, 2075
FL_ARROW_DOUBLE, 2075
FL_ARROW_RETURN, 2075
FL_ARROW_SINGLE, 2075
Fl_Arrow_Type, 2074
FL_BEFORE_MENU, 2081
FL_BEFORE_TOOLTIP, 2081
FL_BORDER_BOX, 2075
FL_BORDER_FRAME, 2075
fl_box, 2084
Fl_Boxtype, 2075
Fl_Callback_Reason, 2076
FL_CLOSE, 2080
fl_color_cube, 2084
FL_CONTRAST_CIELAB, 2077
FL_CONTRAST_CUSTOM, 2077
Fl_Contrast_Function, 2074
FL_CONTRAST_LAST, 2077
FL_CONTRAST_LEGACY, 2077
Fl_Contrast_Mode, 2077
FL_CONTRAST_NONE, 2077
Fl_Cursor, 2077
FL_CURSOR_ARROW, 2077
FL_CURSOR_CROSS, 2077
FL_CURSOR_DEFAULT, 2077
FL_CURSOR_E, 2078
FL_CURSOR_HAND, 2077
FL_CURSOR_HELP, 2077
FL_CURSOR_INSERT, 2077
FL_CURSOR_MOVE, 2077
FL_CURSOR_N, 2078
FL_CURSOR_NE, 2078
FL_CURSOR_NESW, 2078
FL_CURSOR_NONE, 2078
FL_CURSOR_NS, 2077
FL_CURSOR_NW, 2078
FL_CURSOR_NWSE, 2078
FL_CURSOR_S, 2078
FL_CURSOR_SE, 2078
FL_CURSOR_SW, 2078
FL_CURSOR_W, 2078
FL_CURSOR_WAIT, 2077
FL_CURSOR_WE, 2078
Fl_Damage, 2078
FL_DAMAGE_ALL, 2078
FL_DAMAGE_CHILD, 2078
FL_DAMAGE_EXPOSE, 2078
FL_DAMAGE_OVERLAY, 2078
FL_DAMAGE_SCROLL, 2078
FL_DAMAGE_USER1, 2078
FL_DAMAGE_USER2, 2078
FL_DEACTIVATE, 2080
fl_define_FL_EMBOSSED_LABEL, 2084
fl_define_FL_ENGRAVED_LABEL, 2084
fl_define_FL_ICON_LABEL, 2084
fl_define_FL_IMAGE_LABEL, 2084
fl_define_FL_MULTI_LABEL, 2084
fl_define_FL_SHADOW_LABEL, 2084
FL_DIAMOND_DOWN_BOX, 2075
FL_DIAMOND_UP_BOX, 2075
FL_DND_DRAG, 2081
FL_DND_ENTER, 2081
FL_DND_LEAVE, 2081
FL_DND_RELEASE, 2081
fl_down, 2084
FL_DOWN_BOX, 2075
FL_DOWN_FRAME, 2075
FL_DRAG, 2079
FL_EMBOSSED_BOX, 2075
FL_EMBOSSED_FRAME, 2075
FL_ENGRAVED_BOX, 2075
FL_ENGRAVED_FRAME, 2075
FL_ENTER, 2079
Fl_Event, 2078
FL_EXCEPT, 2074
FL_FLAT_BOX, 2075
FL_FOCUS, 2079
Fl_Fontsize, 2074
fl_frame, 2085
FL_FREE_BOXTYPE, 2076
FL_FREE_LABELTYPE, 2082
FL_FULLSCREEN, 2081
FL_GLEAM_DOWN_BOX, 2076
FL_GLEAM_DOWN_FRAME, 2076
FL_GLEAM_ROUND_DOWN_BOX, 2076
FL_GLEAM_ROUND_UP_BOX, 2076
FL_GLEAM_THIN_DOWN_BOX, 2076
FL_GLEAM_THIN_UP_BOX, 2076
FL_GLEAM_UP_BOX, 2076
FL_GLEAM_UP_FRAME, 2076
fl_gray_ramp, 2085
FL_GTK_DOWN_BOX, 2076
FL_GTK_DOWN_FRAME, 2076
FL_GTK_ROUND_DOWN_BOX, 2076
FL_GTK_ROUND_UP_BOX, 2076
FL_GTK_THIN_DOWN_BOX, 2076
FL_GTK_THIN_DOWN_FRAME, 2076
FL_GTK_THIN_UP_BOX, 2076
FL_GTK_THIN_UP_FRAME, 2076
FL_GTK_UP_BOX, 2076
FL_GTK_UP_FRAME, 2076
FL_HIDE, 2081
FL_IMAGE_LABEL, 2073
FL_KEYBOARD, 2080
FL_KEYDOWN, 2080
FL_KEYUP, 2080
Fl_Labeltype, 2082
FL_LEAVE, 2079
FL_MAX_BOXTYPE, 2076
FL_MOUSEWHEEL, 2081
FL_MOVE, 2080
FL_MULTI_LABEL, 2073

FL_NO_BOX, 2075
FL_NO_EVENT, 2078
FL_NO_LABEL, 2082
FL_NORMAL_LABEL, 2082
FL_NORMAL_SIZE, 2085
FL_OFLAT_BOX, 2075
FL_ORIENT_DOWN, 2083
FL_ORIENT_LEFT, 2083
FL_ORIENT_NE, 2083
FL_ORIENT_NONE, 2083
FL_ORIENT_NW, 2083
FL_ORIENT_RIGHT, 2083
FL_ORIENT_SE, 2083
FL_ORIENT_SW, 2083
FL_ORIENT_UP, 2083
Fl_Orientation, 2082
FL_OSHADOW_BOX, 2075
FL_OVAL_BOX, 2075
FL_OVAL_FRAME, 2075
FL_OXY_BUTTON_DOWN_BOX, 2076
FL_OXY_BUTTON_UP_BOX, 2076
FL_OXY_DOWN_BOX, 2076
FL_OXY_DOWN_FRAME, 2076
FL_OXY_ROUND_DOWN_BOX, 2076
FL_OXY_ROUND_UP_BOX, 2076
FL_OXY_THIN_DOWN_BOX, 2076
FL_OXY_THIN_DOWN_FRAME, 2076
FL_OXY_THIN_UP_BOX, 2076
FL_OXY_THIN_UP_FRAME, 2076
FL_OXY_UP_BOX, 2076
FL_OXY_UP_FRAME, 2076
FL_PASTE, 2081
FL_PLASTIC_DOWN_BOX, 2076
FL_PLASTIC_DOWN_FRAME, 2076
FL_PLASTIC_ROUND_DOWN_BOX, 2076
FL_PLASTIC_ROUND_UP_BOX, 2076
FL_PLASTIC_THIN_DOWN_BOX, 2076
FL_PLASTIC_THIN_UP_BOX, 2076
FL_PLASTIC_UP_BOX, 2075
FL_PLASTIC_UP_FRAME, 2076
FL_PUSH, 2079
FL_READ, 2074
FL_REASON_CANCELLED, 2077
FL_REASON_CHANGED, 2077
FL_REASON_CLOSED, 2077
FL_REASON_DESELECTED, 2077
FL_REASON_DRAGGED, 2077
FL_REASON_ENTER_KEY, 2077
FL_REASON_GOT_FOCUS, 2077
FL_REASON_LOST_FOCUS, 2077
FL_REASON_OPENED, 2077
FL_REASON_RELEASED, 2077
FL_REASON_RESELECTED, 2077
FL_REASON_SELECTED, 2077
FL_REASON_UNKNOWN, 2077
FL_REASON_USER, 2077
FL_RELEASE, 2079
FL_RFLAT_BOX, 2075
FL_ROUND_DOWN_BOX, 2075
FL_ROUND_UP_BOX, 2075
FL_ROUNDED_BOX, 2075
FL_ROUNDED_FRAME, 2075
FL_RSHADOW_BOX, 2075
FL_SCREEN_CONFIGURATION_CHANGED, 2081
FL_SELECTIONCLEAR, 2081
FL_SHADOW_BOX, 2075
FL_SHADOW_FRAME, 2075
FL_SHORTCUT, 2080
FL_SHOW, 2081
FL_SYMBOL_LABEL, 2073
FL_THIN_DOWN_BOX, 2075
FL_THIN_DOWN_FRAME, 2075
FL_THIN_UP_BOX, 2075
FL_THIN_UP_FRAME, 2075
FL_UNFOCUS, 2079
FL_UP_BOX, 2075
FL_UP_FRAME, 2075
Fl_When, 2083
FL_WHEN_CHANGED, 2083
FL_WHEN_CLOSED, 2083
FL_WHEN_ENTER_KEY, 2083
FL_WHEN_ENTER_KEY_ALWAYS, 2083
FL_WHEN_ENTER_KEY_CHANGED, 2083
FL_WHEN_NEVER, 2083
FL_WHEN_NOT_CHANGED, 2083
FL_WHEN_RELEASE, 2083
FL_WHEN_RELEASE_ALWAYS, 2083
FL_WRITE, 2074
FL_ZOOM_EVENT, 2081
FL_ZOOM_GESTURE, 2081
EOL
Fl_Terminal, 1650
ERASER
Pen and tablet event handling, 269
ERASER_DOWN
Pen and tablet event handling, 268
ERASER_HOVERS
Pen and tablet event handling, 268
errmsg
Fl_File_Browser, 710
Fl_Native_File_Chooser, 1183
error
Common Dialog Classes and Functions, 392
error_char
Fl_Terminal, 1657, 1658
ERRORS_TO_CP1252
Unicode and UTF-8 functions, 350
ERRORS_TO_ISO8859_1
Unicode and UTF-8 functions, 350
estimate_lines
Fl_Text_Buffer, 1681
Event
Pen and tablet event handling, 267
event
Events handling functions, 255

event_barrel_pressure
Pen and tablet event handling, 270
event_button
Events handling functions, 255
event_button1
Events handling functions, 255
event_button2
Events handling functions, 255
event_button3
Events handling functions, 255
event_button4
Events handling functions, 255
event_button5
Events handling functions, 255
event_buttons
Events handling functions, 256
event_clicks
Events handling functions, 256
event_clipboard
Events handling functions, 256
event_clipboard_type
Events handling functions, 256
event_dispatch
Events handling functions, 256
event_dx
Events handling functions, 257
event_dx_f
Events handling functions, 257
event_dy
Events handling functions, 257
event_dy_f
Events handling functions, 257
event_inside
Events handling functions, 257, 258
event_is_click
Events handling functions, 258
event_key
Events handling functions, 258, 259
event_length
Events handling functions, 259
event_original_key
Events handling functions, 259
event_pen_id
Pen and tablet event handling, 270
event_pressure
Pen and tablet event handling, 270
event_proximity
Pen and tablet event handling, 270
event_state
Events handling functions, 259, 260
Pen and tablet event handling, 270, 271
event_text
Events handling functions, 260
event_tilt_x
Pen and tablet event handling, 271
event_trigger
Pen and tablet event handling, 271
event_twist
Pen and tablet event handling, 271
event_x
Pen and tablet event handling, 272
event_x_root
Events handling functions, 261
Pen and tablet event handling, 272
event_y_root
Events handling functions, 261
Events handling functions, 249
add_handler, 252
add_system_handler, 253
belowmouse, 253
callback_reason, 253
compose, 254
compose_reset, 254
disable_im, 254
enable_im, 254
event, 255
event_button, 255
event_button1, 255
event_button2, 255
event_button3, 255
event_button4, 255
event_button5, 255
event_buttons, 256
event_clicks, 256
event_clipboard, 256
event_clipboard_type, 256
event_dispatch, 256
event_dx, 257
event_dx_f, 257
event_dy, 257
event_dy_f, 257
event_inside, 257, 258
event_is_click, 258
event_key, 258, 259
event_length, 259
event_original_key, 259
event_state, 259, 260
event_text, 260
event_x_root, 261
event_y_root, 261
fl_callback_reason_names, 265
fl_callback_reason_str, 261
fl_eventname_str, 261
fl_fontname_str, 261
fl_fontnames, 265
focus, 262
get_key, 262
get_mouse, 262
handle, 263
handle_, 263
last_handler, 264
pushed, 264
remove_handler, 264
remove_system_handler, 264
test_shortcut, 264
events.H, 2048, 2051

Example Source Code, 154
EXECUTIVE
    Fl_Paged_Device, 1234
exists
    Fl_Widget_Tracker, 1987
extend_range_for_styles
    Fl_Text_Display, 1715
extend_selection
    Fl_Tree, 1842
extend_selection_dir
    Fl_Tree, 1842

fail
    Fl_Image, 973
FAQ (Frequently Asked Questions), 164
fastarrow.h, 2357
fatal
    Common Dialog Classes and Functions, 392
FG_XTERM
    Fl_Terminal, 1650
File names and URI utility functions, 393
    fl_decode_uri, 395
    Fl_File_Sort_F, 394
    fl_filename_absolute, 395
    fl_filename_absolute_str, 396
    fl_filename_expand, 396
    fl_filename_expand_str, 397
    fl_filename_ext, 397
    fl_filename_ext_str, 397
    fl_filename_free_list, 398
    fl_filename_isdir, 398
    fl_filename_list, 398
    fl_filename_match, 399
    fl_filename_name, 400
    fl_filename_name_str, 400
    fl_filename_path_str, 400
    fl_filename_relative, 401
    fl_filename_relative_str, 402
    fl_filename_setext, 402
    fl_filename_setext_str, 403
    fl_getcwd_str, 403
    fl_open_uri, 403
file_access
    Fl_Preferences, 1289, 1290
file_encoding_warning_message
    Fl_Text_Buffer, 1690
filename
    Fl_Help_View, 909
    Fl_Native_File_Chooser, 1183
    Fl_Preferences, 1290
filename.H, 2093, 2095
filetype
    Fl_File_Browser, 710
filter
    Fl_File_Browser, 710
    Fl_File_Chooser, 717
    Fl_Native_File_Chooser, 1183, 1184
filter_value
    Fl_Native_File_Chooser, 1184
find
    Fl_File_Icon, 721
    Fl_Group, 888
    Fl_Help_View, 909
    Fl_Shared_Image, 1464
find_cell
    Fl_Table, 1584
find_child
    Fl_Tree_Item, 1876
find_child_item
    Fl_Tree_Item, 1877
find_clicked
    Fl_Tree, 1843
    Fl_Tree_Item, 1877
find_index
    Fl_Menu_, 1082, 1083
find_item
    Fl_Browser_, 532
    Fl_Menu_, 1084
    Fl_Tree, 1843
    Fl_Tree_Item, 1877
find_item_with_argument
    Fl_Menu_, 1084
find_item_with_user_data
    Fl_Menu_, 1085
find_line
    Fl_Browser, 505
find_line_end
    Fl_Text_Display, 1715
find_shortcut
    Fl_Menu_Item, 1122
find_wrap_range
    Fl_Text_Display, 1715
find_x
    Fl_Text_Display, 1716
findchar_backward
    Fl_Text_Buffer, 1681
findchar_forward
    Fl_Text_Buffer, 1682
first
    Fl_Tree, 1844
first_selected_item
    Fl_Tree, 1844
first_timeout
    Fl_Timeout, 1794
first_visible
    Fl_Tree, 1844
first_visible_item
    Fl_Tree, 1845
first_window
    Windows handling functions, 278
fix_scrollbar_order
    Fl_Scroll, 1417
fixed
    Fl_Flex, 765, 766
Fl, 405
    abi_check, 416
    abi_version, 416

add_check, 416
add_fd, 416
add_idle, 417
add_timeout, 417
api_version, 418
arg, 418
args, 419
args_to_utf8, 420
background, 421
background2, 421
box_bg, 421
box_border_radius_max, 422
box_color, 422
box_dh, 422
box_dw, 422
box_dx, 422
box_dy, 423
box_shadow_width, 423
check, 423
display, 423
dnd_text_ops, 424
draw_box_active, 424
draw_GL_text_with_textures, 424
Fl_Option, 415
flush, 425
get_system_colors, 425
gl_visual, 425
has_idle, 425
has_timeout, 426
help, 438
hide_all_windows, 426
idle, 426
is_scheme, 427
menu_linespacing, 427
now, 428
option, 428, 429
OPTION_ARROW_FOCUS, 415
OPTION_DND_TEXT, 415
OPTION_FNFC_USES_GTK, 415
OPTION_FNFC_USES_KDIALOG, 415
OPTION_FNFC_USES_ZENITY, 415
OPTION_LAST, 415
OPTION_PRINTER_USES_GTK, 415
OPTION_SHOW_SCALING, 415
OPTION_SHOW_TOOLTIPS, 415
OPTION_SIMPLE_ZOOM_SHORTCUT, 415
OPTION_VISIBLE_FOCUS, 415
own_colormap, 430
program_should_quit, 430
readqueue, 430
ready, 430
reload_scheme, 431
remove_check, 431
remove_idle, 431
remove_next_timeout, 431
remove_timeout, 432
repeat_timeout, 433
run, 433
scheme, 433
scrollbar_size, 434
seconds_between, 434
seconds_since, 435
set_box_color, 435
set_boxtype, 435
ticks_between, 436
ticks_since, 436
timeout_list, 437
use_high_res_GL, 437
version, 437
visible_focus, 437
visual, 437
wait, 438
Fl.cxx, 2357
fl_close_display, 2358
fl_disable_wayland, 2359
fl_find, 2358
fl_open_display, 2358
Fl.H, 2096, 2104
Fl::Pen, 438
Fl::TimeoutData, 2044
FL_ABI_VERSION
Version Numbers, 247
fl_access
Unicode and UTF-8 functions, 350
FL_ACTIVATE
Enumerations.H, 2080
fl_add_symbol
Drawing functions, 313
Fl_Adjuster, 444
draw, 453
Fl_Adjuster, 453
handle, 453
soft, 454
value_damage, 454
Fl_Adjuster.H, 2108
fl_alert
Common Dialog Classes and Functions, 378
FL_ALIGN_LEFT
Enumerations.H, 2085
FL_ALIGN_TOP
Enumerations.H, 2085
Fl_Anim_GIF_Image, 454
~Fl_Anim_GIF_Image, 461
canvas, 461
canvas_h, 462
canvas_w, 462
color_average, 462
copy, 462
DEBUG_FLAG, 460
delay, 463
desaturate, 463
DONT_RESIZE_CANVAS, 460
DONT_SET_AS_IMAGE, 460
DONT_START, 460
draw, 463
Fl_Anim_GIF_Image, 460, 461

Flags, 460
frame, 464
frame_count, 464
frame_h, 464
frame_uncache, 465
frame_w, 465
frame_x, 465
frame_y, 466
frames, 466
image, 466
is_animated, 466
load, 467
LOG_FLAG, 460
loop, 469
min_delay, 469
name, 467
next, 467
on_extension_data, 467
on_frame_data, 467
OPTIMIZE_MEMORY, 460
playing, 467
resize, 468
speed, 468
start, 468
stop, 469
uncache, 469
valid, 469
Fl_Anim_GIF_Image.H, 2108
fl_antialias
Drawing functions, 314
FL_API_VERSION
Version Numbers, 248
FL_APP_ACTIVATE
Enumerations.H, 2082
FL_APP_DEACTIVATE
Enumerations.H, 2082
fl_arc
Drawing functions, 314, 315
fl_arc.cxx, 2359
FL_ARROW_CHOICE
Enumerations.H, 2075
FL_ARROW_DOUBLE
Enumerations.H, 2075
FL_ARROW_RETURN
Enumerations.H, 2075
FL_ARROW_SINGLE
Enumerations.H, 2075
Fl_Arrow_Type
Enumerations.H, 2074
fl_ask.cxx, 2359
fl_ask.H, 2110, 2112
Fl_Beep, 2112
FL_BEEP_DEFAULT, 2112
FL_BEEP_ERROR, 2112
FL_BEEP_MESSAGE, 2112
FL_BEEP_NOTIFICATION, 2112
FL_BEEP_PASSWORD, 2112
FL_BEEP_QUESTION, 2112
fl_message_position, 2112
fl_attr.h, 2114, 2115
__fl_attr, 2114
FL_DEPRECATED, 2114
FL_OVERRIDE, 2114
Fl_Beep
fl_ask.H, 2112
fl_beep
Common Dialog Classes and Functions, 379
FL_BEEP_DEFAULT
fl_ask.H, 2112
FL_BEEP_ERROR
fl_ask.H, 2112
FL_BEEP_MESSAGE
fl_ask.H, 2112
FL_BEEP_NOTIFICATION
fl_ask.H, 2112
FL_BEEP_PASSWORD
fl_ask.H, 2112
FL_BEEP_QUESTION
fl_ask.H, 2112
FL_BEFORE_MENU
Enumerations.H, 2081
FL_BEFORE_TOOLTIP
Enumerations.H, 2081
fl_begin_complex_polygon
Drawing functions, 315
fl_begin_offscreen
Drawing functions, 315
fl_begin_points
Drawing functions, 316
Fl_Bitmap, 469
copy, 473
draw, 474
Fl_Bitmap, 472, 473
label, 474
uncache, 475
Fl_Bitmap.H, 2117
Fl_BMP_Image, 475
Fl_BMP_Image, 478
Fl_BMP_Image.H, 2118
FL_BORDER_BOX
Enumerations.H, 2075
FL_BORDER_FRAME
Enumerations.H, 2075
Fl_Box, 479
draw, 487
Fl_Box, 486
handle, 487
fl_box
Enumerations.H, 2084
Fl_Box.H, 2118
Fl_Boxtype
Enumerations.H, 2075
fl_boxtype.cxx, 2361
fl_rectbound, 2363
Fl_Browser, 488
_remove, 502

add, 502
bline_length, 503
bottomline, 503
clear, 503
column_char, 503
column_widths, 504
data, 504, 505
display, 505
displayed, 505
find_line, 505
Fl_Browser, 502
format_char, 506
full_height, 507
hide, 507
icon, 507, 508
incr_height, 508
insert, 508
item_at, 509
item_draw, 509
item_first, 509
item_height, 510
item_last, 510
item_next, 510
item_prev, 511
item_select, 511
item_selected, 511
item_swap, 512
item_text, 512
item_width, 512
lineno, 513
lineposition, 513
load, 513
make_visible, 514
middleline, 514
move, 514
remove, 514
remove_icon, 515
select, 515
selected, 515
show, 516
size, 516
swap, 516
text, 517
textsize, 517
topline, 517, 518
value, 518
visible, 518
Fl_Browser.H, 2119
Fl_Browser_, 519
ALWAYS_ON, 530
bbox, 531
BOTH, 530
BOTH_ALWAYS, 530
deleting, 531
deselect, 531
display, 532
displayed, 532
draw, 532
find_item, 532
Fl_Browser_, 531
full_height, 532
full_width, 533
handle, 533
has_scrollbar, 533
HORIZONTAL, 530
HORIZONTAL_ALWAYS, 530
hposition, 533, 534
hscrollbar, 542
incr_height, 534
inserting, 534
item_at, 534
item_draw, 535
item_first, 535
item_height, 535
item_last, 535
item_next, 535
item_prev, 536
item_quick_height, 536
item_select, 536
item_selected, 536
item_swap, 537
item_text, 537
item_width, 537
leftedge, 538
linespacing, 538
new_list, 538
position, 538
redraw_line, 538
redraw_lines, 539
replacing, 539
resize, 539
scrollbar, 542
scrollbar_left, 539
scrollbar_right, 540
scrollbar_size, 540
select, 540
select_only, 541
selection, 541
sort, 541
swapping, 541
textfont, 541
VERTICAL, 530
VERTICAL_ALWAYS, 530
vposition, 542
Fl_Browser_.H, 2121
Fl_Button, 543
clear, 551
compact, 551
down_box, 552
draw, 553
Fl_Button, 550
handle, 553
set, 553
shortcut, 554
value, 554
Fl_Button.H, 2122

Fl_Cairo.H, 2124
Fl_Cairo_State, 554
cc, 555
Fl_Cairo_Window, 555
draw, 569
set_draw_cb, 569
Fl_Cairo_Window.H, 2125
fl_callback_macros.H, 2126, 2129
FL_FUNCTION_CALLBACK_3, 2126
FL_INLINE_CALLBACK_2, 2127
FL_METHOD_CALLBACK_1, 2128
Fl_Callback_Reason
Enumerations.H, 2076
fl_callback_reason_names
Events handling functions, 265
fl_callback_reason_str
Events handling functions, 261
Fl_Callback_User_Data, 569
fl_can_do_alpha_blending
Drawing functions, 316
FL_CAP_FLAT
Drawing functions, 312
FL_CAP_ROUND
Drawing functions, 312
FL_CAP_SQUARE
Drawing functions, 312
fl_capture_window
Drawing functions, 316
fl_casenumericsort
numericsort.c, 2404
fl_casts.H, 2134
Fl_Chart, 570
add, 578
autosize, 578
bounds, 579
draw, 579
draw_barchart, 579
draw_horbarchart, 580
draw_linechart, 580
draw_piechart, 581
Fl_Chart, 578
insert, 581
maxsize, 581
replace, 581
size, 582
Fl_Chart.H, 2135
FL_CHART_ENTRY, 582
fl_chdir
Unicode and UTF-8 functions, 350
Fl_Check_Browser, 583
add, 594
handle, 594
item_at, 595
item_draw, 595
item_first, 595
item_height, 595
item_next, 596
item_prev, 596
item_select, 596
item_selected, 596
item_swap, 597
item_text, 597
item_width, 597
nitems, 597
remove, 598
Fl_Check_Browser.H, 2137
Fl_Check_Button, 598
Fl_Check_Button, 606
Fl_Check_Button.H, 2138
fl_chmod
Unicode and UTF-8 functions, 351
Fl_Choice, 606
draw, 617
Fl_Choice, 616
handle, 617
value, 617, 618
fl_choice
Common Dialog Classes and Functions, 379
Fl_Choice.H, 2138
fl_choice_n
Common Dialog Classes and Functions, 380
fl_circle
Drawing functions, 316
fl_clip_box
Drawing functions, 317
fl_clip_region
Drawing functions, 317, 318
Fl_Clock, 618
Fl_Clock, 626
handle, 627
Fl_Clock.H, 2139
Fl_Clock_Output, 627
draw, 636
Fl_Clock_Output, 635
hour, 636
minute, 636
second, 636
shadow, 636, 637
value, 637
FL_CLOSE
Enumerations.H, 2080
fl_close_display
Fl.cxx, 2358
fl_close_fd
Unicode and UTF-8 functions, 351
fl_cmap
fl_color.cxx, 2367
fl_cmap.h, 2364
fl_color
Color & Font functions, 293, 294
fl_color.cxx, 2367
fl_cmap, 2367
fl_color_average
Color & Font functions, 294
Fl_Color_Chooser, 638
b, 648

Fl_Color_Chooser, 647
g, 648
handle, 648
hsv, 648
hsv2rgb, 649
hue, 649
mode, 649
r, 650
rgb, 650
rgb2hsv, 650
saturation, 650
value, 650
fl_color_chooser
Common Dialog Classes and Functions, 381, 382
Fl_Color_Chooser.H, 2140
fl_color_cube
Enumerations.H, 2084
Fl_compose.cxx, 2367
fl_config.h, 2047
fl_contrast
Color & Font functions, 294
fl_contrast.cxx, 2368
FL_CONTRAST_CIELAB
Enumerations.H, 2077
FL_CONTRAST_CUSTOM
Enumerations.H, 2077
Fl_Contrast_Function
Enumerations.H, 2074
fl_contrast_function
Color & Font functions, 296
FL_CONTRAST_LAST
Enumerations.H, 2077
FL_CONTRAST_LEGACY
Enumerations.H, 2077
fl_contrast_level
Color & Font functions, 296
Fl_Contrast_Mode
Enumerations.H, 2077
fl_contrast_mode
Color & Font functions, 297
FL_CONTRAST_NONE
Enumerations.H, 2077
fl_copy_offscreen
Drawing functions, 318
Fl_Copy_Surface, 651
Fl_Copy_Surface, 653
is_current, 653
origin, 653
printable_rect, 654
set_current, 654
translate, 654
untranslate, 654
Fl_Copy_Surface.H, 2142
Fl_Counter, 655
arrow_widths, 664
draw, 664
Fl_Counter, 663
handle, 664
lstep, 665
step, 665
Fl_Counter.H, 2142
fl_create_offscreen
Drawing functions, 318
Fl_Cursor
Enumerations.H, 2077
fl_cursor
Drawing functions, 319
FL_CURSOR_ARROW
Enumerations.H, 2077
FL_CURSOR_CROSS
Enumerations.H, 2077
FL_CURSOR_DEFAULT
Enumerations.H, 2077
FL_CURSOR_E
Enumerations.H, 2078
FL_CURSOR_HAND
Enumerations.H, 2077
FL_CURSOR_HELP
Enumerations.H, 2077
FL_CURSOR_INSERT
Enumerations.H, 2077
FL_CURSOR_MOVE
Enumerations.H, 2077
FL_CURSOR_N
Enumerations.H, 2078
FL_CURSOR_NE
Enumerations.H, 2078
FL_CURSOR_NESW
Enumerations.H, 2078
FL_CURSOR_NONE
Enumerations.H, 2078
FL_CURSOR_NS
Enumerations.H, 2077
FL_CURSOR_NW
Enumerations.H, 2078
FL_CURSOR_NWSE
Enumerations.H, 2078
FL_CURSOR_S
Enumerations.H, 2078
FL_CURSOR_SE
Enumerations.H, 2078
FL_CURSOR_SW
Enumerations.H, 2078
FL_CURSOR_W
Enumerations.H, 2078
FL_CURSOR_WAIT
Enumerations.H, 2077
FL_CURSOR_WE
Enumerations.H, 2078
fl_curve
Drawing functions, 319
fl_curve.cxx, 2368
Fl_Damage
Enumerations.H, 2078
FL_DAMAGE_ALL
Enumerations.H, 2078

FL_DAMAGE_CHILD
Enumerations.H, 2078
FL_DAMAGE_EXPOSE
Enumerations.H, 2078
FL_DAMAGE_OVERLAY
Enumerations.H, 2078
FL_DAMAGE_SCROLL
Enumerations.H, 2078
FL_DAMAGE_USER1
Enumerations.H, 2078
FL_DAMAGE_USER2
Enumerations.H, 2078
FL_DASH
Drawing functions, 312
FL_DASHDOT
Drawing functions, 312
FL_DASHDOTDOT
Drawing functions, 312
FL_DEACTIVATE
Enumerations.H, 2080
fl_decode_uri
File names and URI utility functions, 395
fl_define_FL_EMBOSSED_LABEL
Enumerations.H, 2084
fl_define_FL_ENGRAVED_LABEL
Enumerations.H, 2084
fl_define_FL_ICON_LABEL
Enumerations.H, 2084
fl_define_FL_IMAGE_LABEL
Enumerations.H, 2084
fl_define_FL_MULTI_LABEL
Enumerations.H, 2084
fl_define_FL_SHADOW_LABEL
Enumerations.H, 2084
fl_delete_offscreen
Drawing functions, 320
FL_DEPRECATED
fl_attr.h, 2114
fl_descent
Color & Font functions, 298
Fl_Device.H, 2143, 2144
Fl_Device_Plugin, 665
rectangle_capture, 666
Fl_Dial, 667
angle1, 675
draw, 675, 676
Fl_Dial, 675
handle, 676
Fl_Dial.H, 2145
FL_DIAMOND_DOWN_BOX
Enumerations.H, 2075
FL_DIAMOND_UP_BOX
Enumerations.H, 2075
fl_dir_chooser
Common Dialog Classes and Functions, 383
fl_disable_wayland
Fl.cxx, 2359
Fl_Display_Device, 676
FL_DND_DRAG
Enumerations.H, 2081
FL_DND_ENTER
Enumerations.H, 2081
FL_DND_LEAVE
Enumerations.H, 2081
FL_DND_RELEASE
Enumerations.H, 2081
FL_DOT
Drawing functions, 312
Fl_Double_Window, 677
~Fl_Double_Window, 690
as_double_window, 690
flush, 690
hide, 690
resize, 690
show, 691
Fl_Double_Window.cxx, 2369
Fl_Double_Window.H, 2145
fl_down
Enumerations.H, 2084
FL_DOWN_BOX
Enumerations.H, 2075
FL_DOWN_FRAME
Enumerations.H, 2075
FL_DRAG
Enumerations.H, 2079
fl_draw
Drawing functions, 320, 321
fl_draw.H, 2146, 2152
fl_draw_arrow
Drawing functions, 321
fl_draw_box
Drawing functions, 322
fl_draw_box_focus
Drawing functions, 322
fl_draw_check
Drawing functions, 322
fl_draw_circle
Drawing functions, 323
fl_draw_image
Drawing functions, 323, 324
fl_draw_image_mono
Drawing functions, 325
fl_draw_pixmap
Drawing functions, 325
fl_draw_radio
Drawing functions, 326
fl_draw_symbol
Drawing functions, 326
FL_EMBOSSED_BOX
Enumerations.H, 2075
FL_EMBOSSED_FRAME
Enumerations.H, 2075
Fl_End, 691
FL_ENGRAVED_BOX
Enumerations.H, 2075
FL_ENGRAVED_FRAME

Enumerations.H, 2075
FL_ENTER
Enumerations.H, 2079
Fl_EPS_File_Surface, 692
~Fl_EPS_File_Surface, 694
close, 694
Fl_EPS_File_Surface, 694
origin, 694, 695
printable_rect, 695
translate, 695
untranslate, 695
Fl_Event
Enumerations.H, 2078
Fl_Event_Dispatch
Callback Function Typedefs, 277
fl_eventname_str
Events handling functions, 261
FL_EXCEPT
Enumerations.H, 2074
fl_execvp
Unicode and UTF-8 functions, 351
fl_expand_text
Drawing functions, 327
Fl_Export.H, 2157
Fl_File_Browser, 696
errmsg, 710
filetype, 710
filter, 710
Fl_File_Browser, 710
iconsize, 710
load, 711
Fl_File_Browser.H, 2158
Fl_File_Chooser, 711
add_extra, 716
CREATE, 716
DIRECTORY, 716
filter, 717
Fl_File_Chooser, 716
iconsize, 717
MULTI, 716
preview, 717
showHiddenButton, 718
shown, 717
SINGLE, 716
Type, 716
value, 717, 718
fl_file_chooser
Common Dialog Classes and Functions, 383
Fl_File_Chooser.H, 2159
fl_file_chooser_callback
Common Dialog Classes and Functions, 384
fl_file_chooser_ok_label
Common Dialog Classes and Functions, 384
Fl_File_Icon, 718
add, 720
add_color, 720
add_vertex, 720, 721
draw, 721
find, 721
Fl_File_Icon, 720
label, 721
labeltype, 721
load, 722
load_fti, 722
load_image, 722
load_system_icons, 723
next, 723
type, 723
Fl_File_Icon.H, 2161
Fl_File_Input, 723
down_box, 734
draw, 734
Fl_File_Input, 734
handle, 734
value, 734, 735
Fl_File_Input.H, 2162
Fl_File_Sort_F
File names and URI utility functions, 394
fl_filename_absolute
File names and URI utility functions, 395
fl_filename_absolute_str
File names and URI utility functions, 396
fl_filename_expand
File names and URI utility functions, 396
fl_filename_expand_str
File names and URI utility functions, 397
fl_filename_ext
File names and URI utility functions, 397
fl_filename_ext_str
File names and URI utility functions, 397
fl_filename_free_list
File names and URI utility functions, 398
fl_filename_isdir
File names and URI utility functions, 398
fl_filename_list
File names and URI utility functions, 398
fl_filename_match
File names and URI utility functions, 399
fl_filename_name
File names and URI utility functions, 400
fl_filename_name_str
File names and URI utility functions, 400
fl_filename_path_str
File names and URI utility functions, 400
fl_filename_relative
File names and URI utility functions, 401
fl_filename_relative_str
File names and URI utility functions, 402
fl_filename_setext
File names and URI utility functions, 402
fl_filename_setext_str
File names and URI utility functions, 403
Fl_Fill_Dial, 735
Fl_Fill_Dial.H, 2163
Fl_Fill_Slider, 743
Fl_Fill_Slider.H, 2164

fl_find
Fl.cxx, 2358
FL_FLAT_BOX
Enumerations.H, 2075
Fl_Flex, 752
alloc_size, 765
COLUMN, 763
draw, 765
end, 765
fixed, 765, 766
Fl_Flex, 763, 764
gap, 766
HORIZONTAL, 763
horizontal, 767
layout, 767
margin, 767, 768
need_layout, 769
on_remove, 769
resize, 769
ROW, 763
spacing, 770
VERTICAL, 763
Fl_Flex.H, 2164
Fl_Float_Input, 770
Fl_Float_Input, 780
Fl_Float_Input.H, 2166
FL_FOCUS
Enumerations.H, 2079
fl_focus_rect
Drawing functions, 327
fl_font
Color & Font functions, 299
fl_fontname_str
Events handling functions, 261
fl_fontnames
Events handling functions, 265
Fl_Fontsize
Enumerations.H, 2074
fl_fopen
Unicode and UTF-8 functions, 352
Fl_FormsBitmap, 780
draw, 787
set, 787
Fl_FormsBitmap.H, 2166
Fl_FormsPixmap, 788
draw, 795
Fl_FormsPixmap, 794
Pixmap, 795
set, 795
Fl_FormsPixmap.H, 2167
Fl_FormsText, 795
draw, 802
fl_frame
Drawing functions, 327
Enumerations.H, 2085
fl_frame2
Drawing functions, 328
Fl_Free, 802
draw, 810
Fl_Free, 809
handle, 810
Fl_Free.H, 2167
FL_FREE_BOXTYPE
Enumerations.H, 2076
FL_FREE_LABELTYPE
Enumerations.H, 2082
FL_FULLSCREEN
Enumerations.H, 2081
FL_FUNCTION_CALLBACK_3
fl_callback_macros.H, 2126
fl_gap
Drawing functions, 328
Fl_get_system_colors.cxx, 2369
fl_parse_color, 2370
fl_getcwd
Unicode and UTF-8 functions, 352
fl_getcwd_str
File names and URI utility functions, 403
fl_getenv
Unicode and UTF-8 functions, 353
Fl_GIF_Image, 811
animate, 815
Fl_GIF_Image, 814, 815
Fl_GIF_Image.H, 2168
Fl_GIF_Image::GIF_FRAME, 2036
Fl_GIF_Image::GIF_FRAME::CPAL, 443
Fl_Gl_Choice, 816
Fl_Gl_Choice.H, 2370
Fl_Gl_Window, 816
as_gl_window, 830
can_do, 831
can_do_overlay, 831
context, 831
context_valid, 831
draw, 831
draw_begin, 832
draw_end, 832
Fl_Gl_Window, 830
flush, 832
handle, 833
hide, 833
make_current, 833
make_overlay_current, 833
mode, 833
ortho, 834
pixel_h, 834
pixel_w, 835
pixels_per_unit, 835
redraw_overlay, 835
resize, 835
show, 836
swap_buffers, 836
swap_interval, 836
valid, 837
Fl_Gl_Window.H, 2169
Fl_Gl_Window_Driver.H, 2371

FL_GLEAM_DOWN_BOX
Enumerations.H, 2076
FL_GLEAM_DOWN_FRAME
Enumerations.H, 2076
FL_GLEAM_ROUND_DOWN_BOX
Enumerations.H, 2076
FL_GLEAM_ROUND_UP_BOX
Enumerations.H, 2076
FL_GLEAM_THIN_DOWN_BOX
Enumerations.H, 2076
FL_GLEAM_THIN_UP_BOX
Enumerations.H, 2076
FL_GLEAM_UP_BOX
Enumerations.H, 2076
FL_GLEAM_UP_FRAME
Enumerations.H, 2076
Fl_Glut_Bitmap_Font, 837
Fl_Glut_StrokeChar, 838
Fl_Glut_StrokeFont, 838
Fl_Glut_StrokeStrip, 838
Fl_Glut_StrokeVertex, 838
Fl_Glut_Window, 838
draw, 853
draw_overlay, 853
handle, 853
Fl_Graphics_Driver.cxx, 2372
Fl_Graphics_Driver.H, 2170
fl_gray_ramp
Enumerations.H, 2085
Fl_Grid, 854
cell, 865, 866
clear_layout, 866
col_gap, 866
col_weight, 867
col_width, 868
debug, 868
draw, 869
draw_grid, 869
Fl_Grid, 865
gap, 869
layout, 870
margin, 870, 871
need_layout, 871
on_remove, 872
resize, 872
row_gap, 872
row_height, 872, 873
row_weight, 873
show_grid, 873, 874
widget, 874, 875
Fl_Grid.cxx, 2373
Fl_Grid.H, 2173, 2174
Fl_Grid::Cell, 441
~Cell, 441
next, 442
Fl_Group, 875
~Fl_Group, 884
array, 885
as_group, 885
begin, 885
bounds, 885
child, 886
clear, 886
clip_children, 887
current, 887
delete_child, 887
draw, 888
draw_child, 888
draw_children, 888
end, 888
find, 888
Fl_Group, 884
handle, 889
init_sizes, 889
insert, 889, 890
on_insert, 890
on_move, 890
on_remove, 891
remove, 891
resizable, 891, 892
resize, 893
sizes, 893
update_child, 894
Fl_Group.H, 2177, 2178
FL_GTK_DOWN_BOX
Enumerations.H, 2076
FL_GTK_DOWN_FRAME
Enumerations.H, 2076
FL_GTK_ROUND_DOWN_BOX
Enumerations.H, 2076
FL_GTK_ROUND_UP_BOX
Enumerations.H, 2076
FL_GTK_THIN_DOWN_BOX
Enumerations.H, 2076
FL_GTK_THIN_DOWN_FRAME
Enumerations.H, 2076
FL_GTK_THIN_UP_BOX
Enumerations.H, 2076
FL_GTK_THIN_UP_FRAME
Enumerations.H, 2076
FL_GTK_UP_BOX
Enumerations.H, 2076
FL_GTK_UP_FRAME
Enumerations.H, 2076
fl_height
Color & Font functions, 299
Fl_Help_Dialog, 894
load, 896
show, 896
textsize, 896
value, 896
Fl_Help_Dialog.H, 2179
Fl_Help_View, 896
~Fl_Help_View, 908
copy, 908
directory, 908

draw, 909
filename, 909
find, 909
Fl_Help_View, 908
handle, 909
leftline, 910
link, 910
load, 910
resize, 911
scrollbar_size, 911
text_selected, 911
title, 912
topline, 912
value, 912
Fl_Help_View.H, 2180
FL_HIDE
Enumerations.H, 2081
Fl_Hold_Browser, 913
Fl_Hold_Browser, 926
Fl_Hold_Browser.H, 2181
Fl_Hor_Fill_Slider, 927
Fl_Hor_Fill_Slider.H, 2182
Fl_Hor_Nice_Slider, 936
Fl_Hor_Nice_Slider.H, 2182
Fl_Hor_Slider, 945
Fl_Hor_Slider.H, 2183
Fl_Hor_Value_Slider, 954
Fl_Hor_Value_Slider.H, 2183
Fl_ICO_Image, 963
Fl_ICO_Image, 967
Fl_ICO_Image.H, 2183
Fl_ICO_Image::IconDirEntry, 2037
Fl_Idle_Handler
Callback Function Typedefs, 277
Fl_Image, 967
as_shared_image, 970
color_average, 970
copy, 970, 971
count, 971
d, 972
data, 972
desaturate, 972
draw, 972, 973
draw_empty, 973
draw_scaled, 973
fail, 973
Fl_Image, 970
h, 974
inactive, 974
label, 974, 975
ld, 975
release, 975
RGB_scaling, 975
scale, 975
scaling_algorithm, 976
uncache, 976
w, 976, 977
Fl_Image.H, 2184, 2185
Fl_RGB_Scaling, 2185
FL_RGB_SCALING_BILINEAR, 2185
FL_RGB_SCALING_NEAREST, 2185
FL_IMAGE_LABEL
Enumerations.H, 2073
Fl_Image_Reader, 977
Fl_Image_Reader.h, 2373
Fl_Image_Surface, 977
Fl_Image_Surface, 980
highres_image, 980
image, 980
is_current, 980
mask, 980
offscreen, 981
origin, 981, 982
printable_rect, 982
rescale, 982
set_current, 982
translate, 983
untranslate, 983
Fl_Image_Surface.H, 2187
FL_INLINE_CALLBACK_2
fl_callback_macros.H, 2127
Fl_Input, 983
draw, 995
Fl_Input, 994
handle, 995
handle_key, 995
handle_rmb, 996
fl_input
Common Dialog Classes and Functions, 385
Fl_Input.H, 2188
Fl_Input_, 996
~Fl_Input_, 1006
append, 1006
apply_undo, 1006
can_redo, 1006
can_undo, 1007
copy, 1007
copy_cuts, 1007
cursor_color, 1007
cut, 1008
drawtext, 1008, 1009
dvalue, 1009
Fl_Input_, 1005
handle_mouse, 1009
handletext, 1010
index, 1010
input_type, 1010
insert, 1010
insert_position, 1011
ivalue, 1012
line_end, 1012
line_start, 1012
mark, 1013
maximum_size, 1013
placeholder, 1013
position, 1014

readonly, 1014
redo, 1014
replace, 1015
resize, 1015
shortcut, 1016
size, 1016
static_value, 1017
tab_nav, 1017
textcolor, 1018
textfont, 1018
textsize, 1019
undo, 1019
up_down_position, 1019
value, 1020, 1021
word_end, 1021
word_start, 1022
wrap, 1022
Fl_Input_.H, 2189
Fl_Input_Choice, 1022
add, 1033
draw, 1034
Fl_Input_Choice, 1033
inp_x, 1034
input, 1034
menu_x, 1034
menubutton, 1034
resize, 1034
update_menubutton, 1034
value, 1035
Fl_Input_Choice.H, 2192
fl_input_str
Common Dialog Classes and Functions, 386
Fl_Int_Input, 1035
Fl_Int_Input, 1045
Fl_Int_Input.H, 2194
fl_intptr_t
platform_types.h, 2340
FL_JOIN_BEVEL
Drawing functions, 312
FL_JOIN_MITER
Drawing functions, 312
FL_JOIN_ROUND
Drawing functions, 312
Fl_JPEG_Image, 1046
Fl_JPEG_Image, 1049
Fl_JPEG_Image.H, 2194
FL_KEYBOARD
Enumerations.H, 2080
FL_KEYDOWN
Enumerations.H, 2080
FL_KEYUP
Enumerations.H, 2080
Fl_Label, 1050
draw, 1051
measure, 1051
type, 1051
Fl_Labeltype
Enumerations.H, 2082
fl_latin1_to_local
Color & Font functions, 299
FL_LEAVE
Enumerations.H, 2079
Fl_Light_Button, 1051
draw, 1059
Fl_Light_Button, 1059
handle, 1059
Fl_Light_Button.H, 2195
fl_lightness
Color & Font functions, 300
Fl_Line_Dial, 1060
Fl_Line_Dial.H, 2195
fl_line_style
Drawing functions, 328
fl_load_matrix
Drawing functions, 329
fl_local_to_latin1
Color & Font functions, 300
fl_local_to_mac_roman
Color & Font functions, 300
fl_luminance
Color & Font functions, 301
Fl_Mac_App_Menu, 1068
custom_application_menu_items, 1069
print, 1069
fl_mac_os_version
Mac OS X-specific symbols, 376
fl_mac_roman_to_local
Color & Font functions, 301
fl_mac_set_about
Mac OS X-specific symbols, 376
FL_MAJOR_VERSION
Version Numbers, 248
fl_make_path
Unicode and UTF-8 functions, 353
fl_make_path_for_file
Unicode and UTF-8 functions, 354
FL_MAX_BOXTYPE
Enumerations.H, 2076
fl_measure
Drawing functions, 330
fl_measure_pixmap
Drawing functions, 330
Fl_Menu.H, 2195
Fl_Menu_, 1069
add, 1079
clear, 1081
clear_submenu, 1082
copy, 1082
down_box, 1082
find_index, 1082, 1083
find_item, 1084
find_item_with_argument, 1084
find_item_with_user_data, 1085
Fl_Menu_, 1079
global, 1085
insert, 1085

item_pathname, 1086
menu, 1086, 1087
menu_box, 1087
menu_end, 1087
mode, 1088
mvalue, 1088
picked, 1088
prev_mvalue, 1089
remove, 1089
replace, 1089
size, 1089
test_shortcut, 1089
text, 1090
textcolor, 1090
textfont, 1090
textsize, 1090
value, 1090–1092
Fl_Menu_.H, 2196
Fl_Menu_Bar, 1092
draw, 1102
Fl_Menu_Bar, 1102
handle, 1103
play_menu, 1103
update, 1103
Fl_Menu_Bar.H, 2197
Fl_Menu_Button, 1104
draw, 1114
Fl_Menu_Button, 1114
handle, 1114
popup, 1115
POPUP1, 1114
POPUP12, 1114
POPUP123, 1114
POPUP13, 1114
POPUP2, 1114
POPUP23, 1114
POPUP3, 1114
popup_buttons, 1113
Fl_Menu_Button.H, 2198
FL_MENU_CHATTY
Fl_Menu_Item.H, 2199
FL_MENU_DIVIDER
Fl_Menu_Item.H, 2199
FL_MENU_HEADLINE
Fl_Menu_Item.H, 2199
FL_MENU_HORIZONTAL
Fl_Menu_Item.H, 2199
FL_MENU_INACTIVE
Fl_Menu_Item.H, 2199
FL_MENU_INVISIBLE
Fl_Menu_Item.H, 2199
Fl_Menu_Item, 1115
add, 1120
argument, 1120
callback, 1120, 1121
checkbox, 1121
deactivate, 1121
do_callback, 1121, 1122
draw, 1122
find_shortcut, 1122
headline, 1123
image, 1123
image_label, 1123
insert, 1123
label, 1124, 1125
labelcolor, 1125
labelfont, 1125
labeltype, 1125, 1126
measure, 1126
multi_label, 1126
next, 1126
popup, 1127
pulldown, 1127
radio, 1128
selectable, 1128
set, 1128
setonly, 1128
shortcut, 1128
size, 1128
submenu, 1129
test_shortcut, 1129
value, 1129
Fl_Menu_Item.H, 2198, 2199
FL_MENU_CHATTY, 2199
FL_MENU_DIVIDER, 2199
FL_MENU_HEADLINE, 2199
FL_MENU_HORIZONTAL, 2199
FL_MENU_INACTIVE, 2199
FL_MENU_INVISIBLE, 2199
FL_MENU_RADIO, 2199
FL_MENU_TOGGLE, 2199
FL_MENU_VALUE, 2199
FL_SUBMENU, 2199
FL_SUBMENU_POINTER, 2199
FL_MENU_RADIO
Fl_Menu_Item.H, 2199
FL_MENU_TOGGLE
Fl_Menu_Item.H, 2199
FL_MENU_VALUE
Fl_Menu_Item.H, 2199
Fl_Menu_Window, 1129
Fl_Menu_Window.H, 2202
fl_message
Common Dialog Classes and Functions, 386
Fl_Message.h, 2374
fl_message.H, 2202
fl_message_hotspot
Common Dialog Classes and Functions, 387
fl_message_icon
Common Dialog Classes and Functions, 387
fl_message_icon_label
Common Dialog Classes and Functions, 388
fl_message_position
Common Dialog Classes and Functions, 388, 389
fl_ask.H, 2112
fl_message_title

Common Dialog Classes and Functions, 390
fl_message_title_default
Common Dialog Classes and Functions, 390
FL_METHOD_CALLBACK_1
fl_callback_macros.H, 2128
FL_MINOR_VERSION
Version Numbers, 248
fl_mkdir
Unicode and UTF-8 functions, 354
FL_MOUSEWHEEL
Enumerations.H, 2081
FL_MOVE
Enumerations.H, 2080
fl_mult_matrix
Drawing functions, 331
Fl_Multi_Browser, 1142
Fl_Multi_Browser, 1156
Fl_Multi_Browser.H, 2202
FL_MULTI_LABEL
Enumerations.H, 2073
Fl_Multi_Label, 1156
label, 1158
labela, 1158
labelb, 1158
typea, 1158
typeb, 1159
Fl_Multi_Label.H, 2203
Fl_Multiline_Input, 1159
Fl_Multiline_Input, 1169
Fl_Multiline_Input.H, 2203
Fl_Multiline_Output, 1169
Fl_Multiline_Output, 1179
Fl_Multiline_Output.H, 2204
Fl_Native_File_Chooser, 1179
~Fl_Native_File_Chooser, 1183
BROWSE_DIRECTORY, 1182
BROWSE_FILE, 1182
BROWSE_MULTI_DIRECTORY, 1182
BROWSE_MULTI_FILE, 1182
BROWSE_SAVE_DIRECTORY, 1182
BROWSE_SAVE_FILE, 1182
count, 1183
directory, 1183
errmsg, 1183
filename, 1183
filter, 1183, 1184
filter_value, 1184
Fl_Native_File_Chooser, 1183
NEW_FOLDER, 1182
NO_OPTIONS, 1182
Option, 1182
options, 1185
preset_file, 1185
PREVIEW, 1182
SAVEAS_CONFIRM, 1182
show, 1185
title, 1185
Type, 1182
USE_FILTER_EXT, 1182
Fl_Native_File_Chooser.H, 2204
Fl_Native_File_Chooser_Kdialog.H, 2376
Fl_Native_File_Chooser_Zenity.H, 2377
Fl_Nice_Slider, 1186
Fl_Nice_Slider.H, 2207
FL_NO_BOX
Enumerations.H, 2075
FL_NO_EVENT
Enumerations.H, 2078
FL_NO_LABEL
Enumerations.H, 2082
fl_nonspacing
Unicode and UTF-8 functions, 354
FL_NORMAL_LABEL
Enumerations.H, 2082
FL_NORMAL_SIZE
Enumerations.H, 2085
fl_not_clipped
Drawing functions, 331
fl_numericsort
numericsort.c, 2404
Fl_Object.H, 2207
Fl_Offscreen
platform_types.h, 2340
FL_OFLAT_BOX
Enumerations.H, 2075
Fl_Old_Idle_Handler
Callback Function Typedefs, 277
fl_old_shortcut
Drawing functions, 331
fl_open
Unicode and UTF-8 functions, 355
fl_open_callback
Mac OS X-specific symbols, 376
fl_open_display
Fl.cxx, 2358
fl_open_ext
Unicode and UTF-8 functions, 355
fl_open_uri
File names and URI utility functions, 403
Fl_Option
Fl, 415
FL_ORIENT_DOWN
Enumerations.H, 2083
FL_ORIENT_LEFT
Enumerations.H, 2083
FL_ORIENT_NE
Enumerations.H, 2083
FL_ORIENT_NONE
Enumerations.H, 2083
FL_ORIENT_NW
Enumerations.H, 2083
FL_ORIENT_RIGHT
Enumerations.H, 2083
FL_ORIENT_SE
Enumerations.H, 2083
FL_ORIENT_SW

Enumerations.H, 2083
FL_ORIENT_UP
Enumerations.H, 2083
Fl_Orientation
Enumerations.H, 2082
FL_OSHADOW_BOX
Enumerations.H, 2075
Fl_Output, 1195
Fl_Output, 1205
Fl_Output.H, 2207
FL_OVAL_BOX
Enumerations.H, 2075
FL_OVAL_FRAME
Enumerations.H, 2075
fl_overlay_clear
Drawing functions, 332
fl_overlay_rect
Drawing functions, 332
Fl_Overlay_Window, 1205
as_overlay_window, 1219
draw_overlay, 1219
Fl_Overlay_Window, 1218
flush, 1219
hide, 1219
redraw_overlay, 1219
resize, 1219
show, 1220
Fl_Overlay_Window.H, 2208
FL_OVERRIDE
fl_attr.h, 2114
fl_override_scale
Drawing functions, 333
fl_oxy.h, 2377
FL_OXY_BUTTON_DOWN_BOX
Enumerations.H, 2076
FL_OXY_BUTTON_UP_BOX
Enumerations.H, 2076
FL_OXY_DOWN_BOX
Enumerations.H, 2076
FL_OXY_DOWN_FRAME
Enumerations.H, 2076
FL_OXY_ROUND_DOWN_BOX
Enumerations.H, 2076
FL_OXY_ROUND_UP_BOX
Enumerations.H, 2076
FL_OXY_THIN_DOWN_BOX
Enumerations.H, 2076
FL_OXY_THIN_DOWN_FRAME
Enumerations.H, 2076
FL_OXY_THIN_UP_BOX
Enumerations.H, 2076
FL_OXY_THIN_UP_FRAME
Enumerations.H, 2076
FL_OXY_UP_BOX
Enumerations.H, 2076
FL_OXY_UP_FRAME
Enumerations.H, 2076
Fl_Pack, 1220
clear, 1229
draw, 1230
draw_filler_, 1230
Fl_Pack, 1229
horizontal, 1230
resize, 1230
Fl_Pack.H, 2208
Fl_Paged_Device, 1231
A0, 1233
A1, 1233
A2, 1233
A3, 1233
A4, 1234
A5, 1234
A6, 1234
A7, 1234
A8, 1234
A9, 1234
B0, 1234
B1, 1234
B10, 1234
B2, 1234
B3, 1234
B4, 1234
B5, 1234
B6, 1234
B7, 1234
B8, 1234
B9, 1234
begin_job, 1234
begin_page, 1235
end_job, 1235
end_page, 1235
EXECUTIVE, 1234
FOLIO, 1234
LANDSCAPE, 1234
LEDGER, 1234
LEGAL, 1234
LETTER, 1234
margins, 1235
ORIENTATION, 1234
Page_Format, 1233
Page_Layout, 1234
PORTRAIT, 1234
REVERSED, 1234
rotate, 1236
scale, 1236
start_job, 1236
start_page, 1237
TABLOID, 1234
Fl_Paged_Device.cxx, 2378
Fl_Paged_Device.H, 2209
Fl_Paged_Device::page_format, 2040
fl_parse_color
Fl_get_system_colors.cxx, 2370
fl_password
Common Dialog Classes and Functions, 390, 391
fl_password_str

Common Dialog Classes and Functions, 391, 392
FL_PASTE
Enumerations.H, 2081
FL_PATCH_VERSION
Version Numbers, 248
Fl_PDF_File_Surface, 1237
begin_document, 1240
begin_job, 1240
begin_page, 1241
end_job, 1241
end_page, 1241
is_current, 1241
margins, 1241
origin, 1242
printable_rect, 1242
rotate, 1243
scale, 1243
set_current, 1243
translate, 1243
untranslate, 1244
Fl_PDF_File_Surface.H, 2211
fl_pie
Drawing functions, 333
Fl_Pixmap, 1244
color_average, 1247
copy, 1247
desaturate, 1248
draw, 1248
Fl_Pixmap, 1247
label, 1248, 1249
uncache, 1249
Fl_Pixmap.H, 2211
fl_plastic.cxx, 2378
FL_PLASTIC_DOWN_BOX
Enumerations.H, 2076
FL_PLASTIC_DOWN_FRAME
Enumerations.H, 2076
FL_PLASTIC_ROUND_DOWN_BOX
Enumerations.H, 2076
FL_PLASTIC_ROUND_UP_BOX
Enumerations.H, 2076
FL_PLASTIC_THIN_DOWN_BOX
Enumerations.H, 2076
FL_PLASTIC_THIN_UP_BOX
Enumerations.H, 2076
FL_PLASTIC_UP_BOX
Enumerations.H, 2075
FL_PLASTIC_UP_FRAME
Enumerations.H, 2076
Fl_Plugin, 1249
Fl_Plugin, 1250
Fl_Plugin.H, 2212
Fl_Plugin_Manager, 1250
~Fl_Plugin_Manager, 1254
addPlugin, 1254
klass_list, 1254
load, 1254
loadAll, 1254
removePlugin, 1255
Fl_PNG_Image, 1255
Fl_PNG_Image, 1258
Fl_PNG_Image.H, 2213
Fl_PNM_Image, 1259
Fl_PNM_Image, 1262
Fl_PNM_Image.H, 2214
fl_polygon
Drawing functions, 334
fl_pop_clip
Drawing functions, 334
Fl_Positioner, 1262
draw, 1270
Fl_Positioner, 1270
handle, 1270
Fl_Positioner.H, 2214
Fl_PostScript.H, 2215, 2216
Fl_PostScript_Close_Command, 2215
Fl_PostScript_Close_Command
Fl_PostScript.H, 2215
Fl_PostScript_File_Device, 1271
begin_job, 1275
begin_page, 1275
end_current, 1276
end_job, 1276
end_page, 1276
margins, 1276
origin, 1277
printable_rect, 1277
rotate, 1278
scale, 1278
set_current, 1278
start_job, 1278, 1279
translate, 1279
untranslate, 1279
Fl_Preferences, 1279
~Fl_Preferences, 1287
C_LOCALE, 1284
CLEAR, 1284
CORE, 1284
CORE_READ_OK, 1299
CORE_SYSTEM, 1284
CORE_SYSTEM_L, 1284
CORE_USER, 1284
CORE_USER_L, 1284
CORE_WRITE_OK, 1299
delete_entry, 1288
delete_group, 1288
dirty, 1289
entries, 1289
entry, 1289
entry_exists, 1289
file_access, 1289, 1290
filename, 1290
Fl_Preferences, 1284–1288
flush, 1291
get, 1291–1294
get_userdata_path, 1295

group, 1295
group_exists, 1296
groups, 1296
ID, 1284
MEMORY, 1284
new_UUID, 1296
NONE, 1299
Root, 1284
ROOT_MASK, 1284
set, 1296–1299
size, 1299
SYSTEM, 1284
SYSTEM_L, 1284
UNKNOWN_ROOT_TYPE, 1284
USER, 1284
USER_L, 1284
Fl_Preferences.H, 2217
Fl_Preferences::Entry, 444
Fl_Preferences::Name, 2038
Name, 2038
Fl_Preferences::Node, 2039
Fl_Preferences::RootNode, 2041
Fl_Printer, 1300
begin_job, 1304
begin_page, 1304
end_job, 1305
end_page, 1305
is_current, 1305
margins, 1305
origin, 1306
printable_rect, 1306
rotate, 1306
scale, 1307
set_current, 1307
translate, 1307
untranslate, 1307
Fl_Printer.H, 2220
Fl_Private.H, 2378, 2380
cairo_make_current, 2379
idle_, 2380
set_idle_, 2379
Fl_Progress, 1308
draw, 1315
Fl_Progress, 1315
maximum, 1315
minimum, 1315
value, 1315
Fl_Progress.H, 2221
FL_PUSH
Enumerations.H, 2079
fl_push_clip
Drawing functions, 334
fl_push_matrix
Drawing functions, 334
fl_putenv
Unicode and UTF-8 functions, 355
Fl_Radio_Button, 1316
Fl_Radio_Button, 1323
Fl_Radio_Button.H, 2221
Fl_Radio_Light_Button, 1324
Fl_Radio_Light_Button.H, 2222
Fl_Radio_Round_Button, 1331
Fl_Radio_Round_Button, 1339
Fl_Radio_Round_Button.H, 2222
FL_READ
Enumerations.H, 2074
fl_read_image
Drawing functions, 335
FL_REASON_CANCELLED
Enumerations.H, 2077
FL_REASON_CHANGED
Enumerations.H, 2077
FL_REASON_CLOSED
Enumerations.H, 2077
FL_REASON_DESELECTED
Enumerations.H, 2077
FL_REASON_DRAGGED
Enumerations.H, 2077
FL_REASON_ENTER_KEY
Enumerations.H, 2077
FL_REASON_GOT_FOCUS
Enumerations.H, 2077
FL_REASON_LOST_FOCUS
Enumerations.H, 2077
FL_REASON_OPENED
Enumerations.H, 2077
FL_REASON_RELEASED
Enumerations.H, 2077
FL_REASON_RESELECTED
Enumerations.H, 2077
FL_REASON_SELECTED
Enumerations.H, 2077
FL_REASON_UNKNOWN
Enumerations.H, 2077
FL_REASON_USER
Enumerations.H, 2077
Fl_Rect, 1339
b, 1341
Fl_Rect, 1341
inset, 1341
r, 1341
fl_rect
Drawing functions, 335
fl_rect.cxx, 2381
Fl_Rect.H, 2223
fl_rectbound
fl_boxtype.cxx, 2363
fl_rectf
Drawing functions, 336
Fl_Region
platform_types.h, 2340
fl_register_images
Fl_Shared_Image.H, 2232
FL_RELEASE
Enumerations.H, 2079
fl_remove_symbol

Drawing functions, 336
fl_rename
Unicode and UTF-8 functions, 356
Fl_Repeat_Button, 1342
Fl_Repeat_Button, 1349
handle, 1350
Fl_Repeat_Button.H, 2224
fl_rescale_offscreen
Drawing functions, 337
FL_RESERVED_TYPE
Fl_Widget.H, 2299
fl_reset_spot
Drawing functions, 337
fl_restore_scale
Drawing functions, 337
Fl_Return_Button, 1350
draw, 1358
Fl_Return_Button, 1358
handle, 1358
Fl_Return_Button.H, 2224
FL_RFLAT_BOX
Enumerations.H, 2075
Fl_RGB_Image, 1359
array, 1366
as_svg_image, 1364
color_average, 1364
copy, 1364
desaturate, 1365
draw, 1365
Fl_RGB_Image, 1362, 1363
label, 1365
max_size, 1366
normalize, 1366
uncache, 1366
Fl_RGB_Image.H, 2225
Fl_RGB_Scaling
Fl_Image.H, 2185
FL_RGB_SCALING_BILINEAR
Fl_Image.H, 2185
FL_RGB_SCALING_NEAREST
Fl_Image.H, 2185
fl_rmdir
Unicode and UTF-8 functions, 356
Fl_Roller, 1366
draw, 1375
Fl_Roller, 1375
handle, 1375
Fl_Roller.H, 2225
fl_rotate
Drawing functions, 337
Fl_Round_Button, 1376
Fl_Round_Button, 1384
Fl_Round_Button.H, 2226
Fl_Round_Clock, 1384
Fl_Round_Clock, 1392
Fl_Round_Clock.H, 2226
FL_ROUND_DOWN_BOX
Enumerations.H, 2075
FL_ROUND_UP_BOX
Enumerations.H, 2075
FL_ROUNDED_BOX
Enumerations.H, 2075
FL_ROUNDED_FRAME
Enumerations.H, 2075
fl_rounded_rect
Drawing functions, 338
fl_rounded_rectf
Drawing functions, 338
FL_RSHADOW_BOX
Enumerations.H, 2075
Fl_Scalable_Graphics_Driver.H, 2381
fl_scale
Drawing functions, 338
Fl_Scheme, 1392
add_scheme_name, 1393
names, 1393
num_schemes, 1394
plastic_color_average, 1394
Fl_Scheme.cxx, 2383
Fl_Scheme.H, 2226
Fl_Scheme_Choice, 1395
Fl_Scheme_Choice, 1404
handle, 1404
init_value, 1405
scheme_cb_, 1405
Fl_Scheme_Choice.H, 2227
FL_SCREEN_CONFIGURATION_CHANGED
Enumerations.H, 2081
Fl_Screen_Driver.H, 2383
Fl_Scroll, 1405
~Fl_Scroll, 1415
bbox, 1416
delete_child, 1416
draw, 1416
fix_scrollbar_order, 1417
Fl_Scroll, 1415
handle, 1417
on_insert, 1417
on_move, 1418
recalc_scrollbars, 1418
resize, 1419
scroll_to, 1419
scrollbar_size, 1419, 1420
xposition, 1420
yposition, 1420
fl_scroll
Drawing functions, 338
Fl_Scroll.H, 2228
Fl_Scroll::Scrollbar_Data, 2041
Fl_Scroll::ScrollInfo, 2042
Fl_Scrollbar, 1420
draw, 1430
Fl_Scrollbar, 1430
handle, 1430
linesize, 1431
value, 1431

Fl_Scrollbar.H, 2229
Fl_Secret_Input, 1432
Fl_Secret_Input, 1442
handle, 1442
Fl_Secret_Input.H, 2230
Fl_Select_Browser, 1443
Fl_Select_Browser, 1456
Fl_Select_Browser.H, 2230
FL_SELECTIONCLEAR
Enumerations.H, 2081
fl_set_spot
Drawing functions, 339
fl_set_status
Drawing functions, 339
FL_SHADOW_BOX
Enumerations.H, 2075
FL_SHADOW_FRAME
Enumerations.H, 2075
Fl_Shared_Handler
Fl_Shared_Image.H, 2231
Fl_Shared_Image, 1457
~Fl_Shared_Image, 1461
add, 1461
add_handler, 1461
as_shared_image, 1461
color_average, 1461
compare, 1462
copy, 1462, 1463
copy_, 1463
desaturate, 1463
draw, 1464
find, 1464
Fl_Shared_Image, 1460
get, 1464, 1465
image, 1465
images, 1466
num_images, 1466
original, 1466
refcount, 1466
release, 1467
uncache, 1467
update, 1467
Fl_Shared_Image.H, 2230, 2232
fl_register_images, 2232
Fl_Shared_Handler, 2231
Fl_Shared_Image::Deleter, 443
FL_SHORTCUT
Enumerations.H, 2080
Fl_Shortcut
fl_types.h, 2289
Fl_Shortcut_Button, 1467
draw, 1475
Fl_Shortcut_Button, 1475
handle, 1475
value, 1475, 1476
Fl_Shortcut_Button.H, 2233
fl_shortcut_label
Drawing functions, 339, 340
FL_SHOW
Enumerations.H, 2081
fl_show_colormap
Color & Font functions, 301
fl_show_colormap.H, 2234
fl_show_input.H, 2234
Fl_Simple_Counter, 1476
Fl_Simple_Counter.H, 2235
Fl_Single_Window, 1485
flush, 1497
show, 1497
Fl_Single_Window.H, 2235
fl_size
Color & Font functions, 302
Fl_Slider, 1498
bounds, 1508
draw, 1508
draw_ticks, 1508
Fl_Slider, 1507
handle, 1508
increment_lin_log, 1509
LINEAR_SCALE, 1507
LOG_SCALE, 1507
num_ticks, 1509
position_to_value, 1509
scale, 1509
Scale_Type, 1507
scrollvalue, 1510
slider_size, 1510
Tick_Position, 1507
ticks, 1510
TICKS_ABOVE, 1507
TICKS_BELOW, 1507
TICKS_LEFT, 1507
TICKS_NONE, 1507
TICKS_RIGHT, 1507
value_to_position, 1510
Fl_Slider.H, 2236
FL_SOLID
Drawing functions, 312
Fl_Spinner, 1511
draw, 1521
Fl_Spinner, 1521
handle, 1521
resize, 1522
step, 1522
type, 1522
value, 1523
wrap, 1523
Fl_Spinner.H, 2237
Fl_Spinner::Fl_Spinner_Input, 1524
handle, 1533
fl_stat
Unicode and UTF-8 functions, 357
fl_strdup
String handling functions, 372
fl_string_functions.h, 2238, 2239
FL_SUBMENU

Fl_Menu_Item.H, 2199
FL_SUBMENU_POINTER
Fl_Menu_Item.H, 2199
Fl_Surface_Device, 1534
end_current, 1535
is_current, 1535
pop_current, 1535
push_current, 1535
set_current, 1535
surface, 1536
Fl_SVG_File_Surface, 1536
~Fl_SVG_File_Surface, 1539
close, 1539
Fl_SVG_File_Surface, 1538
origin, 1539
printable_rect, 1539
translate, 1540
untranslate, 1540
Fl_SVG_File_Surface.H, 2239
Fl_SVG_Image, 1540
as_svg_image, 1545
color_average, 1546
copy, 1546
desaturate, 1546
draw, 1547
Fl_SVG_Image, 1544, 1545
normalize, 1547
proportional, 1548
resize, 1547
scale, 1547
Fl_SVG_Image.H, 2240
FL_SYMBOL_LABEL
Enumerations.H, 2073
Fl_Sys_Menu_Bar, 1548
about, 1560
add, 1560, 1561
clear, 1561
clear_submenu, 1561
create_window_menu, 1561
draw, 1561
Fl_Sys_Menu_Bar, 1560
insert, 1562
is_global, 1562
menu, 1562
mode, 1563
no_window_menu, 1559
play_menu, 1563
remove, 1563
replace, 1563
tabbing_mode_automatic, 1559
tabbing_mode_none, 1559
tabbing_mode_preferred, 1559
update, 1564
window_menu_style, 1564
window_menu_style_enum, 1559
Fl_Sys_Menu_Bar.H, 2241
Fl_Sys_Menu_Bar_Driver.H, 2386
fl_system
Unicode and UTF-8 functions, 357
Fl_System_Driver.H, 2387
Fl_Table, 1564
~Fl_Table, 1579
array, 1580
callback, 1580
callback_col, 1581
callback_context, 1581
callback_row, 1581
child, 1581
children, 1581
clear, 1581
col_header, 1581
col_resize, 1581
col_resize_min, 1582
col_width, 1582
col_width_all, 1582
CONTEXT_CELL, 1579
CONTEXT_COL_HEADER, 1579
CONTEXT_ENDPAGE, 1579
CONTEXT_NONE, 1579
CONTEXT_RC_RESIZE, 1579
CONTEXT_ROW_HEADER, 1579
CONTEXT_STARTPAGE, 1579
CONTEXT_TABLE, 1579
cursor2rowcol, 1582
damage_zone, 1582
do_callback, 1582
draw, 1583
draw_cell, 1583
find_cell, 1584
Fl_Table, 1579
get_selection, 1584
handle, 1585
init_sizes, 1585
insert, 1585
is_interactive_resize, 1585
is_selected, 1585
move_cursor, 1585
recalc_dimensions, 1586
redraw_range, 1586
resize, 1586
row_col_clamp, 1586
row_header, 1586
row_height, 1586
row_height_all, 1587
row_resize, 1587
row_resize_min, 1587
rows, 1587
scrollbar_size, 1587
set_selection, 1588
tab_cell_nav, 1588
table_box, 1589
table_resized, 1589
table_scrolled, 1589
TableContext, 1579
top_row, 1589
visible_cells, 1589

when, 1590
Fl_Table.H, 2242
Fl_Table_Row, 1590
~Fl_Table_Row, 1604
clear, 1604
Fl_Table_Row, 1604
handle, 1604
row_selected, 1604
rows, 1605
select_all_rows, 1605
select_row, 1605
type, 1606
Fl_Table_Row.H, 2248
Fl_Tabs, 1606
clear_tab_positions, 1620
client_area, 1620
draw, 1621
draw_tab, 1621
Fl_Tabs, 1619
handle, 1621
handle_overflow, 1621
handle_overflow_menu, 1622
hit_close, 1622
hit_overflow_menu, 1622
hit_tabs_area, 1624
maybe_do_callback, 1624
on_insert, 1624
on_move, 1624
on_remove, 1624
OVERFLOW_CLIP, 1619
OVERFLOW_COMPRESS, 1619
OVERFLOW_DRAG, 1619
OVERFLOW_PULLDOWN, 1619
overflow_type, 1628
push, 1625
redraw_tabs, 1625
resize, 1625
show, 1625
tab_align, 1625, 1626
tab_count, 1628
tab_flags, 1628
tab_height, 1626
tab_pos, 1629
tab_positions, 1626
tab_width, 1629
take_focus, 1627
value, 1627
which, 1628
Fl_Tabs.H, 2249
Fl_Terminal, 1629
_RESERVED_1, 1650
_RESERVED_2, 1650
~Fl_Terminal, 1652
ansi, 1652
append, 1653
append_ascii, 1653
append_utf8, 1653
Attrib, 1650
BG_XTERM, 1650
BOLD, 1650
box, 1654
CharFlags, 1650
clear, 1654
clear_screen, 1654
clear_screen_home, 1654
color, 1655
CR_TO_LF, 1651
cursor_col, 1655
cursor_cr, 1655
cursor_down, 1655
cursor_right, 1655
cursor_row, 1656
cursor_up, 1656
delete_rows, 1656
DIM, 1650
display_columns, 1656
display_rows, 1656
draw, 1657
draw_buff, 1657
draw_row, 1657
draw_row_bg, 1657
EOL, 1650
error_char, 1657, 1658
FG_XTERM, 1650
Fl_Terminal, 1651, 1652
get_selection, 1658
h_to_row, 1659
handle, 1659
handle_unknown_char, 1659
history_lines, 1659
history_use, 1659
hscrollbar, 1672
hscrollbar_style, 1659, 1660
insert_char, 1660
insert_rows, 1660
INVERSE, 1650
is_inside_selection, 1660
ITALIC, 1650
LF_TO_CR, 1651
LF_TO_CRLF, 1651
NO_REDRAW, 1651
NORMAL, 1650
OFF, 1651
OutFlags, 1651
output_translate, 1660
PER_WRITE, 1651
plot_char, 1661
print_char, 1661, 1662
printf, 1662
RATE_LIMITED, 1651
redraw_rate, 1662
redraw_style, 1663
RedrawStyle, 1651
reset_terminal, 1663
resize, 1663
scroll, 1664

scrollbar, 1672
scrollbar_actual_size, 1664
SCROLLBAR_AUTO, 1651
SCROLLBAR_OFF, 1651
SCROLLBAR_ON, 1651
scrollbar_size, 1664
ScrollbarStyle, 1651
selection_extend, 1665
selection_text, 1665
selection_text_len, 1665
show_unknown, 1665
STRIKEOUT, 1650
text, 1665
textattrib, 1666
textbgcolor, 1666
textbgcolor_default, 1667
textbgcolor_xterm, 1667
textcolor, 1667
textfgcolor, 1668
textfgcolor_default, 1668
textfgcolor_xterm, 1668
textfont, 1669
textsize, 1669
u8c_disp_row, 1669
u8c_hist_row, 1670
u8c_hist_use_row, 1670
u8c_ring_row, 1670
UNDERLINE, 1650
utf8_char_at_disp, 1671
utf8_char_at_glob, 1671
vprintf, 1671
w_to_col, 1672
walk_selection, 1672
Fl_Terminal Technical Documentation, 167
Fl_Terminal.H, 2250, 2251
Fl_Terminal::CharStyle, 442
Fl_Terminal::Cursor, 443
Fl_Terminal::EscapeSeq, 444
Fl_Terminal::Margin, 2038
Fl_Terminal::PartialUtf8Buf, 2040
Fl_Terminal::RingBuffer, 2040
Fl_Terminal::Selection, 2043
get_selection, 2043
Fl_Terminal::Utf8Char, 2044
Fl_Text_Buffer, 1673
add_modify_callback, 1678
address, 1679
append, 1679
appendfile, 1679
byte_at, 1679
can_redo, 1680
can_undo, 1680
canUndo, 1680
char_at, 1680
copy, 1681
count_displayed_characters, 1681
count_lines, 1681
estimate_lines, 1681
file_encoding_warning_message, 1690
findchar_backward, 1681
findchar_forward, 1682
Fl_Text_Buffer, 1678
highlight_text, 1682
insert, 1682
insert_, 1682
insertfile, 1683
is_word_separator, 1683
length, 1683
line_end, 1683
line_start, 1684
line_text, 1684
loadfile, 1684
mTabDist, 1690
next_char, 1684
outputfile, 1685
prev_char, 1685
printf, 1685
remove, 1685
remove_, 1686
replace, 1686
rewind_lines, 1686
savefile, 1686
search_backward, 1687
search_forward, 1687
secondary_selection_text, 1687
selection_text, 1688
skip_displayed_characters, 1688
tab_distance, 1688
text, 1688
text_range, 1688
text_str, 1689
transcoding_warning_action, 1690
undo, 1689
vprintf, 1689
word_end, 1689
word_start, 1690
Fl_Text_Buffer.H, 2260
Fl_Text_Display, 1690
~Fl_Text_Display, 1709
absolute_top_line_number, 1709
ATTR_BGCOLOR, 1708
ATTR_BGCOLOR_EXT, 1708
ATTR_BGCOLOR_EXT_, 1708
ATTR_GRAMMAR, 1708
ATTR_LINES_MASK, 1708
ATTR_SPELLING, 1708
ATTR_STRIKE_THROUGH, 1708
ATTR_UNDERLINE, 1708
BLOCK_CURSOR, 1708
buffer, 1709
buffer_modified_cb, 1710
buffer_predelete_cb, 1710
calc_last_char, 1710
calc_line_starts, 1711
CARET_CURSOR, 1708
clear_rect, 1711

col_to_x, 1711
count_lines, 1711
cursor_color, 1712
cursor_style, 1712
DIM_CURSOR, 1708
display_insert, 1712
display_needs_recalc, 1713
draw, 1713
draw_cursor, 1713
draw_line_numbers, 1713
draw_range, 1713
draw_string, 1714
draw_text, 1714
draw_vline, 1714
empty_vlines, 1715
extend_range_for_styles, 1715
find_line_end, 1715
find_wrap_range, 1715
find_x, 1716
Fl_Text_Display, 1708
get_absolute_top_line_number, 1716
grammar_underline_color, 1716, 1717
handle, 1717
handle_rmb, 1717
handle_vline, 1717
HEAVY_CURSOR, 1708
highlight_data, 1718
in_selection, 1719
insert, 1719
insert_position, 1719
line_end, 1720
line_start, 1720
linenumber_align, 1720
linenumber_bgcolor, 1721
linenumber_fgcolor, 1721
linenumber_font, 1721
linenumber_format, 1721
linenumber_size, 1721
linenumber_width, 1721
longest_vline, 1722
maintain_absolute_top_line_number, 1722
maintaining_absolute_top_line_number, 1722
measure_deleted_lines, 1722
measure_proportional_character, 1722
measure_vline, 1723
move_down, 1723
move_left, 1723
move_right, 1723
move_up, 1724
NORMAL_CURSOR, 1707
offset_line_starts, 1724
overstrike, 1724
position_style, 1724
position_to_line, 1725
position_to_linecol, 1725
position_to_xy, 1726
recalc_display, 1726
redisplay_range, 1726
reset_absolute_top_line_number, 1726
resize, 1727
rewind_lines, 1727
scroll, 1727
scroll_, 1727
scroll_timer_cb, 1729
scrollbar_align, 1729
scrollbar_size, 1729
scrollbar_width, 1730
secondary_selection_color, 1730
shortcut, 1730, 1731
show_cursor, 1731
show_insert_position, 1731
SIMPLE_CURSOR, 1708
skip_lines, 1731
spelling_underline_color, 1732
string_width, 1732
style_buffer, 1732
textcolor, 1732, 1733
textfont, 1733
textsize, 1733
update_h_scrollbar, 1733
update_line_starts, 1734
update_v_scrollbar, 1734
vline_length, 1734
word_end, 1734
word_start, 1735
WRAP_AT_BOUNDS, 1708
WRAP_AT_COLUMN, 1708
WRAP_AT_PIXEL, 1708
wrap_mode, 1735
WRAP_NONE, 1708
wrap_uses_character, 1735
wrapped_column, 1736
wrapped_line_counter, 1736
wrapped_row, 1737
x_to_col, 1737
xy_to_position, 1738
xy_to_rowcol, 1738
Fl_Text_Display.H, 2264
Fl_Text_Display::Style_Table_Entry, 2044
Fl_Text_Editor, 1738
add_key_binding, 1756
global_key_bindings, 1761
handle, 1756
insert_mode, 1756
kf_backspace, 1756
kf_c_s_move, 1756
kf_copy, 1756
kf_ctrl_move, 1757
kf_cut, 1757
kf_default, 1757
kf_delete, 1757
kf_down, 1757
kf_end, 1757
kf_enter, 1757
kf_home, 1758
kf_ignore, 1758

kf_insert, 1758
kf_left, 1758
kf_m_s_move, 1758
kf_meta_move, 1758
kf_move, 1759
kf_page_down, 1759
kf_page_up, 1759
kf_paste, 1759
kf_redo, 1759
kf_right, 1759
kf_select_all, 1759
kf_shift_move, 1760
kf_undo, 1760
kf_up, 1760
remove_key_binding, 1760
tab_nav, 1760, 1761
Fl_Text_Editor.H, 2269
Fl_Text_Editor::Key_Binding, 2037
fl_text_extents
Color & Font functions, 302, 303
Fl_Text_Selection, 1761
end, 1763
includes, 1763
length, 1763
position, 1763
selected, 1763, 1764
set, 1764
start, 1764
update, 1765
FL_THIN_DOWN_BOX
Enumerations.H, 2075
FL_THIN_DOWN_FRAME
Enumerations.H, 2075
FL_THIN_UP_BOX
Enumerations.H, 2075
FL_THIN_UP_FRAME
Enumerations.H, 2075
Fl_Tile, 1765
cursor, 1776
drag_intersection, 1776
Fl_Tile, 1776
handle, 1777
init_size_range, 1777
move_intersection, 1778
on_insert, 1778
on_move, 1778
on_remove, 1778
position, 1778
request_grow_b, 1779
request_grow_l, 1779
request_grow_r, 1779
request_grow_t, 1779
request_shrink_b, 1780
request_shrink_l, 1780
request_shrink_r, 1780
request_shrink_t, 1781
resize, 1781
set_cursor, 1781
size_range, 1782
Fl_Tile.H, 2270
Fl_Tile::Size_Range, 2043
Fl_Tiled_Image, 1782
color_average, 1785
copy, 1786
desaturate, 1786
draw, 1786
Fl_Tiled_Image, 1785
Fl_Tiled_Image.H, 2271
Fl_Timeout, 1787
add_timeout, 1789
current, 1790
current_timeout, 1794
elapse_timeouts, 1790
first_timeout, 1794
free_timeout, 1794
get, 1790
has_timeout, 1791
insert, 1791
make_current, 1792
release, 1792
remove_next_timeout, 1792
remove_timeout, 1793
repeat_timeout, 1793
time_to_wait, 1793
Fl_Timeout.cxx, 2389
Fl_Timeout.h, 2389, 2390
Fl_Timeout_Handler
Callback Function Typedefs, 277
Fl_Timer, 1794
direction, 1802
draw, 1802
Fl_Timer, 1801
handle, 1802
suspended, 1802
Fl_Timer.H, 2272
Fl_Timestamp
platform_types.h, 2340
Fl_Toggle_Button, 1803
Fl_Toggle_Button, 1810
Fl_Toggle_Button.H, 2272
Fl_Toggle_Light_Button.H, 2273
Fl_Toggle_Round_Button.H, 2273
fl_tolower
Unicode and UTF-8 functions, 357
Fl_Tooltip, 1811
color, 1813
current, 1813
delay, 1813
disable, 1813
enable, 1813
enabled, 1813
enter_area, 1814
font, 1814
hidedelay, 1814
hoverdelay, 1814
margin_height, 1815

margin_width, 1815
override_text, 1815
size, 1815
textcolor, 1816
wrap_width, 1816
Fl_Tooltip.H, 2274
fl_toupper
Unicode and UTF-8 functions, 358
fl_transform_dx
Drawing functions, 341
fl_transform_dy
Drawing functions, 341
fl_transform_x
Drawing functions, 341
fl_transform_y
Drawing functions, 341
fl_transformed_vertex
Drawing functions, 341
fl_translate
Drawing functions, 342
Fl_Tree, 1816
add, 1835
auto_resize_children, 1836
calc_dimensions, 1837
calc_tree, 1837
callback_item, 1837
callback_reason, 1837, 1838
clear, 1838
clear_children, 1838
close, 1838, 1839
closeicon, 1839
connectorstyle, 1840
deselect, 1840
deselect_all, 1841
display, 1841
displayed, 1841
draw, 1842
extend_selection, 1842
extend_selection_dir, 1842
find_clicked, 1843
find_item, 1843
first, 1844
first_selected_item, 1844
first_visible, 1844
first_visible_item, 1845
get_selected_items, 1845
handle, 1845
hposition, 1846
insert, 1846
insert_above, 1847
is_close, 1847, 1848
is_hscroll_visible, 1848
is_open, 1848
is_scrollbar, 1849
is_selected, 1849, 1850
is_vscroll_visible, 1850
item_clicked, 1850
item_draw_mode, 1850, 1851
item_labelbgcolor, 1851
item_labelfgcolor, 1851
item_labelfont, 1851
item_labelsize, 1851
item_pathname, 1851
item_reselect_mode, 1852
last, 1852
last_selected_item, 1853
last_visible, 1853
last_visible_item, 1853
load, 1853
next, 1854
next_item, 1854
next_selected_item, 1855
next_visible_item, 1855
open, 1856, 1857
open_toggle, 1857
openicon, 1858
prev, 1858
recalc_tree, 1858
remove, 1858
resize, 1859
root, 1859
root_label, 1859
scrollbar_size, 1859
select, 1860
select_all, 1861
select_only, 1861
select_toggle, 1862
selectbox, 1862
selectmode, 1862
set_item_focus, 1863
show_item, 1863
show_item_bottom, 1863
show_item_middle, 1864
show_item_top, 1864
show_self, 1864
showcollapse, 1864
showroot, 1865
sortorder, 1865
usericon, 1865
vposition, 1865
Fl_Tree.H, 2275
Fl_Tree_Reason, 2275
FL_TREE_REASON_CLOSED, 2275
FL_TREE_REASON_DESELECTED, 2275
FL_TREE_REASON_DRAGGED, 2275
FL_TREE_REASON_NONE, 2275
FL_TREE_REASON_OPENED, 2275
FL_TREE_REASON_RESELECTED, 2275
FL_TREE_REASON_SELECTED, 2275
Fl_Tree_Connector
Fl_Tree_Prefs.H, 2284
FL_TREE_CONNECTOR_DOTTED
Fl_Tree_Prefs.H, 2285
FL_TREE_CONNECTOR_NONE
Fl_Tree_Prefs.H, 2285
FL_TREE_CONNECTOR_SOLID

Fl_Tree_Prefs.H, 2285
Fl_Tree_Item, 1866
activate, 1872
add, 1872
calc_item_height, 1873
child, 1873
deactivate, 1873
deparent, 1873
depth, 1874
deselect_all, 1874
draw, 1874
draw_horizontal_connector, 1874
draw_item_content, 1875
draw_vertical_connector, 1876
drawbgcolor, 1876
drawfgcolor, 1876
find_child, 1876
find_child_item, 1877
find_clicked, 1877
find_item, 1877
Fl_Tree_Item, 1871
hide_widgets, 1878
insert, 1878
insert_above, 1878
is_visible_r, 1878
label, 1878
label_h, 1879
label_w, 1879
label_x, 1879
label_y, 1879
labelbgcolor, 1879
move, 1879, 1880
move_above, 1880
move_below, 1880
move_into, 1881
next, 1881
next_displayed, 1881
next_sibling, 1881
next_visible, 1881
parent, 1882
prefs, 1882
prev, 1882
prev_displayed, 1882
prev_sibling, 1882
prev_visible, 1882
recalc_tree, 1883
remove_child, 1883
reparent, 1883
replace, 1884
replace_child, 1884
select, 1885
select_all, 1885
show_self, 1885
show_widgets, 1885
swap_children, 1885
tree, 1886
update_prev_next, 1886
userdeicon, 1886
usericon, 1887
visible_r, 1887
Fl_Tree_Item.H, 2278, 2279
Fl_Tree_Item_Array, 1887
add, 1888
clear, 1888
deparent, 1888
Fl_Tree_Item_Array, 1888
insert, 1889
manage_item_destroy, 1889
move, 1889
remove, 1889
reparent, 1890
replace, 1890
Fl_Tree_Item_Array.H, 2283
FL_TREE_ITEM_DRAW_DEFAULT
Fl_Tree_Prefs.H, 2285
FL_TREE_ITEM_DRAW_LABEL_AND_WIDGET
Fl_Tree_Prefs.H, 2285
Fl_Tree_Item_Draw_Mode
Fl_Tree_Prefs.H, 2285
FL_TREE_ITEM_HEIGHT_FROM_WIDGET
Fl_Tree_Prefs.H, 2285
Fl_Tree_Item_Reselect_Mode
Fl_Tree_Prefs.H, 2285
Fl_Tree_Prefs, 1890
closedeicon, 1893
closeicon, 1893
item_draw_mode, 1893
item_labelbgcolor, 1893
marginbottom, 1894
opendeicon, 1894
openicon, 1894
selectmode, 1894
showcollapse, 1894
showroot, 1894
sortorder, 1895
userdeicon, 1895
Fl_Tree_Prefs.H, 2284, 2286
Fl_Tree_Connector, 2284
FL_TREE_CONNECTOR_DOTTED, 2285
FL_TREE_CONNECTOR_NONE, 2285
FL_TREE_CONNECTOR_SOLID, 2285
FL_TREE_ITEM_DRAW_DEFAULT, 2285
FL_TREE_ITEM_DRAW_LABEL_AND_WIDGET, 2285
Fl_Tree_Item_Draw_Mode, 2285
FL_TREE_ITEM_HEIGHT_FROM_WIDGET, 2285
Fl_Tree_Item_Reselect_Mode, 2285
Fl_Tree_Select, 2285
FL_TREE_SELECT_MULTI, 2285
FL_TREE_SELECT_NONE, 2285
FL_TREE_SELECT_SINGLE, 2285
FL_TREE_SELECT_SINGLE_DRAGGABLE, 2285
FL_TREE_SELECTABLE_ALWAYS, 2285
FL_TREE_SELECTABLE_ONCE, 2285
Fl_Tree_Sort, 2285

FL_TREE_SORT_ASCENDING, 2285
FL_TREE_SORT_DESCENDING, 2285
FL_TREE_SORT_NONE, 2285
Fl_Tree_Reason
Fl_Tree.H, 2275
FL_TREE_REASON_CLOSED
Fl_Tree.H, 2275
FL_TREE_REASON_DESELECTED
Fl_Tree.H, 2275
FL_TREE_REASON_DRAGGED
Fl_Tree.H, 2275
FL_TREE_REASON_NONE
Fl_Tree.H, 2275
FL_TREE_REASON_OPENED
Fl_Tree.H, 2275
FL_TREE_REASON_RESELECTED
Fl_Tree.H, 2275
FL_TREE_REASON_SELECTED
Fl_Tree.H, 2275
Fl_Tree_Select
Fl_Tree_Prefs.H, 2285
FL_TREE_SELECT_MULTI
Fl_Tree_Prefs.H, 2285
FL_TREE_SELECT_NONE
Fl_Tree_Prefs.H, 2285
FL_TREE_SELECT_SINGLE
Fl_Tree_Prefs.H, 2285
FL_TREE_SELECT_SINGLE_DRAGGABLE
Fl_Tree_Prefs.H, 2285
FL_TREE_SELECTABLE_ALWAYS
Fl_Tree_Prefs.H, 2285
FL_TREE_SELECTABLE_ONCE
Fl_Tree_Prefs.H, 2285
Fl_Tree_Sort
Fl_Tree_Prefs.H, 2285
FL_TREE_SORT_ASCENDING
Fl_Tree_Prefs.H, 2285
FL_TREE_SORT_DESCENDING
Fl_Tree_Prefs.H, 2285
FL_TREE_SORT_NONE
Fl_Tree_Prefs.H, 2285
fl_types.h, 2289, 2290
Fl_Shortcut, 2289
fl_ucs_to_Utf16
Unicode and UTF-8 functions, 358
fl_uintptr_t
platform_types.h, 2341
FL_UNFOCUS
Enumerations.H, 2079
FL_UNIFORM_WIDTH
Drawing functions, 312
fl_unlink
Unicode and UTF-8 functions, 359
FL_UP_BOX
Enumerations.H, 2075
FL_UP_FRAME
Enumerations.H, 2075
fl_utf2mbcs
Unicode and UTF-8 functions, 359
fl_utf8.h, 2290, 2293
fl_utf8_clean
Unicode and UTF-8 functions, 359
fl_utf8_next_composed_char
Unicode and UTF-8 functions, 360
fl_utf8_previous_composed_char
Unicode and UTF-8 functions, 360
fl_utf8back
Unicode and UTF-8 functions, 361
fl_utf8bytes
Unicode and UTF-8 functions, 361
fl_utf8decode
Unicode and UTF-8 functions, 361
fl_utf8encode
Unicode and UTF-8 functions, 362
fl_utf8from_mb
Unicode and UTF-8 functions, 362
fl_utf8froma
Unicode and UTF-8 functions, 363
fl_utf8fromwc
Unicode and UTF-8 functions, 363
fl_utf8fwd
Unicode and UTF-8 functions, 364
fl_utf8len
Unicode and UTF-8 functions, 364
fl_utf8len1
Unicode and UTF-8 functions, 365
fl_utf8locale
Unicode and UTF-8 functions, 365
fl_utf8strlen
Unicode and UTF-8 functions, 366
fl_utf8test
Unicode and UTF-8 functions, 366
fl_utf8to_mb
Unicode and UTF-8 functions, 367
fl_utf8toa
Unicode and UTF-8 functions, 367
fl_utf8toUtf16
Unicode and UTF-8 functions, 367
fl_utf8towc
Unicode and UTF-8 functions, 368
fl_utf_nb_char
Unicode and UTF-8 functions, 369
fl_utf_strcasecmp
Unicode and UTF-8 functions, 369
fl_utf_strncasecmp
Unicode and UTF-8 functions, 369
fl_utf_tolower
Unicode and UTF-8 functions, 370
fl_utf_toupper
Unicode and UTF-8 functions, 371
Fl_Valuator, 1895
Fl_Valuator, 1903
format, 1904
format_str, 1904
increment, 1904
maximum, 1904

minimum, 1904, 1905
precision, 1905
range, 1905
round, 1905
step, 1905
value, 1905
value_damage, 1906
Fl_Valuator.H, 2295
Fl_Value_Input, 1906
cursor_color, 1915
draw, 1915
Fl_Value_Input, 1915
handle, 1915
resize, 1916
shortcut, 1916
soft, 1917
textcolor, 1917
textfont, 1917
textsize, 1917
Fl_Value_Input.H, 2296
Fl_Value_Output, 1917
draw, 1926
Fl_Value_Output, 1926
handle, 1926
soft, 1927
textcolor, 1927
textfont, 1927
textsize, 1927
Fl_Value_Output.H, 2297
Fl_Value_Slider, 1928
draw, 1937
Fl_Value_Slider, 1937
handle, 1938
value_height, 1938
value_width, 1940
Fl_Value_Slider.H, 2297
FL_VERSION
Version Numbers, 248
fl_vertex
Drawing functions, 342
fl_vertex.cxx, 2391
fl_vsnprintf
vsnprintf.c, 2407
fl_wcwidth
Unicode and UTF-8 functions, 371
fl_wcwidth_
Unicode and UTF-8 functions, 371
Fl_When
Enumerations.H, 2083
FL_WHEN_CHANGED
Enumerations.H, 2083
FL_WHEN_CLOSED
Enumerations.H, 2083
FL_WHEN_ENTER_KEY
Enumerations.H, 2083
FL_WHEN_ENTER_KEY_ALWAYS
Enumerations.H, 2083
FL_WHEN_ENTER_KEY_CHANGED
Enumerations.H, 2083
FL_WHEN_NEVER
Enumerations.H, 2083
FL_WHEN_NOT_CHANGED
Enumerations.H, 2083
FL_WHEN_RELEASE
Enumerations.H, 2083
FL_WHEN_RELEASE_ALWAYS
Enumerations.H, 2083
Fl_Widget, 1940
~Fl_Widget, 1950
activate, 1950
active, 1950
active_r, 1950
align, 1951
argument, 1951
as_gl_window, 1951
as_group, 1952
as_window, 1952
AUTO_DELETE_USER_DATA, 1949
bind_deimage, 1953
bind_image, 1953, 1954
box, 1954
callback, 1954, 1955
CHANGED, 1949
changed, 1956
clear_active, 1956
clear_changed, 1956
clear_damage, 1956
clear_output, 1957
clear_visible, 1957
clear_visible_focus, 1957
CLIP_CHILDREN, 1949
color, 1957, 1958
contains, 1958
COPIED_LABEL, 1949
COPIED_TOOLTIP, 1949
copy_label, 1958
copy_tooltip, 1958
damage, 1959
deactivate, 1960
default_callback, 1960
deimage, 1960, 1961
DEIMAGE_BOUND, 1949
deimage_bound, 1961
do_callback, 1961, 1962
draw, 1963
draw_focus, 1963
draw_label, 1964
Fl_Widget, 1949
FORCE_POSITION, 1949
FULLSCREEN, 1949
GROUP_RELATIVE, 1949
h, 1964, 1965
handle, 1965
hide, 1965
horizontal_label_margin, 1966
image, 1966, 1967

IMAGE_BOUND, 1949
image_bound, 1967
INACTIVE, 1949
inside, 1967
INVISIBLE, 1949
is_label_copied, 1967
label, 1968
label_image_spacing, 1968
label_shortcut, 1969
labelcolor, 1969
labelfont, 1969, 1970
labelsize, 1970
labeltype, 1970, 1971
MAC_USE_ACCENTS_MENU, 1949
MAXIMIZED, 1949
measure_label, 1971
MENU_WINDOW, 1949
MODAL, 1949
NEEDS_KEYBOARD, 1949
needs_keyboard, 1971
NO_OVERLAY, 1949
NOBORDER, 1949
NON_MODAL, 1949
OUTPUT, 1949
output, 1972
OVERRIDE, 1949
parent, 1972
POPUP, 1949
position, 1972
redraw, 1973
redraw_label, 1973
resize, 1973
selection_color, 1973
send, 1974
set_active, 1974
set_changed, 1974
set_output, 1974
set_visible, 1974
set_visible_focus, 1975
SHORTCUT_LABEL, 1949
shortcut_label, 1975
show, 1975
size, 1975
take_focus, 1976
takesevents, 1976
test_shortcut, 1976
tooltip, 1977
TOOLTIP_WINDOW, 1949
top_window, 1977
top_window_offset, 1978
type, 1978
user_data, 1978
USERFLAG1, 1949
USERFLAG2, 1949
USERFLAG3, 1949
vertical_label_margin, 1978, 1979
visible, 1979
VISIBLE_FOCUS, 1949
visible_focus, 1979
visible_r, 1980
w, 1980
when, 1980
window, 1981
x, 1981
y, 1982
Fl_Widget.H, 2298, 2299
FL_RESERVED_TYPE, 2299
Fl_Widget_Surface, 1982
draw, 1984
draw_decorated_window, 1984
Fl_Widget_Surface, 1984
origin, 1984, 1985
print_window_part, 1985
printable_rect, 1985
translate, 1986
untranslate, 1986
Fl_Widget_Surface.H, 2304
Fl_Widget_Tracker, 1986
deleted, 1987
exists, 1987
widget, 1987
Fl_Widget_Tracker.H, 2305
fl_width
Color & Font functions, 303, 304
Fl_Window, 1988
~Fl_Window, 2001
allow_expand_outside_parent, 2001
as_double_window, 2001
as_overlay_window, 2001
as_window, 2001
border, 2002
clear_border, 2002
clear_modal_states, 2002
current, 2002
current_, 2021
cursor, 2002, 2003
decorated_h, 2003
decorated_w, 2003
default_cursor, 2004
default_icon, 2004
default_icons, 2004, 2005
default_size_range, 2005
default_xclass, 2006
draw, 2007
Fl_Window, 2000
flush, 2007
force_position, 2007
free_icons, 2007
free_position, 2008
fullscreen, 2008
fullscreen_screens, 2008
get_size_range, 2008
handle, 2009
hide, 2009
hotspot, 2010
icon, 2010

iconize, 2011
icons, 2011
is_a_rescale, 2011
is_resizable, 2012
make_current, 2012
maximize, 2012
modal, 2012
os_id, 2013
resize, 2013
screen_num, 2013, 2014
set_menu_window, 2014
set_modal, 2014
set_non_modal, 2015
set_tooltip_window, 2015
shape, 2015
show, 2016
show_next_window_iconic, 2018
shown, 2018
size_range, 2018
un_maximize, 2019
wait_for_expose, 2019
xclass, 2020
Fl_Window.H, 2306, 2307
Fl_Window_Driver.H, 2392
Fl_Wizard, 2021
draw, 2030
Fl_Wizard, 2029
next, 2030
value, 2030
Fl_Wizard.H, 2309
fl_wl_compositor
wayland.H, 2349
fl_wl_gc
wayland.H, 2349
FL_WRITE
Enumerations.H, 2074
fl_write_jpeg
fl_write_jpeg.cxx, 2394, 2395
fl_write_jpeg.cxx, 2394
fl_write_jpeg, 2394, 2395
fl_write_png
fl_write_png.cxx, 2396, 2397
fl_write_png.cxx, 2396
fl_write_png, 2396, 2397
fl_x11_find
x11.H, 2352
fl_x11_gc
x11.H, 2353
fl_x11_use_display
x11.H, 2353
fl_x11_xid
x11.H, 2353
Fl_XBM_Image, 2030
Fl_XBM_Image, 2033
Fl_XBM_Image.H, 2310
Fl_XColor.H, 2398
Fl_XPM_Image, 2033
Fl_XPM_Image, 2036
Fl_XPM_Image.H, 2310
FL_ZOOM_EVENT
Enumerations.H, 2081
FL_ZOOM_GESTURE
Enumerations.H, 2081
Flags
Fl_Anim_GIF_Image, 460
flstring.h, 2398
FLTK Basics, 10
FLTK Code Management Plan (CMP), 175
FLTK Programming Manual, 1
FLTK Runtime Options, 99
flush
Fl, 425
Fl_Double_Window, 690
Fl_Gl_Window, 832
Fl_Overlay_Window, 1219
Fl_Preferences, 1291
Fl_Single_Window, 1497
Fl_Window, 2007
focus
Events handling functions, 262
FOLIO
Fl_Paged_Device, 1234
font
Fl_Tooltip, 1814
FORCE_POSITION
Fl_Widget, 1949
force_position
Fl_Window, 2007
format
Fl_Valuator, 1904
format_char
Fl_Browser, 506
format_str
Fl_Valuator, 1904
Forms Compatibility, 121
forms.H, 2311
frame
Fl_Anim_GIF_Image, 464
frame_count
Fl_Anim_GIF_Image, 464
frame_h
Fl_Anim_GIF_Image, 464
frame_uncache
Fl_Anim_GIF_Image, 465
frame_w
Fl_Anim_GIF_Image, 465
frame_x
Fl_Anim_GIF_Image, 465
frame_y
Fl_Anim_GIF_Image, 466
frames
Fl_Anim_GIF_Image, 466
free_color
Color & Font functions, 304
free_icons
Fl_Window, 2007

free_position
Fl_Window, 2008
free_timeout
Fl_Timeout, 1794
freeglut_teapot_data.h, 2402
full_height
Fl_Browser, 507
Fl_Browser_, 532
full_width
Fl_Browser_, 533
FULLSCREEN
Fl_Widget, 1949
fullscreen
Fl_Window, 2008
fullscreen_screens
Fl_Window, 2008
function_types.H, 2054, 2055

g
Fl_Color_Chooser, 648
gap
Fl_Flex, 766
Fl_Grid, 869
get
Fl_Preferences, 1291–1294
Fl_Shared_Image, 1464, 1465
Fl_Timeout, 1790
get_absolute_top_line_number
Fl_Text_Display, 1716
get_color
Color & Font functions, 304
get_font
Color & Font functions, 305
get_font_name
Color & Font functions, 305
get_font_sizes
Color & Font functions, 305
get_key
Events handling functions, 262
get_mouse
Events handling functions, 262
get_selected_items
Fl_Tree, 1845
get_selection
Fl_Table, 1584
Fl_Terminal, 1658
Fl_Terminal::Selection, 2043
get_size_range
Fl_Window, 2008
get_system_colors
Fl, 425
get_userdata_path
Fl_Preferences, 1295
gl.h, 2321, 2324
gl_color, 2322
gl_draw, 2322, 2323
gl_font, 2323
gl_rect, 2323
gl_rectf, 2324
gl_texture_pile_height, 2324
gl2opengl.h, 2325
gl_color
gl.h, 2322
gl_draw
gl.h, 2322, 2323
gl_draw.H, 2326
gl_font
gl.h, 2323
gl_rect
gl.h, 2323
gl_rectf
gl.h, 2324
gl_texture_pile_height
gl.h, 2324
gl_visual
Fl, 425
GLContext
platform_types.h, 2341
global
Fl_Menu_, 1085
global_key_bindings
Fl_Text_Editor, 1761
glu.h, 2326
GLUT Compatibility, 118
glut.H, 2327
grab
Windows handling functions, 279
grammar_underline_color
Fl_Text_Display, 1716, 1717
group
Fl_Preferences, 1295
group_exists
Fl_Preferences, 1296
GROUP_RELATIVE
Fl_Widget, 1949
groups
Fl_Preferences, 1296

h
Fl_Image, 974
Fl_Widget, 1964, 1965
h_to_row
Fl_Terminal, 1659
handle
Events handling functions, 263
Fl_Adjuster, 453
Fl_Box, 487
Fl_Browser_, 533
Fl_Button, 553
Fl_Check_Browser, 594
Fl_Choice, 617
Fl_Clock, 627
Fl_Color_Chooser, 648
Fl_Counter, 664
Fl_Dial, 676
Fl_File_Input, 734
Fl_Free, 810
Fl_Gl_Window, 833

Fl_Glut_Window, 853
Fl_Group, 889
Fl_Help_View, 909
Fl_Input, 995
Fl_Light_Button, 1059
Fl_Menu_Bar, 1103
Fl_Menu_Button, 1114
Fl_Positioner, 1270
Fl_Repeat_Button, 1350
Fl_Return_Button, 1358
Fl_Roller, 1375
Fl_Scheme_Choice, 1404
Fl_Scroll, 1417
Fl_Scrollbar, 1430
Fl_Secret_Input, 1442
Fl_Shortcut_Button, 1475
Fl_Slider, 1508
Fl_Spinner, 1521
Fl_Spinner::Fl_Spinner_Input, 1533
Fl_Table, 1585
Fl_Table_Row, 1604
Fl_Tabs, 1621
Fl_Terminal, 1659
Fl_Text_Display, 1717
Fl_Text_Editor, 1756
Fl_Tile, 1777
Fl_Timer, 1802
Fl_Tree, 1845
Fl_Value_Input, 1915
Fl_Value_Output, 1926
Fl_Value_Slider, 1938
Fl_Widget, 1965
Fl_Window, 2009
handle_
Events handling functions, 263
handle_key
Fl_Input, 995
handle_mouse
Fl_Input_, 1009
handle_overflow
Fl_Tabs, 1621
handle_overflow_menu
Fl_Tabs, 1622
handle_rmb
Fl_Input, 996
Fl_Text_Display, 1717
handle_unknown_char
Fl_Terminal, 1659
handle_vline
Fl_Text_Display, 1717
handletext
Fl_Input_, 1010
Handling Events, 75
has_idle
Fl, 425
has_scrollbar
Fl_Browser_, 533
has_timeout
Fl, 426
Fl_Timeout, 1791
headline
Fl_Menu_Item, 1123
HEAVY_CURSOR
Fl_Text_Display, 1708
help
Fl, 438
hide
Fl_Browser, 507
Fl_Double_Window, 690
Fl_Gl_Window, 833
Fl_Overlay_Window, 1219
Fl_Widget, 1965
Fl_Window, 2009
hide_all_windows
Fl, 426
hide_widgets
Fl_Tree_Item, 1878
hidedelay
Fl_Tooltip, 1814
highlight_data
Fl_Text_Display, 1718
highlight_text
Fl_Text_Buffer, 1682
highres_image
Fl_Image_Surface, 980
history_lines
Fl_Terminal, 1659
history_use
Fl_Terminal, 1659
hit_close
Fl_Tabs, 1622
hit_overflow_menu
Fl_Tabs, 1622
hit_tabs_area
Fl_Tabs, 1624
HORIZONTAL
Fl_Browser_, 530
Fl_Flex, 763
horizontal
Fl_Flex, 767
Fl_Pack, 1230
HORIZONTAL_ALWAYS
Fl_Browser_, 530
horizontal_label_margin
Fl_Widget, 1966
hotspot
Fl_Window, 2010
hour
Fl_Clock_Output, 636
HOVER
Pen and tablet event handling, 268
hoverdelay
Fl_Tooltip, 1814
How Does Resizing Work?, 33
hposition
Fl_Browser_, 533, 534

Fl_Tree, 1846
hscrollbar
Fl_Browser_, 542
Fl_Terminal, 1672
hscrollbar_style
Fl_Terminal, 1659, 1660
hsv
Fl_Color_Chooser, 648
hsv2rgb
Fl_Color_Chooser, 649
hue
Fl_Color_Chooser, 649

icon
Fl_Browser, 507, 508
Fl_Window, 2010
iconize
Fl_Window, 2011
icons
Fl_Window, 2011
iconsize
Fl_File_Browser, 710
Fl_File_Chooser, 717
ID
Fl_Preferences, 1284
idle
Fl, 426
idle_
Fl_Private.H, 2380
image
Fl_Anim_GIF_Image, 466
Fl_Image_Surface, 980
Fl_Menu_Item, 1123
Fl_Shared_Image, 1465
Fl_Widget, 1966, 1967
IMAGE_BOUND
Fl_Widget, 1949
image_bound
Fl_Widget, 1967
image_label
Fl_Menu_Item, 1123
images
Fl_Shared_Image, 1466
IN_RANGE
Pen and tablet event handling, 268
in_selection
Fl_Text_Display, 1719
INACTIVE
Fl_Widget, 1949
inactive
Fl_Image, 974
includes
Fl_Text_Selection, 1763
incr_height
Fl_Browser, 508
Fl_Browser_, 534
increment
Fl_Valuator, 1904
increment_lin_log
Fl_Slider, 1509
index
Fl_Input_, 1010
init_size_range
Fl_Tile, 1777
init_sizes
Fl_Group, 889
Fl_Table, 1585
init_value
Fl_Scheme_Choice, 1405
inp_x
Fl_Input_Choice, 1034
input
Fl_Input_Choice, 1034
input_type
Fl_Input_, 1010
insert
Fl_Browser, 508
Fl_Chart, 581
Fl_Group, 889, 890
Fl_Input_, 1010
Fl_Menu_, 1085
Fl_Menu_Item, 1123
Fl_Sys_Menu_Bar, 1562
Fl_Table, 1585
Fl_Text_Buffer, 1682
Fl_Text_Display, 1719
Fl_Timeout, 1791
Fl_Tree, 1846
Fl_Tree_Item, 1878
Fl_Tree_Item_Array, 1889
insert_
Fl_Text_Buffer, 1682
insert_above
Fl_Tree, 1847
Fl_Tree_Item, 1878
insert_char
Fl_Terminal, 1660
insert_mode
Fl_Text_Editor, 1756
insert_position
Fl_Input_, 1011
Fl_Text_Display, 1719
insert_rows
Fl_Terminal, 1660
insertfile
Fl_Text_Buffer, 1683
inserting
Fl_Browser_, 534
inset
Fl_Rect, 1341
inside
Fl_Widget, 1967
Introduction to FLTK, 4
INVERSE
Fl_Terminal, 1650
INVISIBLE
Fl_Widget, 1949

is_a_rescale
Fl_Window, 2011
is_animated
Fl_Anim_GIF_Image, 466
is_close
Fl_Tree, 1847, 1848
is_current
Fl_Copy_Surface, 653
Fl_Image_Surface, 980
Fl_PDF_File_Surface, 1241
Fl_Printer, 1305
Fl_Surface_Device, 1535
is_global
Fl_Sys_Menu_Bar, 1562
is_hscroll_visible
Fl_Tree, 1848
is_inside_selection
Fl_Terminal, 1660
is_interactive_resize
Fl_Table, 1585
is_label_copied
Fl_Widget, 1967
is_open
Fl_Tree, 1848
is_resizable
Fl_Window, 2012
is_scheme
Fl, 427
is_scrollbar
Fl_Tree, 1849
is_selected
Fl_Table, 1585
Fl_Tree, 1849, 1850
is_visible_r
Fl_Tree_Item, 1878
is_vscroll_visible
Fl_Tree, 1850
is_word_separator
Fl_Text_Buffer, 1683
ITALIC
Fl_Terminal, 1650
item_at
Fl_Browser, 509
Fl_Browser_, 534
Fl_Check_Browser, 595
item_clicked
Fl_Tree, 1850
item_draw
Fl_Browser, 509
Fl_Browser_, 535
Fl_Check_Browser, 595
item_draw_mode
Fl_Tree, 1850, 1851
Fl_Tree_Prefs, 1893
item_first
Fl_Browser, 509
Fl_Browser_, 535
Fl_Check_Browser, 595
item_height
Fl_Browser, 510
Fl_Browser_, 535
Fl_Check_Browser, 595
item_labelbgcolor
Fl_Tree, 1851
Fl_Tree_Prefs, 1893
item_labelfgcolor
Fl_Tree, 1851
item_labelfont
Fl_Tree, 1851
item_labelsize
Fl_Tree, 1851
item_last
Fl_Browser, 510
Fl_Browser_, 535
item_next
Fl_Browser, 510
Fl_Browser_, 535
Fl_Check_Browser, 596
item_pathname
Fl_Menu_, 1086
Fl_Tree, 1851
item_prev
Fl_Browser, 511
Fl_Browser_, 536
Fl_Check_Browser, 596
item_quick_height
Fl_Browser_, 536
item_reselect_mode
Fl_Tree, 1852
item_select
Fl_Browser, 511
Fl_Browser_, 536
Fl_Check_Browser, 596
item_selected
Fl_Browser, 511
Fl_Browser_, 536
Fl_Check_Browser, 596
item_swap
Fl_Browser, 512
Fl_Browser_, 537
Fl_Check_Browser, 597
item_text
Fl_Browser, 512
Fl_Browser_, 537
Fl_Check_Browser, 597
item_width
Fl_Browser, 512
Fl_Browser_, 537
Fl_Check_Browser, 597
ivalue
Fl_Input_, 1012

keyboard_screen_scaling
Screen functions, 285
kf_backspace
Fl_Text_Editor, 1756
kf_c_s_move

Fl_Text_Editor, 1756
kf_copy
Fl_Text_Editor, 1756
kf_ctrl_move
Fl_Text_Editor, 1757
kf_cut
Fl_Text_Editor, 1757
kf_default
Fl_Text_Editor, 1757
kf_delete
Fl_Text_Editor, 1757
kf_down
Fl_Text_Editor, 1757
kf_end
Fl_Text_Editor, 1757
kf_enter
Fl_Text_Editor, 1757
kf_home
Fl_Text_Editor, 1758
kf_ignore
Fl_Text_Editor, 1758
kf_insert
Fl_Text_Editor, 1758
kf_left
Fl_Text_Editor, 1758
kf_m_s_move
Fl_Text_Editor, 1758
kf_meta_move
Fl_Text_Editor, 1758
kf_move
Fl_Text_Editor, 1759
kf_page_down
Fl_Text_Editor, 1759
kf_page_up
Fl_Text_Editor, 1759
kf_paste
Fl_Text_Editor, 1759
kf_redo
Fl_Text_Editor, 1759
kf_right
Fl_Text_Editor, 1759
kf_select_all
Fl_Text_Editor, 1759
kf_shift_move
Fl_Text_Editor, 1760
kf_undo
Fl_Text_Editor, 1760
kf_up
Fl_Text_Editor, 1760
klass_list
Fl_Plugin_Manager, 1254

label
Fl_Bitmap, 474
Fl_File_Icon, 721
Fl_Image, 974, 975
Fl_Menu_Item, 1124, 1125
Fl_Multi_Label, 1158
Fl_Pixmap, 1248, 1249
Fl_RGB_Image, 1365
Fl_Tree_Item, 1878
Fl_Widget, 1968
label_h
Fl_Tree_Item, 1879
label_image_spacing
Fl_Widget, 1968
label_shortcut
Fl_Widget, 1969
label_w
Fl_Tree_Item, 1879
label_x
Fl_Tree_Item, 1879
label_y
Fl_Tree_Item, 1879
labela
Fl_Multi_Label, 1158
labelb
Fl_Multi_Label, 1158
labelbgcolor
Fl_Tree_Item, 1879
labelcolor
Fl_Menu_Item, 1125
Fl_Widget, 1969
labelfont
Fl_Menu_Item, 1125
Fl_Widget, 1969, 1970
labelsize
Fl_Widget, 1970
labeltype
Fl_File_Icon, 721
Fl_Menu_Item, 1125, 1126
Fl_Widget, 1970, 1971
LANDSCAPE
Fl_Paged_Device, 1234
last
Fl_Tree, 1852
last_handler
Events handling functions, 264
last_selected_item
Fl_Tree, 1853
last_visible
Fl_Tree, 1853
last_visible_item
Fl_Tree, 1853
layout
Fl_Flex, 767
Fl_Grid, 870
ld
Fl_Image, 975
LEAVE
Pen and tablet event handling, 268
LEDGER
Fl_Paged_Device, 1234
leftedge
Fl_Browser_, 538
leftline
Fl_Help_View, 910

LEGAL
Fl_Paged_Device, 1234
length
Fl_Text_Buffer, 1683
Fl_Text_Selection, 1763
LETTER
Fl_Paged_Device, 1234
LF_TO_CR
Fl_Terminal, 1651
LF_TO_CRLF
Fl_Terminal, 1651
LIFT
Pen and tablet event handling, 268
line_end
Fl_Input_, 1012
Fl_Text_Buffer, 1683
Fl_Text_Display, 1720
line_start
Fl_Input_, 1012
Fl_Text_Buffer, 1684
Fl_Text_Display, 1720
line_text
Fl_Text_Buffer, 1684
LINEAR_SCALE
Fl_Slider, 1507
lineno
Fl_Browser, 513
linenumber_align
Fl_Text_Display, 1720
linenumber_bgcolor
Fl_Text_Display, 1721
linenumber_fgcolor
Fl_Text_Display, 1721
linenumber_font
Fl_Text_Display, 1721
linenumber_format
Fl_Text_Display, 1721
linenumber_size
Fl_Text_Display, 1721
linenumber_width
Fl_Text_Display, 1721
lineposition
Fl_Browser, 513
linesize
Fl_Scrollbar, 1431
linespacing
Fl_Browser_, 538
link
Fl_Help_View, 910
load
Fl_Anim_GIF_Image, 467
Fl_Browser, 513
Fl_File_Browser, 711
Fl_File_Icon, 722
Fl_Help_Dialog, 896
Fl_Help_View, 910
Fl_Plugin_Manager, 1254
Fl_Tree, 1853
load_fti
Fl_File_Icon, 722
load_image
Fl_File_Icon, 722
load_system_icons
Fl_File_Icon, 723
loadAll
Fl_Plugin_Manager, 1254
loadfile
Fl_Text_Buffer, 1684
lock
Multithreading support functions, 344
LOG_FLAG
Fl_Anim_GIF_Image, 460
LOG_SCALE
Fl_Slider, 1507
longest_vline
Fl_Text_Display, 1722
loop
Fl_Anim_GIF_Image, 469
lstep
Fl_Counter, 665

Mac OS X-specific symbols, 375
fl_mac_os_version, 376
fl_mac_set_about, 376
fl_open_callback, 376
mac.H, 2333, 2334
MAC_USE_ACCENTS_MENU
Fl_Widget, 1949
maintain_absolute_top_line_number
Fl_Text_Display, 1722
maintaining_absolute_top_line_number
Fl_Text_Display, 1722
make_current
Fl_Gl_Window, 833
Fl_Timeout, 1792
Fl_Window, 2012
make_overlay_current
Fl_Gl_Window, 833
make_visible
Fl_Browser, 514
manage_item_destroy
Fl_Tree_Item_Array, 1889
margin
Fl_Flex, 767, 768
Fl_Grid, 870, 871
margin_height
Fl_Tooltip, 1815
margin_width
Fl_Tooltip, 1815
marginbottom
Fl_Tree_Prefs, 1894
margins
Fl_Paged_Device, 1235
Fl_PDF_File_Surface, 1241
Fl_PostScript_File_Device, 1276
Fl_Printer, 1305
mark

Fl_Input_, 1013
mask
Fl_Image_Surface, 980
math.h, 2335
max_size
Fl_RGB_Image, 1366
maximize
Fl_Window, 2012
MAXIMIZED
Fl_Widget, 1949
maximum
Fl_Progress, 1315
Fl_Valuator, 1904
maximum_size
Fl_Input_, 1013
maxsize
Fl_Chart, 581
maybe_do_callback
Fl_Tabs, 1624
measure
Fl_Label, 1051
Fl_Menu_Item, 1126
measure_deleted_lines
Fl_Text_Display, 1722
measure_label
Fl_Widget, 1971
measure_proportional_character
Fl_Text_Display, 1722
measure_vline
Fl_Text_Display, 1723
mediumarrow.h, 2403
MEMORY
Fl_Preferences, 1284
menu
Fl_Menu_, 1086, 1087
Fl_Sys_Menu_Bar, 1562
menu_box
Fl_Menu_, 1087
menu_end
Fl_Menu_, 1087
menu_linespacing
Fl, 427
MENU_WINDOW
Fl_Widget, 1949
menu_x
Fl_Input_Choice, 1034
menubutton
Fl_Input_Choice, 1034
middleline
Fl_Browser, 514
Migrating Code from FLTK 1.3 to 1.4, 142
Migrating Code from FLTK 1.4 to 1.5, 148
min_delay
Fl_Anim_GIF_Image, 469
minimum
Fl_Progress, 1315
Fl_Valuator, 1904, 1905
minute
Fl_Clock_Output, 636
MODAL
Fl_Widget, 1949
modal
Fl_Window, 2012
Windows handling functions, 279
mode
Fl_Color_Chooser, 649
Fl_Gl_Window, 833
Fl_Menu_, 1088
Fl_Sys_Menu_Bar, 1563
move
Fl_Browser, 514
Fl_Tree_Item, 1879, 1880
Fl_Tree_Item_Array, 1889
move_above
Fl_Tree_Item, 1880
move_below
Fl_Tree_Item, 1880
move_cursor
Fl_Table, 1585
move_down
Fl_Text_Display, 1723
move_intersection
Fl_Tile, 1778
move_into
Fl_Tree_Item, 1881
move_left
Fl_Text_Display, 1723
move_right
Fl_Text_Display, 1723
move_up
Fl_Text_Display, 1724
mTabDist
Fl_Text_Buffer, 1690
MULTI
Fl_File_Chooser, 716
multi_label
Fl_Menu_Item, 1126
Multithreading support functions, 342
awake, 343
awake_once, 344
lock, 344
thread_message, 344
unlock, 345
mvalue
Fl_Menu_, 1088

Name
Fl_Preferences::Name, 2038
name
Fl_Anim_GIF_Image, 467
names
Fl_Scheme, 1393
names.h, 2336, 2337
need_layout
Fl_Flex, 769
Fl_Grid, 871
NEEDS_KEYBOARD

Fl_Widget, 1949
needs_keyboard
Fl_Widget, 1971
NEW_FOLDER
Fl_Native_File_Chooser, 1182
new_list
Fl_Browser_, 538
new_UUID
Fl_Preferences, 1296
next
Fl_Anim_GIF_Image, 467
Fl_File_Icon, 723
Fl_Grid::Cell, 442
Fl_Menu_Item, 1126
Fl_Tree, 1854
Fl_Tree_Item, 1881
Fl_Wizard, 2030
next_char
Fl_Text_Buffer, 1684
next_displayed
Fl_Tree_Item, 1881
next_item
Fl_Tree, 1854
next_selected_item
Fl_Tree, 1855
next_sibling
Fl_Tree_Item, 1881
next_visible
Fl_Tree_Item, 1881
next_visible_item
Fl_Tree, 1855
next_window
Windows handling functions, 279
nitems
Fl_Check_Browser, 597
NO_OPTIONS
Fl_Native_File_Chooser, 1182
NO_OVERLAY
Fl_Widget, 1949
NO_REDRAW
Fl_Terminal, 1651
no_window_menu
Fl_Sys_Menu_Bar, 1559
NOBORDER
Fl_Widget, 1949
NON_MODAL
Fl_Widget, 1949
NONE
Fl_Preferences, 1299
Pen and tablet event handling, 268, 269
NORMAL
Fl_Terminal, 1650
NORMAL_CURSOR
Fl_Text_Display, 1707
normalize
Fl_RGB_Image, 1366
Fl_SVG_Image, 1547
normalized_screen_scale
Screen functions, 286
now
Fl, 428
num_images
Fl_Shared_Image, 1466
num_schemes
Fl_Scheme, 1394
num_ticks
Fl_Slider, 1509
numericsort.c, 2404
fl_casenumericsort, 2404
fl_numericsort, 2404

OFF
Fl_Terminal, 1651
offscreen
Fl_Image_Surface, 981
offset_line_starts
Fl_Text_Display, 1724
on_extension_data
Fl_Anim_GIF_Image, 467
on_frame_data
Fl_Anim_GIF_Image, 467
on_insert
Fl_Group, 890
Fl_Scroll, 1417
Fl_Tabs, 1624
Fl_Tile, 1778
on_move
Fl_Group, 890
Fl_Scroll, 1418
Fl_Tabs, 1624
Fl_Tile, 1778
on_remove
Fl_Flex, 769
Fl_Grid, 872
Fl_Group, 891
Fl_Tabs, 1624
Fl_Tile, 1778
open
Fl_Tree, 1856, 1857
open_toggle
Fl_Tree, 1857
opendeicon
Fl_Tree_Prefs, 1894
openicon
Fl_Tree, 1858
Fl_Tree_Prefs, 1894
Operating System Issues, 126
operator&
Pen and tablet event handling, 272
operator |
Pen and tablet event handling, 274
operator | =
Pen and tablet event handling, 274, 275
OPTIMIZE_MEMORY
Fl_Anim_GIF_Image, 460
Option
Fl_Native_File_Chooser, 1182

option
Fl, 428, 429
OPTION_ARROW_FOCUS
Fl, 415
OPTION_DND_TEXT
Fl, 415
OPTION_FNFC_USES_GTK
Fl, 415
OPTION_FNFC_USES_KDIALOG
Fl, 415
OPTION_FNFC_USES_ZENITY
Fl, 415
OPTION_LAST
Fl, 415
OPTION_PRINTER_USES_GTK
Fl, 415
OPTION_SHOW_SCALING
Fl, 415
OPTION_SHOW_TOOLTIPS
Fl, 415
OPTION_SIMPLE_ZOOM_SHORTCUT
Fl, 415
OPTION_VISIBLE_FOCUS
Fl, 415
options
Fl_Native_File_Chooser, 1185
options.H, 2056, 2057
ORIENTATION
Fl_Paged_Device, 1234
origin
Fl_Copy_Surface, 653
Fl_EPS_File_Surface, 694, 695
Fl_Image_Surface, 981, 982
Fl_PDF_File_Surface, 1242
Fl_PostScript_File_Device, 1277
Fl_Printer, 1306
Fl_SVG_File_Surface, 1539
Fl_Widget_Surface, 1984, 1985
original
Fl_Shared_Image, 1466
ortho
Fl_Gl_Window, 834
os_id
Fl_Window, 2013
OUT_OF_RANGE
Pen and tablet event handling, 268
OutFlags
Fl_Terminal, 1651
OUTPUT
Fl_Widget, 1949
output
Fl_Widget, 1972
output_translate
Fl_Terminal, 1660
outputfile
Fl_Text_Buffer, 1685
OVERFLOW_CLIP
Fl_Tabs, 1619
OVERFLOW_COMPRESS
Fl_Tabs, 1619
OVERFLOW_DRAG
Fl_Tabs, 1619
OVERFLOW_PULLDOWN
Fl_Tabs, 1619
overflow_type
Fl_Tabs, 1628
OVERRIDE
Fl_Widget, 1949
override_text
Fl_Tooltip, 1815
overstrike
Fl_Text_Display, 1724
own_colormap
Fl, 430

Page_Format
Fl_Paged_Device, 1233
Page_Layout
Fl_Paged_Device, 1234
parent
Fl_Tree_Item, 1882
Fl_Widget, 1972
paste
Selection & Clipboard functions, 282
Pen and tablet event handling, 265
ANY_DOWN, 268
BARREL_PRESSURE, 269
BUTTON0, 268
BUTTON1, 268
BUTTON2, 268
BUTTON3, 268
BUTTON_PUSH, 268
BUTTON_RELEASE, 268
CHANGED, 268
DETECTED, 268, 269
DISPLAY, 269
DRAW, 268
DRIVER_AVAILABLE, 269
driver_traits, 269
ENTER, 268
ERASER, 269
ERASER_DOWN, 268
ERASER_HOVERS, 268
Event, 267
event_barrel_pressure, 270
event_pen_id, 270
event_pressure, 270
event_proximity, 270
event_state, 270, 271
event_tilt_x, 271
event_trigger, 271
event_twist, 271
event_x, 272
event_x_root, 272
HOVER, 268
IN_RANGE, 268
LEAVE, 268

LIFT, 268
NONE, 268, 269
operator&, 272
operator|, 274
operator|=, 274, 275
OUT_OF_RANGE, 268
PEN_ID, 269
pen_traits, 275
PRESSURE, 269
PROXIMITY, 269
release, 276
State, 268
subscribe, 276
TILT_X, 269
TILT_Y, 269
TIP_DOWN, 268
TIP_HOVERS, 268
TOUCH, 268
Trait, 268
TWIST, 269
unsubscribe, 276
pen_events.H, 2058, 2060
PEN_ID
Pen and tablet event handling, 269
pen_traits
Pen and tablet event handling, 275
PER_WRITE
Fl_Terminal, 1651
picked
Fl_Menu_, 1088
pixel_h
Fl_Gl_Window, 834
pixel_w
Fl_Gl_Window, 835
pixels_per_unit
Fl_Gl_Window, 835
Pixmap
Fl_FormsPixmap, 795
placeholder
Fl_Input_, 1013
plastic_color_average
Fl_Scheme, 1394
platform.H, 2338
platform_types.h, 2339, 2341
fl_intptr_t, 2340
Fl_Offscreen, 2340
Fl_Region, 2340
Fl_Timestamp, 2340
fl_uintptr_t, 2341
GLContext, 2341
play_menu
Fl_Menu_Bar, 1103
Fl_Sys_Menu_Bar, 1563
playing
Fl_Anim_GIF_Image, 467
plot_char
Fl_Terminal, 1661
pop_current
Fl_Surface_Device, 1535
POPUP
Fl_Widget, 1949
popup
Fl_Menu_Button, 1115
Fl_Menu_Item, 1127
POPUP1
Fl_Menu_Button, 1114
POPUP12
Fl_Menu_Button, 1114
POPUP123
Fl_Menu_Button, 1114
POPUP13
Fl_Menu_Button, 1114
POPUP2
Fl_Menu_Button, 1114
POPUP23
Fl_Menu_Button, 1114
POPUP3
Fl_Menu_Button, 1114
popup_buttons
Fl_Menu_Button, 1113
PORTRAIT
Fl_Paged_Device, 1234
position
Fl_Browser_, 538
Fl_Input_, 1014
Fl_Text_Selection, 1763
Fl_Tile, 1778
Fl_Widget, 1972
position_style
Fl_Text_Display, 1724
position_to_line
Fl_Text_Display, 1725
position_to_linecol
Fl_Text_Display, 1725
position_to_value
Fl_Slider, 1509
position_to_xy
Fl_Text_Display, 1726
precision
Fl_Valuator, 1905
Preface, 2
prefs
Fl_Tree_Item, 1882
preset_file
Fl_Native_File_Chooser, 1185
PRESSURE
Pen and tablet event handling, 269
prev
Fl_Tree, 1858
Fl_Tree_Item, 1882
prev_char
Fl_Text_Buffer, 1685
prev_displayed
Fl_Tree_Item, 1882
prev_mvalue
Fl_Menu_, 1089

prev_sibling
Fl_Tree_Item, 1882
prev_visible
Fl_Tree_Item, 1882
PREVIEW
Fl_Native_File_Chooser, 1182
preview
Fl_File_Chooser, 717
print
Fl_Mac_App_Menu, 1069
print_button.h, 2405
print_char
Fl_Terminal, 1661, 1662
print_panel.h, 2406
print_window_part
Fl_Widget_Surface, 1985
printable_rect
Fl_Copy_Surface, 654
Fl_EPS_File_Surface, 695
Fl_Image_Surface, 982
Fl_PDF_File_Surface, 1242
Fl_PostScript_File_Device, 1277
Fl_Printer, 1306
Fl_SVG_File_Surface, 1539
Fl_Widget_Surface, 1985
printf
Fl_Terminal, 1662
Fl_Text_Buffer, 1685
program_should_quit
Fl, 430
proportional
Fl_SVG_Image, 1548
PROXIMITY
Pen and tablet event handling, 269
pulldown
Fl_Menu_Item, 1127
push
Fl_Tabs, 1625
push_current
Fl_Surface_Device, 1535
pushed
Events handling functions, 264

r
Fl_Color_Chooser, 650
Fl_Rect, 1341
radio
Fl_Menu_Item, 1128
range
Fl_Valuator, 1905
RATE_LIMITED
Fl_Terminal, 1651
readonly
Fl_Input_, 1014
readqueue
Fl, 430
ready
Fl, 430
recalc_dimensions
Fl_Table, 1586
recalc_display
Fl_Text_Display, 1726
recalc_scrollbars
Fl_Scroll, 1418
recalc_tree
Fl_Tree, 1858
Fl_Tree_Item, 1883
rectangle_capture
Fl_Device_Plugin, 666
redisplay_range
Fl_Text_Display, 1726
redo
Fl_Input_, 1014
redraw
Fl_Widget, 1973
redraw_label
Fl_Widget, 1973
redraw_line
Fl_Browser_, 538
redraw_lines
Fl_Browser_, 539
redraw_overlay
Fl_Gl_Window, 835
Fl_Overlay_Window, 1219
redraw_range
Fl_Table, 1586
redraw_rate
Fl_Terminal, 1662
redraw_style
Fl_Terminal, 1663
redraw_tabs
Fl_Tabs, 1625
RedrawStyle
Fl_Terminal, 1651
refcount
Fl_Shared_Image, 1466
release
Fl_Image, 975
Fl_Shared_Image, 1467
Fl_Timeout, 1792
Pen and tablet event handling, 276
release_widget_pointer
Safe widget deletion support functions, 374
reload_scheme
Fl, 431
remove
Fl_Browser, 514
Fl_Check_Browser, 598
Fl_Group, 891
Fl_Menu_, 1089
Fl_Sys_Menu_Bar, 1563
Fl_Text_Buffer, 1685
Fl_Tree, 1858
Fl_Tree_Item_Array, 1889
remove_
Fl_Text_Buffer, 1686
remove_check

Fl, 431
remove_child
Fl_Tree_Item, 1883
remove_handler
Events handling functions, 264
remove_icon
Fl_Browser, 515
remove_idle
Fl, 431
remove_key_binding
Fl_Text_Editor, 1760
remove_next_timeout
Fl, 431
Fl_Timeout, 1792
remove_system_handler
Events handling functions, 264
remove_timeout
Fl, 432
Fl_Timeout, 1793
removePlugin
Fl_Plugin_Manager, 1255
reparent
Fl_Tree_Item, 1883
Fl_Tree_Item_Array, 1890
repeat_timeout
Fl, 433
Fl_Timeout, 1793
replace
Fl_Chart, 581
Fl_Input_, 1015
Fl_Menu_, 1089
Fl_Sys_Menu_Bar, 1563
Fl_Text_Buffer, 1686
Fl_Tree_Item, 1884
Fl_Tree_Item_Array, 1890
replace_child
Fl_Tree_Item, 1884
replacing
Fl_Browser_, 539
request_grow_b
Fl_Tile, 1779
request_grow_l
Fl_Tile, 1779
request_grow_r
Fl_Tile, 1779
request_grow_t
Fl_Tile, 1779
request_shrink_b
Fl_Tile, 1780
request_shrink_l
Fl_Tile, 1780
request_shrink_r
Fl_Tile, 1780
request_shrink_t
Fl_Tile, 1781
rescale
Fl_Image_Surface, 982
reset_absolute_top_line_number
Fl_Text_Display, 1726
reset_terminal
Fl_Terminal, 1663
resizable
Fl_Group, 891, 892
resize
Fl_Anim_GIF_Image, 468
Fl_Browser_, 539
Fl_Double_Window, 690
Fl_Flex, 769
Fl_Gl_Window, 835
Fl_Grid, 872
Fl_Group, 893
Fl_Help_View, 911
Fl_Input_, 1015
Fl_Input_Choice, 1034
Fl_Overlay_Window, 1219
Fl_Pack, 1230
Fl_Scroll, 1419
Fl_Spinner, 1522
Fl_SVG_Image, 1547
Fl_Table, 1586
Fl_Tabs, 1625
Fl_Terminal, 1663
Fl_Text_Display, 1727
Fl_Tile, 1781
Fl_Tree, 1859
Fl_Value_Input, 1916
Fl_Widget, 1973
Fl_Window, 2013
REVERSED
Fl_Paged_Device, 1234
rewind_lines
Fl_Text_Buffer, 1686
Fl_Text_Display, 1727
rgb
Fl_Color_Chooser, 650
rgb2hsv
Fl_Color_Chooser, 650
rgb_colors.H, 2342
RGB_scaling
Fl_Image, 975
Root
Fl_Preferences, 1284
root
Fl_Tree, 1859
root_label
Fl_Tree, 1859
ROOT_MASK
Fl_Preferences, 1284
rotate
Fl_Paged_Device, 1236
Fl_PDF_File_Surface, 1243
Fl_PostScript_File_Device, 1278
Fl_Printer, 1306
round
Fl_Valuator, 1905
ROW

Fl_Flex, 763
row_col_clamp
Fl_Table, 1586
row_gap
Fl_Grid, 872
row_header
Fl_Table, 1586
row_height
Fl_Grid, 872, 873
Fl_Table, 1586
row_height_all
Fl_Table, 1587
row_resize
Fl_Table, 1587
row_resize_min
Fl_Table, 1587
row_selected
Fl_Table_Row, 1604
row_weight
Fl_Grid, 873
rows
Fl_Table, 1587
Fl_Table_Row, 1605
run
Fl, 433

Safe widget deletion support functions, 372
clear_widget_pointer, 373
delete_widget, 373
do_widget_deletion, 374
release_widget_pointer, 374
watch_widget_pointer, 374
saturation
Fl_Color_Chooser, 650
SAVEAS_CONFIRM
Fl_Native_File_Chooser, 1182
savefile
Fl_Text_Buffer, 1686
scale
Fl_Image, 975
Fl_Paged_Device, 1236
Fl_PDF_File_Surface, 1243
Fl_PostScript_File_Device, 1278
Fl_Printer, 1307
Fl_Slider, 1509
Fl_SVG_Image, 1547
Scale_Type
Fl_Slider, 1507
scaling_algorithm
Fl_Image, 976
scheme
Fl, 433
scheme_cb_
Fl_Scheme_Choice, 1405
Screen functions, 284
keyboard_screen_scaling, 285
normalized_screen_scale, 286
screen_count, 286
screen_dpi, 286
screen_num, 287
screen_scale, 288
screen_scaling_supported, 288
screen_work_area, 289
screen_xywh, 290, 291
screen_count
Screen functions, 286
screen_dpi
Screen functions, 286
screen_num
Fl_Window, 2013, 2014
Screen functions, 287
screen_scale
Screen functions, 288
screen_scaling_supported
Screen functions, 288
screen_work_area
Screen functions, 289
screen_xywh
Screen functions, 290, 291
scroll
Fl_Terminal, 1664
Fl_Text_Display, 1727
scroll_
Fl_Text_Display, 1727
scroll_timer_cb
Fl_Text_Display, 1729
scroll_to
Fl_Scroll, 1419
scrollbar
Fl_Browser_, 542
Fl_Terminal, 1672
scrollbar_actual_size
Fl_Terminal, 1664
scrollbar_align
Fl_Text_Display, 1729
SCROLLBAR_AUTO
Fl_Terminal, 1651
scrollbar_left
Fl_Browser_, 539
SCROLLBAR_OFF
Fl_Terminal, 1651
SCROLLBAR_ON
Fl_Terminal, 1651
scrollbar_right
Fl_Browser_, 540
scrollbar_size
Fl, 434
Fl_Browser_, 540
Fl_Help_View, 911
Fl_Scroll, 1419, 1420
Fl_Table, 1587
Fl_Terminal, 1664
Fl_Text_Display, 1729
Fl_Tree, 1859
scrollbar_width
Fl_Text_Display, 1730
ScrollbarStyle

Fl_Terminal, 1651
scrollvalue
Fl_Slider, 1510
search_backward
Fl_Text_Buffer, 1687
search_forward
Fl_Text_Buffer, 1687
second
Fl_Clock_Output, 636
secondary_selection_color
Fl_Text_Display, 1730
secondary_selection_text
Fl_Text_Buffer, 1687
seconds_between
Fl, 434
seconds_since
Fl, 435
select
Fl_Browser, 515
Fl_Browser_, 540
Fl_Tree, 1860
Fl_Tree_Item, 1885
select_all
Fl_Tree, 1861
Fl_Tree_Item, 1885
select_all_rows
Fl_Table_Row, 1605
select_only
Fl_Browser_, 541
Fl_Tree, 1861
select_row
Fl_Table_Row, 1605
select_toggle
Fl_Tree, 1862
selectable
Fl_Menu_Item, 1128
selectbox
Fl_Tree, 1862
selected
Fl_Browser, 515
Fl_Text_Selection, 1763, 1764
selection
Fl_Browser_, 541
Selection & Clipboard functions, 283
Selection & Clipboard functions, 280
add_clipboard_notify, 281
clipboard_contains, 281
copy, 281
dnd, 282
paste, 282
selection, 283
selection_owner, 283
selection_to_clipboard, 283
selection_color
Fl_Widget, 1973
selection_extend
Fl_Terminal, 1665
selection_owner
Selection & Clipboard functions, 283
selection_text
Fl_Terminal, 1665
Fl_Text_Buffer, 1688
selection_text_len
Fl_Terminal, 1665
selection_to_clipboard
Selection & Clipboard functions, 283
selectmode
Fl_Tree, 1862
Fl_Tree_Prefs, 1894
send
Fl_Widget, 1974
set
Fl_Button, 553
Fl_FormsBitmap, 787
Fl_FormsPixmap, 795
Fl_Menu_Item, 1128
Fl_Preferences, 1296–1299
Fl_Text_Selection, 1764
set_active
Fl_Widget, 1974
set_atclose
Windows handling functions, 279
set_box_color
Fl, 435
set_boxtype
Fl, 435
set_changed
Fl_Widget, 1974
set_color
Color & Font functions, 305, 306
set_current
Fl_Copy_Surface, 654
Fl_Image_Surface, 982
Fl_PDF_File_Surface, 1243
Fl_PostScript_File_Device, 1278
Fl_Printer, 1307
Fl_Surface_Device, 1535
set_cursor
Fl_Tile, 1781
set_draw_cb
Fl_Cairo_Window, 569
set_font
Color & Font functions, 306
set_fonts
Color & Font functions, 306
set_idle_
Fl_Private.H, 2379
set_item_focus
Fl_Tree, 1863
set_menu_window
Fl_Window, 2014
set_modal
Fl_Window, 2014
set_non_modal
Fl_Window, 2015
set_output

Fl_Widget, 1974
set_selection
Fl_Table, 1588
set_tooltip_window
Fl_Window, 2015
set_visible
Fl_Widget, 1974
set_visible_focus
Fl_Widget, 1975
setonly
Fl_Menu_Item, 1128
shadow
Fl_Clock_Output, 636, 637
shape
Fl_Window, 2015
shortcut
Fl_Button, 554
Fl_Input_, 1016
Fl_Menu_Item, 1128
Fl_Text_Display, 1730, 1731
Fl_Value_Input, 1916
SHORTCUT_LABEL
Fl_Widget, 1949
shortcut_label
Fl_Widget, 1975
show
Fl_Browser, 516
Fl_Double_Window, 691
Fl_Gl_Window, 836
Fl_Help_Dialog, 896
Fl_Native_File_Chooser, 1185
Fl_Overlay_Window, 1220
Fl_Single_Window, 1497
Fl_Tabs, 1625
Fl_Widget, 1975
Fl_Window, 2016
show_cursor
Fl_Text_Display, 1731
show_grid
Fl_Grid, 873, 874
show_insert_position
Fl_Text_Display, 1731
show_item
Fl_Tree, 1863
show_item_bottom
Fl_Tree, 1863
show_item_middle
Fl_Tree, 1864
show_item_top
Fl_Tree, 1864
show_next_window_iconic
Fl_Window, 2018
show_self
Fl_Tree, 1864
Fl_Tree_Item, 1885
show_unknown
Fl_Terminal, 1665
show_widgets
Fl_Tree_Item, 1885
showcollapse
Fl_Tree, 1864
Fl_Tree_Prefs, 1894
showHiddenButton
Fl_File_Chooser, 718
shown
Fl_File_Chooser, 717
Fl_Window, 2018
showroot
Fl_Tree, 1865
Fl_Tree_Prefs, 1894
SIMPLE_CURSOR
Fl_Text_Display, 1708
SINGLE
Fl_File_Chooser, 716
size
Fl_Browser, 516
Fl_Chart, 582
Fl_Input_, 1016
Fl_Menu_, 1089
Fl_Menu_Item, 1128
Fl_Preferences, 1299
Fl_Tooltip, 1815
Fl_Widget, 1975
size_range
Fl_Tile, 1782
Fl_Window, 2018
sizes
Fl_Group, 893
skip_displayed_characters
Fl_Text_Buffer, 1688
skip_lines
Fl_Text_Display, 1731
slider_size
Fl_Slider, 1510
slowarrow.h, 2406
soft
Fl_Adjuster, 454
Fl_Value_Input, 1917
Fl_Value_Output, 1927
Software License, 149
sort
Fl_Browser_, 541
sortorder
Fl_Tree, 1865
Fl_Tree_Prefs, 1895
spacing
Fl_Flex, 770
speed
Fl_Anim_GIF_Image, 468
spelling_underline_color
Fl_Text_Display, 1732
start
Fl_Anim_GIF_Image, 468
Fl_Text_Selection, 1764
start_job
Fl_Paged_Device, 1236

Fl_PostScript_File_Device, 1278, 1279
start_page
Fl_Paged_Device, 1237
State
Pen and tablet event handling, 268
static_value
Fl_Input_, 1017
step
Fl_Counter, 665
Fl_Spinner, 1522
Fl_Valuator, 1905
stop
Fl_Anim_GIF_Image, 469
STRICT_RFC3629
Unicode and UTF-8 functions, 350
STRIKEOUT
Fl_Terminal, 1650
String handling functions, 372
fl_strdup, 372
string_width
Fl_Text_Display, 1732
style_buffer
Fl_Text_Display, 1732
submenu
Fl_Menu_Item, 1129
subscribe
Pen and tablet event handling, 276
surface
Fl_Surface_Device, 1536
suspended
Fl_Timer, 1802
swap
Fl_Browser, 516
swap_buffers
Fl_Gl_Window, 836
swap_children
Fl_Tree_Item, 1885
swap_interval
Fl_Gl_Window, 836
swapping
Fl_Browser_, 541
SYSTEM
Fl_Preferences, 1284
SYSTEM_L
Fl_Preferences, 1284

tab_align
Fl_Tabs, 1625, 1626
tab_cell_nav
Fl_Table, 1588
tab_count
Fl_Tabs, 1628
tab_distance
Fl_Text_Buffer, 1688
tab_flags
Fl_Tabs, 1628
tab_height
Fl_Tabs, 1626
tab_nav
Fl_Input_, 1017
Fl_Text_Editor, 1760, 1761
tab_pos
Fl_Tabs, 1629
tab_positions
Fl_Tabs, 1626
tab_width
Fl_Tabs, 1629
tabbing_mode_automatic
Fl_Sys_Menu_Bar, 1559
tabbing_mode_none
Fl_Sys_Menu_Bar, 1559
tabbing_mode_preferred
Fl_Sys_Menu_Bar, 1559
table_box
Fl_Table, 1589
table_resized
Fl_Table, 1589
table_scrolled
Fl_Table, 1589
TableContext
Fl_Table, 1579
TABLOID
Fl_Paged_Device, 1234
take_focus
Fl_Tabs, 1627
Fl_Widget, 1976
takesevents
Fl_Widget, 1976
test_shortcut
Events handling functions, 264
Fl_Menu_, 1089
Fl_Menu_Item, 1129
Fl_Widget, 1976
text
Fl_Browser, 517
Fl_Menu_, 1090
Fl_Terminal, 1665
Fl_Text_Buffer, 1688
text_range
Fl_Text_Buffer, 1688
text_selected
Fl_Help_View, 911
text_str
Fl_Text_Buffer, 1689
textattrib
Fl_Terminal, 1666
textbgcolor
Fl_Terminal, 1666
textbgcolor_default
Fl_Terminal, 1667
textbgcolor_xterm
Fl_Terminal, 1667
textcolor
Fl_Input_, 1018
Fl_Menu_, 1090
Fl_Terminal, 1667
Fl_Text_Display, 1732, 1733

Fl_Tooltip, 1816
Fl_Value_Input, 1917
Fl_Value_Output, 1927
textfgcolor
Fl_Terminal, 1668
textfgcolor_default
Fl_Terminal, 1668
textfgcolor_xterm
Fl_Terminal, 1668
textfont
Fl_Browser_, 541
Fl_Input_, 1018
Fl_Menu_, 1090
Fl_Terminal, 1669
Fl_Text_Display, 1733
Fl_Value_Input, 1917
Fl_Value_Output, 1927
textsize
Fl_Browser, 517
Fl_Help_Dialog, 896
Fl_Input_, 1019
Fl_Menu_, 1090
Fl_Terminal, 1669
Fl_Text_Display, 1733
Fl_Value_Input, 1917
Fl_Value_Output, 1927
The Wayland backend for its developer, 188
thread_message
Multithreading support functions, 344
Tick_Position
Fl_Slider, 1507
ticks
Fl_Slider, 1510
TICKS_ABOVE
Fl_Slider, 1507
TICKS_BELOW
Fl_Slider, 1507
ticks_between
Fl, 436
TICKS_LEFT
Fl_Slider, 1507
TICKS_NONE
Fl_Slider, 1507
TICKS_RIGHT
Fl_Slider, 1507
ticks_since
Fl, 436
TILT_X
Pen and tablet event handling, 269
TILT_Y
Pen and tablet event handling, 269
time_to_wait
Fl_Timeout, 1793
timeout_list
Fl, 437
TIP_DOWN
Pen and tablet event handling, 268
TIP_HOVERS
Pen and tablet event handling, 268
title
Fl_Help_View, 912
Fl_Native_File_Chooser, 1185
Todo List, 217
tooltip
Fl_Widget, 1977
TOOLTIP_WINDOW
Fl_Widget, 1949
top_row
Fl_Table, 1589
top_window
Fl_Widget, 1977
top_window_offset
Fl_Widget, 1978
topline
Fl_Browser, 517, 518
Fl_Help_View, 912
TOUCH
Pen and tablet event handling, 268
Trait
Pen and tablet event handling, 268
transcoding_warning_action
Fl_Text_Buffer, 1690
translate
Fl_Copy_Surface, 654
Fl_EPS_File_Surface, 695
Fl_Image_Surface, 983
Fl_PDF_File_Surface, 1243
Fl_PostScript_File_Device, 1279
Fl_Printer, 1307
Fl_SVG_File_Surface, 1540
Fl_Widget_Surface, 1986
tree
Fl_Tree_Item, 1886
TWIST
Pen and tablet event handling, 269
Type
Fl_File_Chooser, 716
Fl_Native_File_Chooser, 1182
type
Fl_File_Icon, 723
Fl_Label, 1051
Fl_Spinner, 1522
Fl_Table_Row, 1606
Fl_Widget, 1978
typea
Fl_Multi_Label, 1158
typeb
Fl_Multi_Label, 1159

u8c_disp_row
Fl_Terminal, 1669
u8c_hist_row
Fl_Terminal, 1670
u8c_hist_use_row
Fl_Terminal, 1670
u8c_ring_row
Fl_Terminal, 1670

un_maximize
Fl_Window, 2019
uncache
Fl_Anim_GIF_Image, 469
Fl_Bitmap, 475
Fl_Image, 976
Fl_Pixmap, 1249
Fl_RGB_Image, 1366
Fl_Shared_Image, 1467
UNDERLINE
Fl_Terminal, 1650
undo
Fl_Input_, 1019
Fl_Text_Buffer, 1689
Unicode and UTF-8 functions, 347
ERRORS_TO_CP1252, 350
ERRORS_TO_ISO8859_1, 350
fl_access, 350
fl_chdir, 350
fl_chmod, 351
fl_close_fd, 351
fl_execvp, 351
fl_fopen, 352
fl_getcwd, 352
fl_getenv, 353
fl_make_path, 353
fl_make_path_for_file, 354
fl_mkdir, 354
fl_nonspacing, 354
fl_open, 355
fl_open_ext, 355
fl_putenv, 355
fl_rename, 356
fl_rmdir, 356
fl_stat, 357
fl_system, 357
fl_tolower, 357
fl_toupper, 358
fl_ucs_to_Utf16, 358
fl_unlink, 359
fl_utf2mbcs, 359
fl_utf8_clean, 359
fl_utf8_next_composed_char, 360
fl_utf8_previous_composed_char, 360
fl_utf8back, 361
fl_utf8bytes, 361
fl_utf8decode, 361
fl_utf8encode, 362
fl_utf8from_mb, 362
fl_utf8froma, 363
fl_utf8fromwc, 363
fl_utf8fwd, 364
fl_utf8len, 364
fl_utf8len1, 365
fl_utf8locale, 365
fl_utf8strlen, 366
fl_utf8test, 366
fl_utf8to_mb, 367
fl_utf8toa, 367
fl_utf8toUtf16, 367
fl_utf8towc, 368
fl_utf_nb_char, 369
fl_utf_strcasecmp, 369
fl_utf_strncasecmp, 369
fl_utf_tolower, 370
fl_utf_toupper, 371
fl_wcwidth, 371
fl_wcwidth_, 371
STRICT_RFC3629, 350
Unicode and UTF-8 Support, 105
UNKNOWN_ROOT_TYPE
Fl_Preferences, 1284
unlock
Multithreading support functions, 345
unsubscribe
Pen and tablet event handling, 276
untranslate
Fl_Copy_Surface, 654
Fl_EPS_File_Surface, 695
Fl_Image_Surface, 983
Fl_PDF_File_Surface, 1244
Fl_PostScript_File_Device, 1279
Fl_Printer, 1307
Fl_SVG_File_Surface, 1540
Fl_Widget_Surface, 1986
up_down_position
Fl_Input_, 1019
update
Fl_Menu_Bar, 1103
Fl_Shared_Image, 1467
Fl_Sys_Menu_Bar, 1564
Fl_Text_Selection, 1765
update_child
Fl_Group, 894
update_h_scrollbar
Fl_Text_Display, 1733
update_line_starts
Fl_Text_Display, 1734
update_menubutton
Fl_Input_Choice, 1034
update_prev_next
Fl_Tree_Item, 1886
update_v_scrollbar
Fl_Text_Display, 1734
USE_FILTER_EXT
Fl_Native_File_Chooser, 1182
use_high_res_GL
Fl, 437
USER
Fl_Preferences, 1284
user_data
Fl_Widget, 1978
USER_L
Fl_Preferences, 1284
userdeicon
Fl_Tree_Item, 1886

Fl_Tree_Prefs, 1895
USERFLAG1
Fl_Widget, 1949
USERFLAG2
Fl_Widget, 1949
USERFLAG3
Fl_Widget, 1949
usericon
Fl_Tree, 1865
Fl_Tree_Item, 1887
Using OpenGL, 90
utf8_char_at_disp
Fl_Terminal, 1671
utf8_char_at_glob
Fl_Terminal, 1671
utf8_internal.h, 2406

valid
Fl_Anim_GIF_Image, 469
Fl_GL_Window, 837
value
Fl_Browser, 518
Fl_Button, 554
Fl_Choice, 617, 618
Fl_Clock_Output, 637
Fl_Color_Chooser, 650
Fl_File_Chooser, 717, 718
Fl_File_Input, 734, 735
Fl_Help_Dialog, 896
Fl_Help_View, 912
Fl_Input_, 1020, 1021
Fl_Input_Choice, 1035
Fl_Menu_, 1090–1092
Fl_Menu_Item, 1129
Fl_Progress, 1315
Fl_Scrollbar, 1431
Fl_Shortcut_Button, 1475, 1476
Fl_Spinner, 1523
Fl_Tabs, 1627
Fl_Valuator, 1905
Fl_Wizard, 2030
value_damage
Fl_Adjuster, 454
Fl_Valuator, 1906
value_height
Fl_Value_Slider, 1938
value_to_position
Fl_Slider, 1510
value_width
Fl_Value_Slider, 1940
version
Fl, 437
Version Numbers, 247
FL_ABI_VERSION, 247
FL_API_VERSION, 248
FL_MAJOR_VERSION, 248
FL_MINOR_VERSION, 248
FL_PATCH_VERSION, 248
FL_VERSION, 248
VERTICAL
Fl_Browser_, 530
Fl_Flex, 763
VERTICAL_ALWAYS
Fl_Browser_, 530
vertical_label_margin
Fl_Widget, 1978, 1979
visible
Fl_Browser, 518
Fl_Widget, 1979
visible_cells
Fl_Table, 1589
VISIBLE_FOCUS
Fl_Widget, 1949
visible_focus
Fl, 437
Fl_Widget, 1979
visible_r
Fl_Tree_Item, 1887
Fl_Widget, 1980
visual
Fl, 437
vline_length
Fl_Text_Display, 1734
vposition
Fl_Browser_, 542
Fl_Tree, 1865
vprintf
Fl_Terminal, 1671
Fl_Text_Buffer, 1689
vsnprintf.c, 2407
fl_vsnprintf, 2407

w
Fl_Image, 976, 977
Fl_Widget, 1980
w_to_col
Fl_Terminal, 1672
wait
Fl, 438
wait_for_expose
Fl_Window, 2019
walk_selection
Fl_Terminal, 1672
warning
Common Dialog Classes and Functions, 393
watch_widget_pointer
Safe widget deletion support functions, 374
wayland.H, 2348, 2349
fl_wl_compositor, 2349
fl_wl_gc, 2349
when
Fl_Table, 1590
Fl_Widget, 1980
which
Fl_Tabs, 1628
widget
Fl_Grid, 874, 875
Fl_Widget_Tracker, 1987

win32.H, 2350
win32_target.h, 2351
window
    Fl_Widget, 1981
window_menu_style
    Fl_Sys_Menu_Bar, 1564
window_menu_style_enum
    Fl_Sys_Menu_Bar, 1559
Windows handling functions, 278
    atclose, 280
    default_atclose, 278
    first_window, 278
    grab, 279
    modal, 279
    next_window, 279
    set_atclose, 279
word_end
    Fl_Input_, 1021
    Fl_Text_Buffer, 1689
    Fl_Text_Display, 1734
word_start
    Fl_Input_, 1022
    Fl_Text_Buffer, 1690
    Fl_Text_Display, 1735
wrap
    Fl_Input_, 1022
    Fl_Spinner, 1523
WRAP_AT_BOUNDS
    Fl_Text_Display, 1708
WRAP_AT_COLUMN
    Fl_Text_Display, 1708
WRAP_AT_PIXEL
    Fl_Text_Display, 1708
wrap_mode
    Fl_Text_Display, 1735
WRAP_NONE
    Fl_Text_Display, 1708
wrap_uses_character
    Fl_Text_Display, 1735
wrap_width
    Fl_Tooltip, 1816
wrapped_column
    Fl_Text_Display, 1736
wrapped_line_counter
    Fl_Text_Display, 1736
wrapped_row
    Fl_Text_Display, 1737

x
    Fl_Widget, 1981
x.H, 2352
x11.H, 2352, 2353
    fl_x11_find, 2352
    fl_x11_gc, 2353
    fl_x11_use_display, 2353
    fl_x11_xid, 2353
x_to_col
    Fl_Text_Display, 1737
xclass
    Fl_Window, 2020
xposition
    Fl_Scroll, 1420
Xutf8.h, 2408
xy_to_position
    Fl_Text_Display, 1738
xy_to_rowcol
    Fl_Text_Display, 1738

y
    Fl_Widget, 1982
yposition
    Fl_Scroll, 1420