

FLTK Programming Manual



Revision 11 by F. Costantini, D. Gibson, M. Melcher,
A. Schlosser, B. Spitzak, and M. Sweet.

Copyright 1998-2020 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 1.8.13

September 8, 2020

Contents

1 FLTK Programming Manual	1
2 Preface	3
2.1 Organization	3
2.2 Conventions	4
2.3 Abbreviations	4
2.4 Copyrights and Trademarks	4
3 Introduction to FLTK	5
3.1 History of FLTK	5
3.2 Features	6
3.3 Licensing	6
3.4 What Does "FLTK" Mean?	7
3.5 Building and Installing FLTK Under UNIX and Apple macOS	7
3.6 Building FLTK Under Microsoft Windows	9
3.6.1 GNU toolsets (Cygwin or MinGW) hosted on Windows	9
3.6.2 Using the Visual C++ DLL Library	10
3.7 Internet Resources	10
3.8 Reporting Bugs	10

4 FLTK Basics	11
4.1 Writing Your First FLTK Program	11
4.1.1 Creating the Widgets	12
4.1.2 Creating Widget hierarchies	12
4.1.3 Get/Set Methods	13
4.1.4 Redrawing After Changing Attributes	13
4.1.5 Labels	13
4.1.6 Showing the Window	13
4.1.7 The Main Event Loop	13
4.2 Compiling Programs with Standard Compilers	14
4.3 Compiling Programs with Makefiles	15
4.4 Compiling Programs with Microsoft Visual C++	15
4.5 Naming	15
4.6 Header Files	16
5 Common Widgets and Attributes	17
5.1 Buttons	17
5.2 Text	18
5.3 Valuators	19
5.4 Groups	20
5.5 Setting the Size and Position of Widgets	20
5.6 Colors	21
5.7 Box Types	22
5.7.1 Making Your Own Boxtypes	22
5.8 Labels and Label Types	24
5.9 Callbacks	28
5.10 Shortcuts	28
6 How does resizing work?	29
6.1 Resizing can be disabled	29
6.2 Resizing can be simple	29
6.3 Resizing can be complex	30
6.4 Practical examples	31

7 Designing a Simple Text Editor	35
7.1 Determining the Goals of the Text Editor	35
7.2 Designing the Main Window	35
7.3 Variables	36
7.4 Menubars and Menus	36
7.5 Editing the Text	37
7.6 The Replace Dialog	37
7.7 Callbacks	37
7.7.1 <code>changed_cb()</code>	38
7.7.2 <code>copy_cb()</code>	38
7.7.3 <code>cut_cb()</code>	38
7.7.4 <code>delete_cb()</code>	38
7.7.5 <code>find_cb()</code>	38
7.7.6 <code>find2_cb()</code>	39
7.7.7 <code>new_cb()</code>	39
7.7.8 <code>open_cb()</code>	39
7.7.9 <code>paste_cb()</code>	39
7.7.10 <code>quit_cb()</code>	40
7.7.11 <code>replace_cb()</code>	40
7.7.12 <code>replace2_cb()</code>	40
7.7.13 <code>replall_cb()</code>	40
7.7.14 <code>replcan_cb()</code>	41
7.7.15 <code>save_cb()</code>	41
7.7.16 <code>saveas_cb()</code>	41
7.8 Other Functions	42
7.8.1 <code>check_save()</code>	42
7.8.2 <code>load_file()</code>	42
7.8.3 <code>save_file()</code>	42
7.8.4 <code>set_title()</code>	43
7.9 The <code>main()</code> Function	43
7.10 Compiling the Editor	43
7.11 The Final Product	44
7.12 Advanced Features	44
7.12.1 Syntax Highlighting	44

8 Drawing Things in FLTK	49
8.1 When Can You Draw Things in FLTK?	49
8.1.1 What Drawing Unit do FLTK drawing functions use?	50
8.2 Drawing Functions	51
8.2.1 Boxes	51
8.2.2 Clipping	52
8.3 Colors	53
8.3.1 Line Dashes and Thickness	55
8.3.2 Drawing Fast Shapes	56
8.3.3 Drawing Complex Shapes	58
8.3.4 Drawing Text	61
8.3.5 Fonts	63
8.3.6 Character Encoding	63
8.3.7 Drawing Overlays	64
8.4 Drawing Images	64
8.4.1 Direct Image Drawing	64
8.4.2 Direct Image Reading	66
8.4.3 Image Classes	66
8.4.4 Offscreen Drawing	68
9 Handling Events	69
9.1 The FLTK Event Model	69
9.2 Mouse Events	69
9.2.1 FL_PUSH	69
9.2.2 FL_DRAG	69
9.2.3 FL_RELEASE	70
9.2.4 FL_MOVE	70
9.2.5 FL_MOUSEWHEEL	70
9.3 Focus Events	70
9.3.1 FL_ENTER	70
9.3.2 FL_LEAVE	70

9.3.3	FL_FOCUS	70
9.3.4	FL_UNFOCUS	70
9.4	Keyboard Events	71
9.4.1	FL_KEYBOARD, FL_KEYDOWN, FL_KEYUP	71
9.4.2	FL_SHORTCUT	71
9.5	Widget Events	71
9.5.1	FL_DEACTIVATE	71
9.5.2	FL_ACTIVATE	71
9.5.3	FL_HIDE	72
9.5.4	FL_SHOW	72
9.6	Clipboard Events	72
9.6.1	FL_PASTE	72
9.6.2	FL_SELECTIONCLEAR	72
9.7	Drag and Drop Events	72
9.7.1	FL_DND_ENTER	73
9.7.2	FL_DND_DRAG	73
9.7.3	FL_DND_LEAVE	73
9.7.4	FL_DND_RELEASE	73
9.8	Other events	73
9.8.1	FL_SCREEN_CONFIGURATION_CHANGED	73
9.8.2	FL_FULLSCREEN	73
9.9	Fl::event_*() methods	74
9.10	Event Propagation	74
9.11	FLTK Compose-Character Sequences	75

10 Adding and Extending Widgets	77
10.1 Subclassing	77
10.2 Making a Subclass of Fl_Widget	77
10.3 The Constructor	77
10.4 Protected Methods of Fl_Widget	78
10.5 Handling Events	81
10.6 Drawing the Widget	82
10.7 Resizing the Widget	82
10.8 Making a Composite Widget	82
10.9 Cut and Paste Support	84
10.10 Drag And Drop Support	84
10.11 Making a subclass of Fl_Window	84
 11 Using OpenGL	 85
11.1 Using OpenGL in FLTK	85
11.2 Making a Subclass of Fl_Gl_Window	85
11.2.1 Defining the Subclass	86
11.2.2 The draw() Method	86
11.2.3 The handle() Method	86
11.3 OpenGL and support of HighDPI displays	87
11.4 Using OpenGL in Normal FLTK Windows	88
11.5 OpenGL Drawing Functions	89
11.6 Speeding up OpenGL	90
11.7 Using OpenGL Optimizer with FLTK	90
11.8 Using OpenGL 3.0 (or higher versions)	92

12 Programming with FLUID	93
12.1 What is FLUID?	93
12.2 Running FLUID Under UNIX	94
12.3 Running FLUID Under Microsoft Windows	95
12.4 Compiling .fl Files	95
12.5 A Short Tutorial	96
12.5.1 The CubeView Class	96
12.5.2 The CubeViewUI Class	99
12.5.3 Adding Constructor Initialization Code	102
12.5.4 Generating the Code	102
12.6 FLUID Reference	102
12.6.1 The Widget Browser	102
12.6.2 Menu Items	103
12.6.3 The Widget Panel	111
12.7 GUI Attributes	111
12.7.1 Style Attributes	114
12.7.2 C++ Attributes	116
12.8 Selecting and Moving Widgets	118
12.9 Image Labels	119
12.10 FLUID Templates	121
12.11 Internationalization with FLUID	121
12.11.1 I18N Methods	122
12.11.2 Using GNU gettext for I18N	122
12.11.3 Using POSIX catgets for I18N	123
12.12 Known Limitations	123
13 Advanced FLTK	125
13.1 Multithreading	125
13.2 FLTK multithread locking - Fl::lock() and Fl::unlock()	125
13.3 Simple multithreaded examples using Fl::lock	126
13.4 FLTK multithreaded "lockless programming"	128
13.5 FLTK multithreaded Constraints	130

14 Unicode and UTF-8 Support	131
14.1 About Unicode, ISO 10646 and UTF-8	131
14.2 Unicode in FLTK	133
14.3 Illegal Unicode and UTF-8 Sequences	134
14.4 FLTK Unicode and UTF-8 Functions	134
14.5 FLTK Unicode Versions of System Calls	137
15 FLTK Enumerations	139
15.1 Version Numbers	139
15.2 Events	140
15.3 Callback "When" Conditions	141
15.4 Fl::event_button() Values	141
15.5 Fl::event_key() Values	141
15.6 Fl::event_state() Values	142
15.7 Alignment Values	143
15.8 Fonts	143
15.9 Colors	144
15.9.1 Color Constants	144
15.10 Cursors	145
15.11 FD "When" Conditions	146
15.12 Damage Masks	146
16 GLUT Compatibility	147
16.1 Using the GLUT Compatibility Header File	147
16.2 Known Problems	147
16.3 Mixing GLUT and FLTK Code	148
16.4 class Fl_Glut_Window	149
16.4.1 Class Hierarchy	149
16.4.2 Include Files	149
16.4.3 Description	149
16.4.4 Members	149
16.4.5 Methods	150

17 Forms Compatibility	151
17.1 Importing Forms Layout Files	151
17.2 Using the Compatibility Header File	151
17.3 Problems You Will Encounter	152
17.4 Additional Notes	153
18 Operating System Issues	157
18.1 Accessing the OS Interfaces	157
18.2 The UNIX (X11) Interface	158
18.2.1 Handling Other X Events	158
18.2.2 Drawing using Xlib	159
18.2.3 Changing the Display, Screen, or X Visual	160
18.2.4 Using a Subclass of Fl_Window for Special X Stuff	161
18.2.5 Setting the Icon of a Window	162
18.2.6 X Resources	163
18.2.7 Display Scaling Factor	164
18.3 The Windows Interface	164
18.3.1 Using filenames with non-ASCII characters	164
18.3.2 Responding to WM_QUIT	164
18.3.3 Handling Other Windows API Messages	164
18.3.4 Drawing Things Using the Windows GDI	165
18.3.5 Display Scaling Factor	165
18.3.6 Setting the Icon of a Window	165
18.3.7 How to Not Get a MSDOS Console Window	166
18.3.8 Known Windows Bugs and Problems	166
18.4 The Apple OS X Interface	166
18.4.1 Setting the icon of an application	168
18.4.2 Drawing Things Using Quartz	168
18.4.3 Internationalization	169
18.4.4 OpenGL and 'retina' displays	169
18.4.5 Fl_Double_Window	170
18.4.6 Mac File System Specifics	170

19 Migrating Code from FLTK 1.3 to 1.4	171
19.1 Migrating from FLTK 1.0 or 1.1 to 1.4	171
19.2 Minor Changes in Header Files	171
19.3 Fl_Preferences	171
20 Developer Information	173
20.1 Non-ASCII Characters	175
20.2 Document Structure	175
20.3 Creating Links	176
20.4 Paragraph Layout	177
20.5 Navigation Elements	178
21 Software License	179
22 Example Source Code	187
22.1 Example Applications: Overview	187
22.1.1 adjuster	187
22.1.2 animated	188
22.1.3 arc	188
22.1.4 ask	188
22.1.5 bitmap	188
22.1.6 blocks	188
22.1.7 boxtyle	188
22.1.8 browser	188
22.1.9 button	188
22.1.10 buttons	189
22.1.11 cairo_test	189
22.1.12 checkers	189
22.1.13 clock	189
22.1.14 colrowser	189
22.1.15 color_chooser	189
22.1.16 cube	189

22.1.17 CubeView	189
22.1.18 cursor	190
22.1.19 curve	190
22.1.20 demo	190
22.1.21 device	190
22.1.22 doublebuffer	190
22.1.23 editor	190
22.1.24 fast_slow	190
22.1.25 file_chooser	191
22.1.26 fonts	191
22.1.27 forms	191
22.1.28 fractals	191
22.1.29 fullscreen	191
22.1.30 gl_overlay	191
22.1.31 glpuzzle	191
22.1.32 hello	191
22.1.33 help_dialog	192
22.1.34 icon	192
22.1.35 iconize	192
22.1.36 image	192
22.1.37 inactive	192
22.1.38 input	192
22.1.39 input_choice	192
22.1.40 keyboard	193
22.1.41 label	193
22.1.42 line_style	193
22.1.43 list_visuals	193
22.1.44 mandelbrot	193
22.1.45 menubar	193
22.1.46 message	193

22.1.47 minimum	193
22.1.48 native-filechooser	194
22.1.49 navigation	194
22.1.50 offscreen	194
22.1.51 output	194
22.1.52 overlay	194
22.1.53 pack	194
22.1.54 pixmap	194
22.1.55 pixmap_browser	194
22.1.56 preferences	195
22.1.57 radio	195
22.1.58 resizebox	195
22.1.59 rotated_text	195
22.1.60 resize	195
22.1.61 scroll	195
22.1.62 shape	195
22.1.63 subwindow	195
22.1.64 sudoku	196
22.1.65 symbols	196
22.1.66 table	196
22.1.67 tabs	196
22.1.68 threads	196
22.1.69 tile	196
22.1.70 tiled_image	196
22.1.71 tree	197
22.1.72 twowin	197
22.1.73 unittests	197
22.1.74 utf8	197
22.1.75 valuators	197
22.1.76 windowfocus	197
22.1.77 fluid	197
22.2 Example Applications: Images	197
22.2.1 cairo_test	198
22.2.2 icon	198
22.2.3 unittests	198

23 FAQ (Frequently Asked Questions)	201
23.1 Where do I start learning FLTK?	201
23.2 How do I make a box with text?	201
23.3 Can I use FLTK to make closed-source commercial applications?	202
23.4 Hitting the 'Escape' key closes windows - how do I prevent this?	202
24 Todo List	205
25 Deprecated List	209
26 Module Index	211
26.1 Modules	211
27 Hierarchical Index	213
27.1 Class Hierarchy	213
28 Class Index	217
28.1 Class List	217
29 File Index	223
29.1 File List	223
30 Module Documentation	229
30.1 Callback function typedefs	229
30.1.1 Detailed Description	230
30.1.2 Typedef Documentation	230
30.1.2.1 Fl_Event_Dispatch	230
30.2 runtime graphics driver configuration	231
30.2.1 Detailed Description	231
30.3 runtime printer driver configuration	232
30.3.1 Detailed Description	232
30.4 runtime window and event manager configuration	233
30.4.1 Detailed Description	233
30.5 runtime system configuration	234

30.5.1 Detailed Description	234
30.6 Windows handling functions	235
30.6.1 Detailed Description	235
30.6.2 Function Documentation	235
30.6.2.1 default_atclose()	235
30.6.2.2 first_window() [1/2]	236
30.6.2.3 first_window() [2/2]	236
30.6.2.4 grab() [1/2]	236
30.6.2.5 grab() [2/2]	236
30.6.2.6 modal()	237
30.6.2.7 next_window()	237
30.6.2.8 set_atclose()	237
30.6.3 Variable Documentation	237
30.6.3.1 atclose	237
30.7 Events handling functions	238
30.7.1 Detailed Description	240
30.7.2 Function Documentation	240
30.7.2.1 add_handler()	240
30.7.2.2 add_system_handler()	241
30.7.2.3 belowmouse() [1/2]	241
30.7.2.4 belowmouse() [2/2]	241
30.7.2.5 compose()	242
30.7.2.6 compose_reset()	242
30.7.2.7 disable_im()	242
30.7.2.8 enable_im()	243
30.7.2.9 event()	243
30.7.2.10 event_alt()	243
30.7.2.11 event_button()	243
30.7.2.12 event_button1()	244
30.7.2.13 event_button2()	244

30.7.2.14 event_button3()	244
30.7.2.15 event_buttons()	244
30.7.2.16 event_clicks() [1/2]	244
30.7.2.17 event_clicks() [2/2]	245
30.7.2.18 event_clipboard()	245
30.7.2.19 event_clipboard_type()	245
30.7.2.20 event_command()	245
30.7.2.21 event_ctrl()	246
30.7.2.22 event_dispatch()	246
30.7.2.23 event_dx()	246
30.7.2.24 event_dy()	247
30.7.2.25 event_inside() [1/2]	247
30.7.2.26 event_inside() [2/2]	247
30.7.2.27 event_is_click() [1/2]	248
30.7.2.28 event_is_click() [2/2]	248
30.7.2.29 event_key() [1/2]	248
30.7.2.30 event_key() [2/2]	249
30.7.2.31 event_length()	249
30.7.2.32 event_original_key()	249
30.7.2.33 event_shift()	250
30.7.2.34 event_state() [1/2]	250
30.7.2.35 event_state() [2/2]	250
30.7.2.36 event_text()	251
30.7.2.37 event_x_root()	251
30.7.2.38 event_y_root()	251
30.7.2.39 focus() [1/2]	251
30.7.2.40 focus() [2/2]	252
30.7.2.41 get_key()	252
30.7.2.42 get_mouse()	252
30.7.2.43 handle()	252

30.7.2.44 handle_()	253
30.7.2.45 pushed() [1/2]	253
30.7.2.46 pushed() [2/2]	254
30.7.2.47 remove_handler()	254
30.7.2.48 remove_system_handler()	254
30.7.2.49 test_shortcut()	254
30.7.3 Variable Documentation	255
30.7.3.1 fl_eventnames	255
30.7.3.2 fl_fontnames	255
30.8 Selection & Clipboard functions	256
30.8.1 Detailed Description	256
30.8.2 Function Documentation	256
30.8.2.1 add_clipboard_notify()	257
30.8.2.2 clipboard_contains()	257
30.8.2.3 copy()	257
30.8.2.4 dnd()	258
30.8.2.5 paste() [1/2]	258
30.8.2.6 paste() [2/2]	259
30.8.2.7 selection()	259
30.8.2.8 selection_owner() [1/2]	259
30.8.2.9 selection_owner() [2/2]	259
30.9 Screen functions	260
30.9.1 Detailed Description	261
30.9.2 Function Documentation	261
30.9.2.1 h()	261
30.9.2.2 keyboard_screen_scaling()	261
30.9.2.3 screen_dpi()	261
30.9.2.4 screen_num() [1/2]	262
30.9.2.5 screen_num() [2/2]	262
30.9.2.6 screen_scale()	263

30.9.2.7 screen_scaling_supported()	263
30.9.2.8 screen_work_area() [1/3]	263
30.9.2.9 screen_work_area() [2/3]	263
30.9.2.10 screen_work_area() [3/3]	264
30.9.2.11 screen_xywh() [1/4]	264
30.9.2.12 screen_xywh() [2/4]	265
30.9.2.13 screen_xywh() [3/4]	265
30.9.2.14 screen_xywh() [4/4]	266
30.9.2.15 w()	266
30.9.2.16 x()	266
30.9.2.17 y()	266
30.10 Color & Font functions	267
30.10.1 Detailed Description	268
30.10.2 Function Documentation	268
30.10.2.1 fl_color() [1/3]	268
30.10.2.2 fl_color() [2/3]	269
30.10.2.3 fl_color() [3/3]	269
30.10.2.4 fl_color_average()	269
30.10.2.5 fl_contrast()	270
30.10.2.6 fl_font() [1/2]	270
30.10.2.7 fl_font() [2/2]	270
30.10.2.8 fl_height() [1/2]	271
30.10.2.9 fl_height() [2/2]	271
30.10.2.10 fl_latin1_to_local()	271
30.10.2.11 fl_local_to_latin1()	272
30.10.2.12 fl_local_to_mac_roman()	272
30.10.2.13 fl_mac_roman_to_local()	272
30.10.2.14 fl_show_colormap()	273
30.10.2.15 fl_size()	274
30.10.2.16 fl_text_extents() [1/2]	274

30.10.2.17fl_text_extents() [2/2]	274
30.10.2.18fl_width() [1/3]	275
30.10.2.19fl_width() [2/3]	275
30.10.2.20fl_width() [3/3]	275
30.10.2.21free_color()	275
30.10.2.22get_color() [1/2]	275
30.10.2.23get_color() [2/2]	276
30.10.2.24get_font()	276
30.10.2.25get_font_name()	276
30.10.2.26get_font_sizes()	276
30.10.2.27set_color() [1/2]	277
30.10.2.28set_color() [2/2]	277
30.10.2.29set_font() [1/2]	277
30.10.2.30set_font() [2/2]	277
30.10.2.31set_fonts()	277
30.11 Drawing functions	278
30.11.1 Detailed Description	282
30.11.2 Macro Definition Documentation	282
30.11.2.1 fl_clip	282
30.11.3 Enumeration Type Documentation	282
30.11.3.1 anonymous enum	283
30.11.4 Function Documentation	283
30.11.4.1 fl_add_symbol()	283
30.11.4.2 fl_arc() [1/2]	283
30.11.4.3 fl_arc() [2/2]	284
30.11.4.4 fl_begin_complex_polygon()	285
30.11.4.5 fl_begin_offscreen()	286
30.11.4.6 fl_begin_points()	286
30.11.4.7 fl_can_do_alpha_blending()	286
30.11.4.8 fl_capture_window_part()	286

30.11.4.9 fl_circle()	287
30.11.4.10 fl_clip_box()	287
30.11.4.11 fl_clip_region() [1/2]	288
30.11.4.12 fl_clip_region() [2/2]	289
30.11.4.13 fl_copy_offscreen()	289
30.11.4.14 fl_create_offscreen()	289
30.11.4.15 fl_cursor()	290
30.11.4.16 fl_curve()	290
30.11.4.17 fl_delete_offscreen()	291
30.11.4.18 fl_draw() [1/4]	291
30.11.4.19 fl_draw() [2/4]	291
30.11.4.20 fl_draw() [3/4]	292
30.11.4.21 fl_draw() [4/4]	292
30.11.4.22 fl_draw_box()	292
30.11.4.23 fl_draw_image() [1/2]	293
30.11.4.24 fl_draw_image() [2/2]	294
30.11.4.25 fl_draw_image_mono() [1/2]	294
30.11.4.26 fl_draw_image_mono() [2/2]	295
30.11.4.27 fl_draw_pixmap() [1/2]	295
30.11.4.28 fl_draw_pixmap() [2/2]	295
30.11.4.29 fl_draw_symbol()	296
30.11.4.30 fl_expand_text()	296
30.11.4.31 fl_frame()	297
30.11.4.32 fl_frame2()	297
30.11.4.33 fl_gap()	297
30.11.4.34 fl_line_style()	298
30.11.4.35 fl_measure()	298
30.11.4.36 fl_measure_pixmap() [1/2]	299
30.11.4.37 fl_measure_pixmap() [2/2]	299
30.11.4.38 fl_mult_matrix()	300

30.11.4.3 <code>fl_not_clipped()</code>	300
30.11.4.4 <code>fl_old_shortcut()</code>	301
30.11.4.41 <code>fl_pie()</code>	302
30.11.4.42 <code>fl_polygon()</code> [1/2]	302
30.11.4.43 <code>fl_polygon()</code> [2/2]	303
30.11.4.44 <code>fl_pop_clip()</code>	303
30.11.4.45 <code>fl_push_clip()</code>	303
30.11.4.46 <code>fl_push_matrix()</code>	303
30.11.4.47 <code>fl_read_image()</code>	304
30.11.4.48 <code>fl_rect()</code>	304
30.11.4.49 <code>fl_rectf()</code>	305
30.11.4.50 <code>fl_rescale_offscreen()</code>	305
30.11.4.51 <code>fl_reset_spot()</code>	305
30.11.4.52 <code>fl_rotate()</code>	305
30.11.4.53 <code>fl_scale()</code> [1/2]	306
30.11.4.54 <code>fl_scale()</code> [2/2]	306
30.11.4.55 <code>fl_scroll()</code>	306
30.11.4.56 <code>fl_set_spot()</code>	307
30.11.4.57 <code>fl_set_status()</code>	307
30.11.4.58 <code>fl_shortcut_label()</code> [1/2]	308
30.11.4.59 <code>fl_shortcut_label()</code> [2/2]	309
30.11.4.60 <code>fl_transform_dx()</code>	309
30.11.4.61 <code>fl_transform_dy()</code>	309
30.11.4.62 <code>fl_transform_x()</code>	311
30.11.4.63 <code>fl_transform_y()</code>	311
30.11.4.64 <code>fl_transformed_vertex()</code>	311
30.11.4.65 <code>fl_translate()</code>	312
30.11.4.66 <code>fl_vertex()</code>	312
30.12 Multithreading support functions	313
30.12.1 Detailed Description	313

30.12.2 Function Documentation	313
30.12.2.1 <code>awake()</code> [1/2]	313
30.12.2.2 <code>awake()</code> [2/2]	314
30.12.2.3 <code>lock()</code>	314
30.12.2.4 <code>thread_message()</code>	314
30.12.2.5 <code>unlock()</code>	314
30.13 Safe widget deletion support functions	315
30.13.1 Detailed Description	315
30.13.2 Function Documentation	316
30.13.2.1 <code>clear_widget_pointer()</code>	316
30.13.2.2 <code>delete_widget()</code>	316
30.13.2.3 <code>do_widget_deletion()</code>	317
30.13.2.4 <code>release_widget_pointer()</code>	317
30.13.2.5 <code>watch_widget_pointer()</code>	317
30.14 Cairo Support Functions and Classes	319
30.14.1 Detailed Description	319
30.14.2 Function Documentation	319
30.14.2.1 <code>cairo_autolink_context()</code> [1/2]	319
30.14.2.2 <code>cairo_autolink_context()</code> [2/2]	319
30.14.2.3 <code>cairo_cc()</code> [1/2]	320
30.14.2.4 <code>cairo_cc()</code> [2/2]	320
30.14.2.5 <code>cairo_make_current()</code>	320
30.15 Unicode and UTF-8 functions	321
30.15.1 Detailed Description	323
30.15.2 Macro Definition Documentation	323
30.15.2.1 <code>ERRORS_TO_CP1252</code>	323
30.15.2.2 <code>ERRORS_TO_ISO8859_1</code>	323
30.15.2.3 <code>STRICT_RFC3629</code>	323
30.15.3 Function Documentation	324
30.15.3.1 <code>fl_access()</code>	324

30.15.3.2 fl_chdir()	324
30.15.3.3 fl_chmod()	325
30.15.3.4 fl_fopen()	325
30.15.3.5 fl_getcwd()	326
30.15.3.6 fl_getenv()	326
30.15.3.7 fl_make_path()	327
30.15.3.8 fl_make_path_for_file()	327
30.15.3.9 fl_mkdir()	328
30.15.3.10 fl_nonspacing()	328
30.15.3.11 fl_open()	328
30.15.3.12 fl_open_ext()	329
30.15.3.13 fl_putenv()	329
30.15.3.14 fl_rename()	330
30.15.3.15 fl_rmdir()	331
30.15.3.16 fl_stat()	331
30.15.3.17 fl_system()	331
30.15.3.18 fl_ucs_to_Utf16()	332
30.15.3.19 fl_unlink()	332
30.15.3.20 fl_utf8back()	333
30.15.3.21 fl_utf8bytes()	333
30.15.3.22 fl_utf8decode()	333
30.15.3.23 fl_utf8encode()	334
30.15.3.24 fl_utf8from_mb()	334
30.15.3.25 fl_utf8froma()	335
30.15.3.26 fl_utf8fromwc()	335
30.15.3.27 fl_utf8fwd()	336
30.15.3.28 fl_utf8len()	336
30.15.3.29 fl_utf8len1()	336
30.15.3.30 fl_utf8locale()	337
30.15.3.31 fl_utf8test()	337

30.15.3.32fl_utf8to_mb()	337
30.15.3.33fl_utf8toa()	338
30.15.3.34fl_utf8toUtf16()	338
30.15.3.35fl_utf8towc()	339
30.15.3.36fl_utf_strcasecmp()	339
30.15.3.37fl_utf_strncasecmp()	339
30.15.3.38fl_utf_tolower()	340
30.15.3.39fl_utf_toupper()	340
30.15.3.40fl_wcwidth()	340
30.15.3.41fl_wcwidth_(1)	341
30.16 Mac OS X-specific symbols	342
30.16.1 Detailed Description	342
30.16.2 Function Documentation	342
30.16.2.1 fl_mac_set_about()	342
30.16.2.2 fl_open_callback()	343
30.16.3 Variable Documentation	343
30.16.3.1 fl_mac_os_version	343
30.17 Common Dialogs classes and functions	344
30.17.1 Detailed Description	345
30.17.2 Function Documentation	345
30.17.2.1 fl_alert()	345
30.17.2.2 fl_ask()	346
30.17.2.3 fl_beep()	346
30.17.2.4 fl_choice()	347
30.17.2.5 fl_color_chooser() [1/2]	348
30.17.2.6 fl_color_chooser() [2/2]	349
30.17.2.7 fl_dir_chooser()	350
30.17.2.8 fl_file_chooser()	351
30.17.2.9 fl_file_chooser_callback()	352
30.17.2.10 fl_file_chooser_ok_label()	352

30.17.2.1 <code>fl_input()</code>	352
30.17.2.12 <code>fl_message()</code>	353
30.17.2.13 <code>fl_message_hotspot() [1/2]</code>	353
30.17.2.14 <code>fl_message_hotspot() [2/2]</code>	354
30.17.2.15 <code>fl_message_icon()</code>	354
30.17.2.16 <code>fl_message_position() [1/3]</code>	354
30.17.2.17 <code>fl_message_position() [2/3]</code>	355
30.17.2.18 <code>fl_message_position() [3/3]</code>	355
30.17.2.19 <code>fl_message_title()</code>	356
30.17.2.20 <code>fl_message_title_default()</code>	357
30.17.2.21 <code>fl_password()</code>	357
30.17.3 Variable Documentation	358
30.17.3.1 <code>error</code>	358
30.17.3.2 <code>fatal</code>	358
30.17.3.3 <code>warning</code>	358
30.18 File names and URI utility functions	359
30.18.1 Detailed Description	359
30.18.2 Typedef Documentation	360
30.18.2.1 <code>FI_File_Sort_F</code>	360
30.18.3 Function Documentation	360
30.18.3.1 <code>fl_decode_uri()</code>	360
30.18.3.2 <code>fl_filename_absolute()</code>	360
30.18.3.3 <code>fl_filename_expand()</code>	361
30.18.3.4 <code>fl_filename_ext()</code>	361
30.18.3.5 <code>fl_filename_free_list()</code>	362
30.18.3.6 <code>fl_filename_isdir()</code>	362
30.18.3.7 <code>fl_filename_list()</code>	363
30.18.3.8 <code>fl_filename_match()</code>	363
30.18.3.9 <code>fl_filename_name()</code>	364
30.18.3.10 <code>fl_filename_relative()</code>	365
30.18.3.11 <code>fl_filename_setext()</code>	365
30.18.3.12 <code>fl_open_uri()</code>	366
30.19 <code>FI_string</code>	367
30.19.1 Detailed Description	367
30.19.2 Function Documentation	367
30.19.2.1 <code>fl_strdup()</code>	367

31 Class Documentation	369
31.1 Fl_Preferences::Entry Struct Reference	369
31.2 Fl Class Reference	369
31.2.1 Detailed Description	379
31.2.2 Member Enumeration Documentation	379
31.2.2.1 Fl_Option	379
31.2.3 Member Function Documentation	380
31.2.3.1 abi_check()	380
31.2.3.2 abi_version()	380
31.2.3.3 add_awake_handler_()	380
31.2.3.4 add_check()	381
31.2.3.5 add_fd() [1/2]	381
31.2.3.6 add_fd() [2/2]	382
31.2.3.7 add_idle()	382
31.2.3.8 add_timeout()	382
31.2.3.9 api_version()	383
31.2.3.10 arg()	383
31.2.3.11 args() [1/2]	384
31.2.3.12 args() [2/2]	384
31.2.3.13 background()	385
31.2.3.14 background2()	385
31.2.3.15 box_border_radius_max() [1/2]	385
31.2.3.16 box_border_radius_max() [2/2]	385
31.2.3.17 box_color()	386
31.2.3.18 box_dh()	386
31.2.3.19 box_dw()	386
31.2.3.20 box_dx()	386
31.2.3.21 box_dy()	387
31.2.3.22 box_shadow_width() [1/2]	387
31.2.3.23 box_shadow_width() [2/2]	387

31.2.3.24 check()	388
31.2.3.25 damage()	388
31.2.3.26 display()	388
31.2.3.27 dnd_text_ops() [1/2]	388
31.2.3.28 dnd_text_ops() [2/2]	388
31.2.3.29 draw_box_active()	389
31.2.3.30 draw_GL_text_with_textures() [1/2]	389
31.2.3.31 draw_GL_text_with_textures() [2/2]	389
31.2.3.32 flush()	390
31.2.3.33 foreground()	390
31.2.3.34 get_awake_handler_()	390
31.2.3.35 get_boxtype()	390
31.2.3.36 get_system_colors()	390
31.2.3.37 gl_visual()	391
31.2.3.38 insertion_point_location()	391
31.2.3.39 is_scheme()	391
31.2.3.40 menu_linespacing() [1/2]	392
31.2.3.41 menu_linespacing() [2/2]	392
31.2.3.42 option() [1/2]	392
31.2.3.43 option() [2/2]	393
31.2.3.44 own_colormap()	394
31.2.3.45 program_should_quit() [1/2]	394
31.2.3.46 program_should_quit() [2/2]	394
31.2.3.47 readqueue()	395
31.2.3.48 ready()	395
31.2.3.49 release()	395
31.2.3.50 reload_scheme()	396
31.2.3.51 remove_check()	396
31.2.3.52 remove_fd() [1/2]	396
31.2.3.53 remove_fd() [2/2]	396

31.2.3.54 remove_timeout()	396
31.2.3.55 repeat_timeout()	397
31.2.3.56 reset_marked_text()	397
31.2.3.57 run()	397
31.2.3.58 scheme()	398
31.2.3.59 scrollbar_size() [1/2]	398
31.2.3.60 scrollbar_size() [2/2]	398
31.2.3.61 set_box_color()	399
31.2.3.62 set_boxtype() [1/2]	399
31.2.3.63 set_boxtype() [2/2]	399
31.2.3.64 set_idle()	400
31.2.3.65 set_labeltype() [1/2]	400
31.2.3.66 set_labeltype() [2/2]	400
31.2.3.67 use_high_res_GL() [1/2]	400
31.2.3.68 use_high_res_GL() [2/2]	401
31.2.3.69 version()	401
31.2.3.70 visible_focus() [1/2]	401
31.2.3.71 visible_focus() [2/2]	401
31.2.3.72 visual()	402
31.2.3.73 wait() [1/2]	402
31.2.3.74 wait() [2/2]	403
31.2.4 Member Data Documentation	403
31.2.4.1 help	403
31.2.4.2 idle	403
31.3 Fl_Adjuster Class Reference	404
31.3.1 Detailed Description	404
31.3.2 Constructor & Destructor Documentation	405
31.3.2.1 Fl_Adjuster()	405
31.3.3 Member Function Documentation	405
31.3.3.1 draw()	405

31.3.3.2 handle()	405
31.3.3.3 soft() [1/2]	406
31.3.3.4 soft() [2/2]	406
31.4 Fl_Bitmap Class Reference	406
31.4.1 Detailed Description	407
31.4.2 Constructor & Destructor Documentation	407
31.4.2.1 Fl_Bitmap() [1/2]	408
31.4.2.2 Fl_Bitmap() [2/2]	408
31.4.3 Member Function Documentation	408
31.4.3.1 copy()	408
31.4.3.2 draw()	408
31.4.3.3 label() [1/2]	409
31.4.3.4 label() [2/2]	409
31.4.3.5 uncache()	409
31.5 Fl_BMP_Image Class Reference	410
31.5.1 Detailed Description	410
31.5.2 Constructor & Destructor Documentation	410
31.5.2.1 Fl_BMP_Image() [1/2]	410
31.5.2.2 Fl_BMP_Image() [2/2]	411
31.6 Fl_Box Class Reference	411
31.6.1 Detailed Description	412
31.6.2 Constructor & Destructor Documentation	412
31.6.2.1 Fl_Box()	412
31.6.3 Member Function Documentation	413
31.6.3.1 draw()	413
31.6.3.2 handle()	413
31.7 Fl_Browser Class Reference	414
31.7.1 Detailed Description	417
31.7.2 Constructor & Destructor Documentation	418
31.7.2.1 Fl_Browser()	418

31.7.3 Member Function Documentation	418
31.7.3.1 <code>_remove()</code>	418
31.7.3.2 <code>add()</code>	419
31.7.3.3 <code>bottomline()</code>	419
31.7.3.4 <code>clear()</code>	420
31.7.3.5 <code>column_char() [1/2]</code>	420
31.7.3.6 <code>column_char() [2/2]</code>	420
31.7.3.7 <code>column_widths() [1/2]</code>	421
31.7.3.8 <code>column_widths() [2/2]</code>	421
31.7.3.9 <code>data() [1/2]</code>	421
31.7.3.10 <code>data() [2/2]</code>	422
31.7.3.11 <code>display()</code>	422
31.7.3.12 <code>displayed()</code>	422
31.7.3.13 <code>find_line()</code>	423
31.7.3.14 <code>format_char() [1/2]</code>	423
31.7.3.15 <code>format_char() [2/2]</code>	424
31.7.3.16 <code>full_height()</code>	425
31.7.3.17 <code>hide() [1/2]</code>	425
31.7.3.18 <code>hide() [2/2]</code>	425
31.7.3.19 <code>icon() [1/2]</code>	426
31.7.3.20 <code>icon() [2/2]</code>	426
31.7.3.21 <code>incr_height()</code>	426
31.7.3.22 <code>insert() [1/2]</code>	427
31.7.3.23 <code>insert() [2/2]</code>	427
31.7.3.24 <code>item_at()</code>	427
31.7.3.25 <code>item_draw()</code>	428
31.7.3.26 <code>item_first()</code>	428
31.7.3.27 <code>item_height()</code>	429
31.7.3.28 <code>item_last()</code>	429
31.7.3.29 <code>item_next()</code>	430

31.7.3.30 item_prev()	430
31.7.3.31 item_select()	431
31.7.3.32 item_selected()	431
31.7.3.33 item_swap()	432
31.7.3.34 item_text()	432
31.7.3.35 item_width()	432
31.7.3.36 lineno()	433
31.7.3.37 lineposition()	433
31.7.3.38 load()	434
31.7.3.39 make_visible()	434
31.7.3.40 middleline()	435
31.7.3.41 move()	435
31.7.3.42 remove()	435
31.7.3.43 remove_icon()	436
31.7.3.44 replace()	436
31.7.3.45 select()	436
31.7.3.46 selected()	437
31.7.3.47 show() [1/2]	437
31.7.3.48 show() [2/2]	438
31.7.3.49 size()	438
31.7.3.50 swap() [1/2]	438
31.7.3.51 swap() [2/2]	438
31.7.3.52 text() [1/2]	439
31.7.3.53 text() [2/2]	439
31.7.3.54 textSize()	440
31.7.3.55 topline() [1/2]	440
31.7.3.56 topline() [2/2]	440
31.7.3.57 value() [1/2]	441
31.7.3.58 value() [2/2]	441
31.7.3.59 visible()	441

31.8 Fl_Browser_Class Reference	442
31.8.1 Detailed Description	445
31.8.2 Member Enumeration Documentation	445
31.8.2.1 anonymous enum	446
31.8.3 Constructor & Destructor Documentation	446
31.8.3.1 Fl_Browser_()	446
31.8.4 Member Function Documentation	447
31.8.4.1 bbox()	447
31.8.4.2 deleting()	447
31.8.4.3 deselect()	447
31.8.4.4 display()	448
31.8.4.5 displayed()	448
31.8.4.6 find_item()	449
31.8.4.7 full_height()	449
31.8.4.8 full_width()	449
31.8.4.9 handle()	449
31.8.4.10 has_scrollbar()	450
31.8.4.11 hposition() [1/2]	450
31.8.4.12 hposition() [2/2]	450
31.8.4.13 incr_height()	451
31.8.4.14 inserting()	451
31.8.4.15 item_at()	451
31.8.4.16 item_first()	453
31.8.4.17 item_height()	453
31.8.4.18 item_last()	454
31.8.4.19 item_next()	454
31.8.4.20 item_prev()	454
31.8.4.21 item_quick_height()	454
31.8.4.22 item_select()	455
31.8.4.23 item_selected()	455

31.8.4.24 item_swap()	456
31.8.4.25 item_text()	456
31.8.4.26 item_width()	456
31.8.4.27 leftedge()	457
31.8.4.28 new_list()	457
31.8.4.29 position() [1/2]	457
31.8.4.30 position() [2/2]	457
31.8.4.31 redraw_line()	458
31.8.4.32 redraw_lines()	458
31.8.4.33 replacing()	458
31.8.4.34 resize()	459
31.8.4.35 scrollbar_left()	459
31.8.4.36 scrollbar_right()	459
31.8.4.37 scrollbar_size() [1/2]	459
31.8.4.38 scrollbar_size() [2/2]	460
31.8.4.39 scrollbar_width() [1/2]	460
31.8.4.40 scrollbar_width() [2/2]	460
31.8.4.41 select()	461
31.8.4.42 select_only()	461
31.8.4.43 selection()	461
31.8.4.44 sort()	462
31.8.4.45 swapping()	462
31.8.4.46 textfont()	462
31.8.5 Member Data Documentation	463
31.8.5.1 hscrollbar	463
31.8.5.2 scrollbar	463
31.9 Fl_Button Class Reference	463
31.9.1 Detailed Description	465
31.9.2 Constructor & Destructor Documentation	465
31.9.2.1 Fl_Button()	465

31.9.3 Member Function Documentation	466
31.9.3.1 clear()	466
31.9.3.2 down_box() [1/2]	466
31.9.3.3 down_box() [2/2]	466
31.9.3.4 draw()	467
31.9.3.5 handle()	467
31.9.3.6 set()	468
31.9.3.7 shortcut() [1/2]	468
31.9.3.8 shortcut() [2/2]	468
31.9.3.9 value()	469
31.10 Fl_Cairo_State Class Reference	469
31.10.1 Detailed Description	470
31.10.2 Member Function Documentation	470
31.10.2.1 cc()	470
31.11 Fl_Cairo_Window Class Reference	470
31.11.1 Detailed Description	471
31.11.2 Member Function Documentation	471
31.11.2.1 set_draw_cb()	471
31.12 Fl_Chart Class Reference	472
31.12.1 Detailed Description	473
31.12.2 Constructor & Destructor Documentation	473
31.12.2.1 Fl_Chart()	473
31.12.3 Member Function Documentation	474
31.12.3.1 add()	474
31.12.3.2 autosize() [1/2]	474
31.12.3.3 autosize() [2/2]	474
31.12.3.4 bounds() [1/2]	475
31.12.3.5 bounds() [2/2]	475
31.12.3.6 draw()	475
31.12.3.7 insert()	476

31.12.3.8 maxsize()	476
31.12.3.9 replace()	476
31.12.3.10textcolor()	477
31.12.3.11textfont()	477
31.12.3.12textsize()	477
31.13FL_CHART_ENTRY Struct Reference	477
31.13.1 Detailed Description	478
31.13.2 Member Data Documentation	478
31.13.2.1 col	478
31.13.2.2 str	478
31.13.2.3 val	478
31.14Fl_Check_Browser Class Reference	478
31.14.1 Detailed Description	480
31.14.2 Constructor & Destructor Documentation	480
31.14.2.1 Fl_Check_Browser()	480
31.14.2.2 ~Fl_Check_Browser()	480
31.14.3 Member Function Documentation	481
31.14.3.1 add() [1/2]	481
31.14.3.2 add() [2/2]	481
31.14.3.3 check_all()	481
31.14.3.4 check_none()	481
31.14.3.5 checked() [1/2]	481
31.14.3.6 checked() [2/2]	482
31.14.3.7 clear()	482
31.14.3.8 handle()	482
31.14.3.9 item_at()	482
31.14.3.10item_first()	483
31.14.3.11item_height()	483
31.14.3.12item_next()	484
31.14.3.13item_prev()	484

31.14.3.14	item_select()	484
31.14.3.15	item_selected()	485
31.14.3.16	item_swap()	485
31.14.3.17	item_text()	485
31.14.3.18	item_width()	486
31.14.3.19	nchecked()	486
31.14.3.20	items()	486
31.14.3.21	remove()	486
31.14.3.22	set_checked()	487
31.14.3.23	text()	487
31.14.3.24	value()	487
31.15	FI_Check_Button Class Reference	487
31.15.1	Detailed Description	488
31.15.2	Constructor & Destructor Documentation	488
31.15.2.1	FI_Check_Button()	488
31.16	FI_Choice Class Reference	489
31.16.1	Detailed Description	489
31.16.2	Constructor & Destructor Documentation	491
31.16.2.1	FI_Choice()	491
31.16.3	Member Function Documentation	491
31.16.3.1	draw()	491
31.16.3.2	handle()	492
31.16.3.3	value() [1/3]	492
31.16.3.4	value() [2/3]	492
31.16.3.5	value() [3/3]	493
31.17	FI_Clock Class Reference	493
31.17.1	Detailed Description	494
31.17.2	Constructor & Destructor Documentation	495
31.17.2.1	FI_Clock() [1/2]	495
31.17.2.2	FI_Clock() [2/2]	495

31.17.3 Member Function Documentation	496
31.17.3.1 handle()	496
31.18 Fl_Clock_Output Class Reference	496
31.18.1 Detailed Description	497
31.18.2 Constructor & Destructor Documentation	498
31.18.2.1 Fl_Clock_Output()	498
31.18.3 Member Function Documentation	499
31.18.3.1 draw()	499
31.18.3.2 hour()	499
31.18.3.3 minute()	499
31.18.3.4 second()	500
31.18.3.5 shadow() [1/2]	500
31.18.3.6 shadow() [2/2]	500
31.18.3.7 value() [1/3]	501
31.18.3.8 value() [2/3]	501
31.18.3.9 value() [3/3]	501
31.19 Fl_Color_Chooser Class Reference	502
31.19.1 Detailed Description	503
31.19.2 Constructor & Destructor Documentation	504
31.19.2.1 Fl_Color_Chooser()	504
31.19.3 Member Function Documentation	504
31.19.3.1 b()	504
31.19.3.2 g()	504
31.19.3.3 handle()	505
31.19.3.4 hsv()	505
31.19.3.5 hsv2rgb()	506
31.19.3.6 hue()	506
31.19.3.7 mode() [1/2]	506
31.19.3.8 mode() [2/2]	507
31.19.3.9 r()	507

31.19.3.10gb()	507
31.19.3.11rgb2hsv()	507
31.19.3.12saturation()	508
31.19.3.13value()	508
31.20Fl_Copy_Surface Class Reference	508
31.20.1 Detailed Description	509
31.20.2 Constructor & Destructor Documentation	510
31.20.2.1 Fl_Copy_Surface()	510
31.20.3 Member Function Documentation	510
31.20.3.1 draw_decorated_window()	510
31.20.3.2 origin() [1/2]	511
31.20.3.3 origin() [2/2]	511
31.20.3.4 printable_rect()	511
31.20.3.5 set_current()	512
31.20.3.6 translate()	512
31.21Fl_Counter Class Reference	512
31.21.1 Detailed Description	513
31.21.2 Constructor & Destructor Documentation	514
31.21.2.1 Fl_Counter()	514
31.21.3 Member Function Documentation	514
31.21.3.1 draw()	515
31.21.3.2 handle()	515
31.21.3.3 lstep()	516
31.21.3.4 step() [1/2]	516
31.21.3.5 step() [2/2]	516
31.22Fl_Device_Plugin Class Reference	516
31.22.1 Detailed Description	517
31.22.2 Member Function Documentation	517
31.22.2.1 rectangle_capture()	518
31.23Fl_Dial Class Reference	518

31.23.1 Detailed Description	519
31.23.2 Constructor & Destructor Documentation	519
31.23.2.1 Fl_Dial()	519
31.23.3 Member Function Documentation	520
31.23.3.1 angle1()	520
31.23.3.2 draw()	520
31.23.3.3 handle()	520
31.24 Fl_Display_Device Class Reference	521
31.24.1 Detailed Description	521
31.25 Fl_Double_Window Class Reference	521
31.25.1 Detailed Description	522
31.25.2 Constructor & Destructor Documentation	522
31.25.2.1 ~Fl_Double_Window()	522
31.25.3 Member Function Documentation	523
31.25.3.1 flush()	523
31.25.3.2 hide()	523
31.25.3.3 resize()	523
31.25.3.4 show()	524
31.26 Fl_End Class Reference	524
31.26.1 Detailed Description	525
31.27 Fl_EPS_File_Surface Class Reference	525
31.27.1 Detailed Description	526
31.27.2 Constructor & Destructor Documentation	526
31.27.2.1 Fl_EPS_File_Surface()	526
31.27.2.2 ~Fl_EPS_File_Surface()	527
31.27.3 Member Function Documentation	527
31.27.3.1 close()	527
31.27.3.2 driver()	527
31.27.3.3 origin() [1/2]	527
31.27.3.4 origin() [2/2]	528

31.27.3.5 printable_rect()	528
31.27.3.6 translate()	528
31.28 Fl_File_Browser Class Reference	529
31.28.1 Detailed Description	530
31.28.2 Constructor & Destructor Documentation	530
31.28.2.1 Fl_File_Browser()	530
31.28.3 Member Function Documentation	530
31.28.3.1 errmsg() [1/2]	530
31.28.3.2 errmsg() [2/2]	530
31.28.3.3 filetype() [1/2]	531
31.28.3.4 filetype() [2/2]	531
31.28.3.5 filter() [1/2]	531
31.28.3.6 filter() [2/2]	531
31.28.3.7 iconsize() [1/2]	531
31.28.3.8 iconsize() [2/2]	531
31.28.3.9 load()	532
31.29 Fl_File_Chooser Class Reference	532
31.29.1 Detailed Description	535
31.29.2 Constructor & Destructor Documentation	537
31.29.2.1 Fl_File_Chooser()	537
31.29.2.2 ~Fl_File_Chooser()	537
31.29.3 Member Function Documentation	538
31.29.3.1 add_extra()	538
31.29.3.2 color() [1/2]	538
31.29.3.3 color() [2/2]	539
31.29.3.4 count()	539
31.29.3.5 directory() [1/2]	539
31.29.3.6 directory() [2/2]	539
31.29.3.7 filter() [1/2]	539
31.29.3.8 filter() [2/2]	540

31.29.3.9 filter_value() [1/2]	540
31.29.3.10filter_value() [2/2]	540
31.29.3.11hide()	540
31.29.3.12consize() [1/2]	540
31.29.3.13consize() [2/2]	540
31.29.3.14label() [1/2]	541
31.29.3.15label() [2/2]	541
31.29.3.16preview() [1/2]	541
31.29.3.17preview() [2/2]	541
31.29.3.18escan()	541
31.29.3.19show()	541
31.29.3.20shown()	542
31.29.3.21textcolor() [1/2]	542
31.29.3.22textcolor() [2/2]	542
31.29.3.23textfont() [1/2]	542
31.29.3.24textfont() [2/2]	542
31.29.3.25textsize() [1/2]	543
31.29.3.26textsize() [2/2]	543
31.29.3.27type() [1/2]	543
31.29.3.28type() [2/2]	543
31.29.3.29user_data()	543
31.29.3.30value()	544
31.29.3.31visible()	544
31.29.4 Member Data Documentation	544
31.29.4.1 showHiddenButton	544
31.30 Fl_File_Icon Class Reference	544
31.30.1 Detailed Description	546
31.30.2 Constructor & Destructor Documentation	546
31.30.2.1 Fl_File_Icon()	546
31.30.3 Member Function Documentation	546

31.30.3.1 add()	546
31.30.3.2 add_color()	547
31.30.3.3 add_vertex() [1/2]	547
31.30.3.4 add_vertex() [2/2]	547
31.30.3.5 clear()	548
31.30.3.6 draw()	548
31.30.3.7 find()	548
31.30.3.8 first()	549
31.30.3.9 label()	549
31.30.3.10abeltype()	549
31.30.3.11load()	549
31.30.3.12oad_ftl()	550
31.30.3.13oad_image()	550
31.30.3.14oad_system_icons()	550
31.30.3.15next()	551
31.30.3.16attern()	551
31.30.3.17size()	551
31.30.3.18type()	551
31.30.3.19value()	551
31.31 Fl_File_Input Class Reference	552
31.31.1 Detailed Description	553
31.31.2 Constructor & Destructor Documentation	553
31.31.2.1 Fl_File_Input()	553
31.31.3 Member Function Documentation	553
31.31.3.1 down_box() [1/2]	554
31.31.3.2 down_box() [2/2]	554
31.31.3.3 errorcolor() [1/2]	554
31.31.3.4 errorcolor() [2/2]	554
31.31.3.5 handle()	554
31.31.3.6 value() [1/2]	555

31.31.3.7 value() [2/2]	555
31.32 Fl_Fill_Dial Class Reference	555
31.32.1 Detailed Description	556
31.32.2 Constructor & Destructor Documentation	556
31.32.2.1 Fl_Fill_Dial()	556
31.33 Fl_Fill_Slider Class Reference	557
31.33.1 Detailed Description	557
31.33.2 Constructor & Destructor Documentation	557
31.33.2.1 Fl_Fill_Slider()	557
31.34 Fl_Float_Input Class Reference	558
31.34.1 Detailed Description	558
31.34.2 Constructor & Destructor Documentation	558
31.34.2.1 Fl_Float_Input()	558
31.35 Fl_FormsBitmap Class Reference	559
31.35.1 Detailed Description	559
31.35.2 Member Function Documentation	559
31.35.2.1 bitmap() [1/2]	559
31.35.2.2 bitmap() [2/2]	560
31.35.2.3 draw()	560
31.35.2.4 set()	560
31.36 Fl_FormsPixmap Class Reference	560
31.36.1 Detailed Description	561
31.36.2 Constructor & Destructor Documentation	561
31.36.2.1 Fl_FormsPixmap()	561
31.36.3 Member Function Documentation	562
31.36.3.1 draw()	562
31.36.3.2 Pixmap() [1/2]	562
31.36.3.3 Pixmap() [2/2]	562
31.36.3.4 set()	562
31.37 Fl_FormsText Class Reference	563

31.37.1 Member Function Documentation	563
31.37.1.1 draw()	563
31.38 Fl_Free Class Reference	564
31.38.1 Detailed Description	564
31.38.2 Constructor & Destructor Documentation	565
31.38.2.1 Fl_Free()	565
31.38.3 Member Function Documentation	565
31.38.3.1 draw()	566
31.38.3.2 handle()	566
31.39 Fl_GIF_Image Class Reference	567
31.39.1 Detailed Description	567
31.39.2 Constructor & Destructor Documentation	567
31.39.2.1 Fl_GIF_Image() [1/2]	567
31.39.2.2 Fl_GIF_Image() [2/2]	568
31.40 Fl_GL_Choice Class Reference	568
31.41 Fl_GL_Window Class Reference	569
31.41.1 Detailed Description	571
31.41.2 Constructor & Destructor Documentation	571
31.41.2.1 Fl_GL_Window() [1/2]	572
31.41.2.2 Fl_GL_Window() [2/2]	572
31.41.3 Member Function Documentation	572
31.41.3.1 as_gl_window()	572
31.41.3.2 can_do() [1/3]	573
31.41.3.3 can_do() [2/3]	573
31.41.3.4 can_do() [3/3]	573
31.41.3.5 can_do_overlay()	573
31.41.3.6 context() [1/2]	573
31.41.3.7 context() [2/2]	574
31.41.3.8 context_valid()	574
31.41.3.9 draw()	574

31.41.3.10flush()	575
31.41.3.11hide_overlay()	575
31.41.3.12make_current()	575
31.41.3.13make_overlay_current()	575
31.41.3.14mode() [1/3]	576
31.41.3.15mode() [2/3]	576
31.41.3.16mode() [3/3]	576
31.41.3.17ortho()	577
31.41.3.18pixel_h()	577
31.41.3.19pixel_w()	577
31.41.3.20pixels_per_unit()	578
31.41.3.21redraw_overlay()	578
31.41.3.22resize()	578
31.41.3.23show()	579
31.41.3.24swap_buffers()	579
31.41.3.25valid()	579
31.42Fl_Glut_Bitmap_Font Struct Reference	580
31.42.1 Detailed Description	580
31.43Fl_Glut_StrokeChar Struct Reference	580
31.44Fl_Glut_StrokeFont Struct Reference	580
31.45Fl_Glut_StrokeStrip Struct Reference	581
31.46Fl_Glut_StrokeVertex Struct Reference	581
31.47Fl_Glut_Window Class Reference	581
31.47.1 Detailed Description	582
31.47.2 Constructor & Destructor Documentation	582
31.47.2.1 Fl_Glut_Window() [1/2]	583
31.47.2.2 Fl_Glut_Window() [2/2]	583
31.47.3 Member Function Documentation	583
31.47.3.1 draw()	583
31.47.3.2 draw_overlay()	584

31.48 Fl_Group Class Reference	584
31.48.1 Detailed Description	587
31.48.2 Constructor & Destructor Documentation	587
31.48.2.1 Fl_Group()	587
31.48.2.2 ~Fl_Group()	588
31.48.3 Member Function Documentation	588
31.48.3.1 array()	588
31.48.3.2 as_group()	588
31.48.3.3 begin()	589
31.48.3.4 bounds()	589
31.48.3.5 child()	590
31.48.3.6 clear()	590
31.48.3.7 clip_children() [1/2]	590
31.48.3.8 clip_children() [2/2]	591
31.48.3.9 current() [1/2]	591
31.48.3.10 current() [2/2]	591
31.48.3.11 draw()	591
31.48.3.12 draw_child()	592
31.48.3.13 draw_children()	592
31.48.3.14 draw_outside_label()	592
31.48.3.15 end()	592
31.48.3.16 find()	592
31.48.3.17 focus()	593
31.48.3.18 handle()	593
31.48.3.19 init_sizes()	594
31.48.3.20 insert() [1/2]	594
31.48.3.21 insert() [2/2]	594
31.48.3.22 remove() [1/3]	594
31.48.3.23 remove() [2/3]	595
31.48.3.24 remove() [3/3]	595

31.48.3.25 resizable() [1/3]	595
31.48.3.26 resizable() [2/3]	596
31.48.3.27 resizable() [3/3]	597
31.48.3.28 resize()	597
31.48.3.29 sizes()	597
31.48.3.30 update_child()	598
31.49 Fl_Help_Block Struct Reference	598
31.50 Fl_Help_Dialog Class Reference	599
31.50.1 Detailed Description	600
31.50.2 Constructor & Destructor Documentation	600
31.50.2.1 Fl_Help_Dialog()	600
31.50.3 Member Function Documentation	600
31.50.3.1 h()	600
31.50.3.2 hide()	601
31.50.3.3 load()	601
31.50.3.4 position()	601
31.50.3.5 resize()	601
31.50.3.6 show()	602
31.50.3.7 textsize() [1/2]	602
31.50.3.8 textsize() [2/2]	602
31.50.3.9 value() [1/2]	602
31.50.3.10 value() [2/2]	602
31.50.3.11 visible()	602
31.50.3.12 w()	603
31.50.3.13 x()	603
31.50.3.14 y()	603
31.51 Fl_Help_Font_Stack Struct Reference	603
31.51.1 Constructor & Destructor Documentation	604
31.51.1.1 Fl_Help_Font_Stack()	604
31.51.2 Member Function Documentation	604

31.51.2.1 count()	604
31.51.2.2 top()	604
31.52 Fl_Help_Font_Style Struct Reference	604
31.52.1 Detailed Description	605
31.53 Fl_Help_Link Struct Reference	605
31.53.1 Detailed Description	605
31.54 Fl_Help_Target Struct Reference	606
31.54.1 Detailed Description	606
31.55 Fl_Help_View Class Reference	606
31.55.1 Detailed Description	608
31.55.2 Constructor & Destructor Documentation	610
31.55.2.1 ~Fl_Help_View()	610
31.55.3 Member Function Documentation	610
31.55.3.1 clear_selection()	610
31.55.3.2 directory()	610
31.55.3.3 draw()	610
31.55.3.4 filename()	610
31.55.3.5 find()	611
31.55.3.6 handle()	611
31.55.3.7 leftline() [1/2]	611
31.55.3.8 leftline() [2/2]	611
31.55.3.9 link()	612
31.55.3.10 load()	612
31.55.3.11 resize()	613
31.55.3.12 scrollbar_size() [1/2]	613
31.55.3.13 scrollbar_size() [2/2]	613
31.55.3.14 select_all()	614
31.55.3.15 size()	614
31.55.3.16 textcolor() [1/2]	614
31.55.3.17 textcolor() [2/2]	614

31.55.3.18 <code>textfont()</code> [1/2]	614
31.55.3.19 <code>textfont()</code> [2/2]	615
31.55.3.20 <code>textsize()</code> [1/2]	615
31.55.3.21 <code>textsize()</code> [2/2]	615
31.55.3.22 <code>title()</code>	615
31.55.3.23 <code>topline()</code> [1/3]	615
31.55.3.24 <code>topline()</code> [2/3]	616
31.55.3.25 <code>topline()</code> [3/3]	616
31.55.3.26 <code>value()</code> [1/2]	616
31.55.3.27 <code>value()</code> [2/2]	616
31.56 <code>FI_Hold_Browser</code> Class Reference	617
31.56.1 Detailed Description	617
31.56.2 Constructor & Destructor Documentation	617
31.56.2.1 <code>FI_Hold_Browser()</code>	618
31.57 <code>FI_Hor_Fill_Slider</code> Class Reference	618
31.58 <code>FI_Hor_Nice_Slider</code> Class Reference	619
31.59 <code>FI_Hor_Slider</code> Class Reference	619
31.59.1 Detailed Description	620
31.60 <code>FI_Hor_Value_Slider</code> Class Reference	620
31.61 <code>FI_Image</code> Class Reference	621
31.61.1 Detailed Description	623
31.61.2 Constructor & Destructor Documentation	623
31.61.2.1 <code>FI_Image()</code>	624
31.61.3 Member Function Documentation	624
31.61.3.1 <code>as_shared_image()</code>	624
31.61.3.2 <code>color_average()</code>	624
31.61.3.3 <code>copy()</code> [1/2]	624
31.61.3.4 <code>copy()</code> [2/2]	625
31.61.3.5 <code>count()</code>	625
31.61.3.6 <code>d()</code>	625

31.61.3.7 <code>data()</code>	625
31.61.3.8 <code>desaturate()</code>	625
31.61.3.9 <code>draw()</code> [1/2]	626
31.61.3.10 <code>draw()</code> [2/2]	626
31.61.3.11 <code>draw_empty()</code>	626
31.61.3.12 <code>draw_scaled()</code>	627
31.61.3.13 <code>fail()</code>	627
31.61.3.14 <code>h()</code> [1/2]	628
31.61.3.15 <code>h()</code> [2/2]	628
31.61.3.16 <code>inactive()</code>	628
31.61.3.17 <code>label()</code> [1/2]	628
31.61.3.18 <code>label()</code> [2/2]	628
31.61.3.19 <code>d()</code> [1/2]	629
31.61.3.20 <code>d()</code> [2/2]	629
31.61.3.21 <code>release()</code>	629
31.61.3.22 <code>RGB_scaling()</code> [1/2]	630
31.61.3.23 <code>RGB_scaling()</code> [2/2]	630
31.61.3.24 <code>scale()</code>	630
31.61.3.25 <code>scaling_algorithm()</code> [1/2]	631
31.61.3.26 <code>scaling_algorithm()</code> [2/2]	631
31.61.3.27 <code>uncache()</code>	631
31.61.3.28 <code>w()</code> [1/2]	631
31.61.3.29 <code>w()</code> [2/2]	632
31.62 <code>FI_Image_Reader</code> Class Reference	632
31.63 <code>FI_Image_Surface</code> Class Reference	632
31.63.1 Detailed Description	633
31.63.2 Constructor & Destructor Documentation	634
31.63.2.1 <code>FI_Image_Surface()</code>	634
31.63.2.2 <code>~FI_Image_Surface()</code>	635
31.63.3 Member Function Documentation	635

31.63.3.1 highres_image()	635
31.63.3.2 image()	635
31.63.3.3 offscreen()	635
31.63.3.4 origin() [1/2]	635
31.63.3.5 origin() [2/2]	636
31.63.3.6 printable_rect()	636
31.63.3.7 rescale()	637
31.63.3.8 set_current()	637
31.63.3.9 translate()	637
31.64 Fl_Input Class Reference	638
31.64.1 Detailed Description	638
31.64.2 Constructor & Destructor Documentation	640
31.64.2.1 Fl_Input()	640
31.64.3 Member Function Documentation	640
31.64.3.1 draw()	640
31.64.3.2 handle()	640
31.65 Fl_Input_ Class Reference	641
31.65.1 Detailed Description	644
31.65.2 Constructor & Destructor Documentation	644
31.65.2.1 Fl_Input_()	644
31.65.2.2 ~Fl_Input_()	645
31.65.3 Member Function Documentation	645
31.65.3.1 append()	645
31.65.3.2 copy()	645
31.65.3.3 copy_cuts()	646
31.65.3.4 cursor_color() [1/2]	646
31.65.3.5 cursor_color() [2/2]	646
31.65.3.6 cut() [1/3]	647
31.65.3.7 cut() [2/3]	647
31.65.3.8 cut() [3/3]	647

31.65.3.9 drawtext()	648
31.65.3.10 handle_mouse()	648
31.65.3.11 handleText()	649
31.65.3.12 index()	649
31.65.3.13 input_type() [1/2]	649
31.65.3.14 input_type() [2/2]	649
31.65.3.15 insert()	650
31.65.3.16 line_end()	650
31.65.3.17 line_start()	651
31.65.3.18 mark() [1/2]	651
31.65.3.19 mark() [2/2]	651
31.65.3.20 maximum_size() [1/2]	652
31.65.3.21 maximum_size() [2/2]	652
31.65.3.22 position() [1/3]	652
31.65.3.23 position() [2/3]	652
31.65.3.24 position() [3/3]	653
31.65.3.25 readonly()	653
31.65.3.26 readonly()	654
31.65.3.27 replace()	654
31.65.3.28 resize()	655
31.65.3.29 shortcut() [1/2]	655
31.65.3.30 shortcut() [2/2]	656
31.65.3.31 size()	656
31.65.3.32 size()	656
31.65.3.33 static_value()	657
31.65.3.34 static_value()	657
31.65.3.35 tab_nav()	658
31.65.3.36 tab_nav()	658
31.65.3.37 textColor()	659
31.65.3.38 textColor()	659

31.65.3.39textfont() [1/2]	659
31.65.3.40textfont() [2/2]	659
31.65.3.41textsize() [1/2]	660
31.65.3.42textsize() [2/2]	660
31.65.3.43undo()	660
31.65.3.44up_down_position()	661
31.65.3.45value() [1/3]	661
31.65.3.46value() [2/3]	662
31.65.3.47value() [3/3]	662
31.65.3.48word_end()	662
31.65.3.49word_start()	663
31.65.3.50wrap() [1/2]	663
31.65.3.51wrap() [2/2]	663
31.66Fl_Input_Choice Class Reference	664
31.66.1 Detailed Description	665
31.66.2 Constructor & Destructor Documentation	666
31.66.2.1 Fl_Input_Choice()	666
31.66.3 Member Function Documentation	666
31.66.3.1 add()	667
31.66.3.2 changed()	667
31.66.3.3 clear()	667
31.66.3.4 clear_changed()	667
31.66.3.5 input()	667
31.66.3.6 menu() [1/2]	668
31.66.3.7 menu() [2/2]	668
31.66.3.8 menubutton()	668
31.66.3.9 update_menubutton()	668
31.66.3.10value() [1/2]	669
31.66.3.11value() [2/2]	669
31.67Fl_Int_Input Class Reference	670

31.67.1 Detailed Description	670
31.67.2 Constructor & Destructor Documentation	670
31.67.2.1 Fl_Int_Input()	670
31.68 Fl_JPEG_Image Class Reference	671
31.68.1 Detailed Description	671
31.68.2 Constructor & Destructor Documentation	671
31.68.2.1 Fl_JPEG_Image() [1/2]	671
31.68.2.2 Fl_JPEG_Image() [2/2]	672
31.69 Fl_Label Struct Reference	672
31.69.1 Detailed Description	673
31.69.2 Member Function Documentation	673
31.69.2.1 draw()	673
31.69.2.2 measure()	674
31.69.3 Member Data Documentation	674
31.69.3.1 type	674
31.70 Fl_Light_Button Class Reference	674
31.70.1 Detailed Description	675
31.70.2 Constructor & Destructor Documentation	675
31.70.2.1 Fl_Light_Button()	675
31.70.3 Member Function Documentation	676
31.70.3.1 draw()	676
31.70.3.2 handle()	676
31.71 Fl_Line_Dial Class Reference	677
31.72 Fl_Mac_App_Menu Class Reference	677
31.72.1 Member Function Documentation	678
31.72.1.1 custom_application_menu_items()	678
31.72.2 Member Data Documentation	678
31.72.2.1 print	678
31.73 Fl_Menu_Class Reference	679
31.73.1 Detailed Description	681

31.73.2 Constructor & Destructor Documentation	682
31.73.2.1 Fl_Menu_()	682
31.73.3 Member Function Documentation	682
31.73.3.1 add() [1/2]	682
31.73.3.2 add() [2/2]	684
31.73.3.3 clear()	685
31.73.3.4 clear_submenu()	685
31.73.3.5 copy()	686
31.73.3.6 down_box()	686
31.73.3.7 find_index() [1/3]	686
31.73.3.8 find_index() [2/3]	687
31.73.3.9 find_index() [3/3]	687
31.73.3.10 find_item() [1/2]	688
31.73.3.11 find_item() [2/2]	689
31.73.3.12 global()	689
31.73.3.13 insert()	689
31.73.3.14 item.pathname()	690
31.73.3.15 menu() [1/2]	691
31.73.3.16 menu() [2/2]	691
31.73.3.17 menu_end()	692
31.73.3.18 mode() [1/2]	692
31.73.3.19 mode() [2/2]	692
31.73.3.20 mvalue()	693
31.73.3.21 picked()	693
31.73.3.22 remove()	693
31.73.3.23 replace()	693
31.73.3.24 setonly()	694
31.73.3.25 shortcut()	694
31.73.3.26 size()	694
31.73.3.27 test_shortcut()	694

31.73.3.28text() [1/2]	695
31.73.3.29text() [2/2]	695
31.73.3.30textcolor() [1/2]	695
31.73.3.31textcolor() [2/2]	695
31.73.3.32textfont() [1/2]	695
31.73.3.33textfont() [2/2]	695
31.73.3.34textsize() [1/2]	696
31.73.3.35textsize() [2/2]	696
31.73.3.36value() [1/3]	696
31.73.3.37value() [2/3]	696
31.73.3.38value() [3/3]	696
31.74 Fl_Menu_Bar Class Reference	697
31.74.1 Detailed Description	697
31.74.2 Constructor & Destructor Documentation	698
31.74.2.1 Fl_Menu_Bar()	698
31.74.3 Member Function Documentation	699
31.74.3.1 draw()	699
31.74.3.2 handle()	699
31.74.3.3 update()	700
31.75 Fl_Menu_Button Class Reference	700
31.75.1 Detailed Description	701
31.75.2 Member Enumeration Documentation	702
31.75.2.1 popup_buttons	702
31.75.3 Constructor & Destructor Documentation	702
31.75.3.1 Fl_Menu_Button()	702
31.75.4 Member Function Documentation	702
31.75.4.1 draw()	703
31.75.4.2 handle()	703
31.75.4.3 popup()	704
31.76 Fl_Menu_Item Struct Reference	704

31.76.1 Detailed Description	707
31.76.2 Member Function Documentation	708
31.76.2.1 activate()	708
31.76.2.2 active()	709
31.76.2.3 activevisible()	709
31.76.2.4 add()	709
31.76.2.5 argument() [1/2]	709
31.76.2.6 argument() [2/2]	710
31.76.2.7 callback() [1/5]	710
31.76.2.8 callback() [2/5]	710
31.76.2.9 callback() [3/5]	710
31.76.2.10callback() [4/5]	711
31.76.2.11callback() [5/5]	711
31.76.2.12check()	711
31.76.2.13checkbox()	711
31.76.2.14checked()	712
31.76.2.15clear()	712
31.76.2.16deactivate()	712
31.76.2.17do_callback() [1/3]	712
31.76.2.18do_callback() [2/3]	712
31.76.2.19do_callback() [3/3]	713
31.76.2.20draw()	713
31.76.2.21find_shortcut()	713
31.76.2.22first() [1/2]	713
31.76.2.23first() [2/2]	714
31.76.2.24hide()	714
31.76.2.25image() [1/2]	714
31.76.2.26image() [2/2]	714
31.76.2.27insert()	714
31.76.2.28label()	715

31.76.2.29labelcolor() [1/2]	715
31.76.2.30labelcolor() [2/2]	715
31.76.2.31labelfont() [1/2]	716
31.76.2.32labelfont() [2/2]	716
31.76.2.33labelsize() [1/2]	716
31.76.2.34labelsize() [2/2]	716
31.76.2.35abeltype() [1/2]	716
31.76.2.36abeltype() [2/2]	717
31.76.2.37measure()	717
31.76.2.38next() [1/2]	717
31.76.2.39next() [2/2]	717
31.76.2.40popup()	717
31.76.2.41pulldown()	718
31.76.2.42radio()	718
31.76.2.43set()	718
31.76.2.44setonly()	719
31.76.2.45shortcut() [1/2]	719
31.76.2.46shortcut() [2/2]	719
31.76.2.47show()	719
31.76.2.48size()	720
31.76.2.49submenu()	720
31.76.2.50test_shortcut()	720
31.76.2.51uncheck()	720
31.76.2.52value()	721
31.76.2.53visible()	721
31.77 FI_Menu_Window Class Reference	721
31.77.1 Detailed Description	722
31.77.2 Constructor & Destructor Documentation	722
31.77.2.1 ~FI_Menu_Window()	722
31.77.2.2 FI_Menu_Window() [1/2]	723

31.77.2.3 Fl_Menu_Window() [2/2]	723
31.77.3 Member Function Documentation	723
31.77.3.1 clear_overlay()	723
31.77.3.2 flush()	723
31.77.3.3 hide()	724
31.77.3.4 set_overlay()	724
31.77.3.5 show()	724
31.78 Fl_Multi_Browser Class Reference	725
31.78.1 Detailed Description	725
31.78.2 Constructor & Destructor Documentation	725
31.78.2.1 Fl_Multi_Browser()	726
31.79 Fl_Multi_Label Struct Reference	726
31.79.1 Detailed Description	727
31.79.2 Member Function Documentation	727
31.79.2.1 label() [1/2]	728
31.79.2.2 label() [2/2]	728
31.79.3 Member Data Documentation	728
31.79.3.1 labela	728
31.79.3.2 labelb	728
31.79.3.3 typea	728
31.79.3.4 typeb	729
31.80 Fl_Multiline_Input Class Reference	729
31.80.1 Detailed Description	729
31.80.2 Constructor & Destructor Documentation	730
31.80.2.1 Fl_Multiline_Input()	730
31.81 Fl_Multiline_Output Class Reference	730
31.81.1 Detailed Description	731
31.81.2 Constructor & Destructor Documentation	731
31.81.2.1 Fl_Multiline_Output()	731
31.82 Fl_Native_File_Chooser Class Reference	731

31.82.1 Detailed Description	733
31.82.2 Member Enumeration Documentation	734
31.82.2.1 Option	734
31.82.2.2 Type	734
31.82.3 Constructor & Destructor Documentation	735
31.82.3.1 Fl_Native_File_Chooser()	735
31.82.3.2 ~Fl_Native_File_Chooser()	735
31.82.4 Member Function Documentation	735
31.82.4.1 count()	735
31.82.4.2 directory()	736
31.82.4.3 errmsg()	736
31.82.4.4 filename() [1/2]	736
31.82.4.5 filename() [2/2]	736
31.82.4.6 filter() [1/2]	736
31.82.4.7 filter() [2/2]	737
31.82.4.8 filter_value() [1/2]	737
31.82.4.9 filter_value() [2/2]	737
31.82.4.10 options()	737
31.82.4.11 preset_file()	738
31.82.4.12 show()	738
31.82.4.13 title() [1/2]	738
31.82.4.14 title() [2/2]	738
31.83 Fl_Nice_Slider Class Reference	739
31.84 Fl_Output Class Reference	739
31.84.1 Detailed Description	740
31.84.2 Constructor & Destructor Documentation	740
31.84.2.1 Fl_Output()	741
31.85 Fl_Overlay_Window Class Reference	741
31.85.1 Detailed Description	742
31.85.2 Constructor & Destructor Documentation	742

31.85.2.1 Fl_Overlay_Window()	742
31.85.3 Member Function Documentation	743
31.85.3.1 draw_overlay()	743
31.85.3.2 flush()	743
31.85.3.3 hide()	743
31.85.3.4 redraw_overlay()	743
31.85.3.5 resize()	744
31.85.3.6 show()	744
31.86 Fl_Pack Class Reference	745
31.86.1 Detailed Description	745
31.86.2 Constructor & Destructor Documentation	746
31.86.2.1 Fl_Pack()	746
31.86.3 Member Function Documentation	746
31.86.3.1 draw()	746
31.86.3.2 horizontal()	747
31.87 Fl_Paged_Device Class Reference	747
31.87.1 Detailed Description	749
31.87.2 Member Enumeration Documentation	749
31.87.2.1 Page_Format	749
31.87.2.2 Page_Layout	749
31.87.3 Member Function Documentation	750
31.87.3.1 begin_job()	750
31.87.3.2 begin_page()	750
31.87.3.3 end_page()	751
31.87.3.4 margins()	751
31.87.3.5 rotate()	751
31.87.3.6 scale()	752
31.87.3.7 start_job()	752
31.87.3.8 start_page()	752
31.88 Fl_Pixmap Class Reference	753

31.88.1 Detailed Description	754
31.88.2 Constructor & Destructor Documentation	754
31.88.2.1 Fl_Pixmap() [1/4]	754
31.88.2.2 Fl_Pixmap() [2/4]	754
31.88.2.3 Fl_Pixmap() [3/4]	754
31.88.2.4 Fl_Pixmap() [4/4]	755
31.88.3 Member Function Documentation	755
31.88.3.1 color_average()	755
31.88.3.2 copy()	755
31.88.3.3 desaturate()	755
31.88.3.4 draw()	756
31.88.3.5 label() [1/2]	756
31.88.3.6 label() [2/2]	756
31.88.3.7 uncache()	757
31.89 Fl_Plugin Class Reference	757
31.89.1 Detailed Description	757
31.89.2 Constructor & Destructor Documentation	758
31.89.2.1 Fl_Plugin()	758
31.90 Fl_Plugin_Manager Class Reference	758
31.90.1 Detailed Description	759
31.90.2 Constructor & Destructor Documentation	759
31.90.2.1 ~Fl_Plugin_Manager()	759
31.90.3 Member Function Documentation	759
31.90.3.1 addPlugin()	759
31.90.3.2 load()	760
31.90.3.3 removePlugin()	760
31.91 Fl_PNG_Image Class Reference	760
31.91.1 Detailed Description	761
31.91.2 Constructor & Destructor Documentation	761
31.91.2.1 Fl_PNG_Image() [1/2]	761

31.91.2.2 Fl_PNG_Image() [2/2]	761
31.92 Fl_PNM_Image Class Reference	762
31.92.1 Detailed Description	762
31.92.2 Constructor & Destructor Documentation	762
31.92.2.1 Fl_PNM_Image()	762
31.93 Fl_Positioner Class Reference	763
31.93.1 Detailed Description	764
31.93.2 Constructor & Destructor Documentation	764
31.93.2.1 Fl_Positioner()	765
31.93.3 Member Function Documentation	765
31.93.3.1 draw()	765
31.93.3.2 handle()	765
31.93.3.3 value()	766
31.93.3.4 xbounds()	766
31.93.3.5 xstep()	766
31.93.3.6 xvalue() [1/2]	767
31.93.3.7 xvalue() [2/2]	767
31.93.3.8 ybounds()	767
31.93.3.9 ystep()	767
31.93.3.10 value() [1/2]	767
31.93.3.11 value() [2/2]	768
31.94 Fl_PostScript_File_Device Class Reference	768
31.94.1 Detailed Description	769
31.94.2 Member Function Documentation	770
31.94.2.1 begin_job() [1/3]	770
31.94.2.2 begin_job() [2/3]	770
31.94.2.3 begin_job() [3/3]	771
31.94.2.4 begin_page()	771
31.94.2.5 end_page()	772
31.94.2.6 margins()	772

31.94.2.7 origin() [1/2]	772
31.94.2.8 origin() [2/2]	773
31.94.2.9 printable_rect()	773
31.94.2.10 rotate()	773
31.94.2.11 scale()	774
31.94.2.12 start_job() [1/2]	774
31.94.2.13 start_job() [2/2]	774
31.94.2.14 translate()	775
31.95 Fl_Preferences Class Reference	775
31.95.1 Detailed Description	779
31.95.2 Member Typedef Documentation	779
31.95.2.1 ID	779
31.95.3 Member Enumeration Documentation	779
31.95.3.1 Root	779
31.95.4 Constructor & Destructor Documentation	780
31.95.4.1 Fl_Preferences() [1/7]	780
31.95.4.2 Fl_Preferences() [2/7]	781
31.95.4.3 Fl_Preferences() [3/7]	782
31.95.4.4 Fl_Preferences() [4/7]	782
31.95.4.5 Fl_Preferences() [5/7]	782
31.95.4.6 Fl_Preferences() [6/7]	783
31.95.4.7 Fl_Preferences() [7/7]	783
31.95.4.8 ~Fl_Preferences()	783
31.95.5 Member Function Documentation	783
31.95.5.1 deleteEntry()	783
31.95.5.2 deleteGroup()	784
31.95.5.3 entries()	784
31.95.5.4 entry()	784
31.95.5.5 entryExists()	785
31.95.5.6 file_access() [1/2]	785

31.95.5.7 file_access() [2/2]	786
31.95.5.8 flush()	786
31.95.5.9 get() [1/7]	786
31.95.5.10get() [2/7]	787
31.95.5.11get() [3/7]	787
31.95.5.12get() [4/7]	788
31.95.5.13get() [5/7]	788
31.95.5.14get() [6/7]	789
31.95.5.15get() [7/7]	789
31.95.5.16getUserdataPath()	790
31.95.5.17group()	791
31.95.5.18groupExists()	791
31.95.5.19groups()	791
31.95.5.20newUUID()	792
31.95.5.21set() [1/7]	792
31.95.5.22set() [2/7]	792
31.95.5.23set() [3/7]	793
31.95.5.24set() [4/7]	793
31.95.5.25set() [5/7]	794
31.95.5.26set() [6/7]	794
31.95.5.27set() [7/7]	795
31.95.5.28size()	795
31.95.6 Member Data Documentation	795
31.95.6.1 CORE_READ_OK	796
31.95.6.2 CORE_WRITE_OK	796
31.95.6.3 NONE	796
31.96 FI_Printer Class Reference	796
31.96.1 Detailed Description	798
31.96.2 Member Function Documentation	799
31.96.2.1 begin_job()	799

31.96.2.2 begin_page()	800
31.96.2.3 draw_decorated_window()	800
31.96.2.4 end_page()	800
31.96.2.5 margins()	800
31.96.2.6 origin() [1/2]	801
31.96.2.7 origin() [2/2]	801
31.96.2.8 printable_rect()	802
31.96.2.9 rotate()	802
31.96.2.10 scale()	802
31.96.2.11 set_current()	803
31.96.2.12 translate()	803
31.97 Fl_Progress Class Reference	803
31.97.1 Detailed Description	804
31.97.2 Constructor & Destructor Documentation	804
31.97.2.1 Fl_Progress()	804
31.97.3 Member Function Documentation	805
31.97.3.1 draw()	805
31.97.3.2 maximum() [1/2]	805
31.97.3.3 maximum() [2/2]	805
31.97.3.4 minimum() [1/2]	805
31.97.3.5 minimum() [2/2]	805
31.97.3.6 value() [1/2]	806
31.97.3.7 value() [2/2]	806
31.98 Fl_Radio_Button Class Reference	806
31.98.1 Constructor & Destructor Documentation	806
31.98.1.1 Fl_Radio_Button()	806
31.99 Fl_Radio_Light_Button Class Reference	807
31.10 Fl_Radio_Round_Button Class Reference	807
31.10.1 Constructor & Destructor Documentation	808
31.100.1.1 Fl_Radio_Round_Button()	808

31.10 FI_Rect Class Reference	808
31.101. Detailed Description	809
31.101.2. Constructor & Destructor Documentation	809
31.101.2.1 FI_Rect() [1/5]	809
31.101.2.2 FI_Rect() [2/5]	810
31.101.2.3 FI_Rect() [3/5]	810
31.101.2.4 FI_Rect() [4/5]	810
31.101.2.5 FI_Rect() [5/5]	810
31.101.3. Member Function Documentation	810
31.101.3.1 b()	810
31.101.3.2 ()	811
31.10 FI_Scroll::ScrollInfo::FI_Region_LRTB Struct Reference	811
31.102. Detailed Description	811
31.10 FI_Scroll::ScrollInfo::FI_Region_XYWH Struct Reference	811
31.103. Detailed Description	812
31.10 FI_Repeat_Button Class Reference	812
31.104. Detailed Description	812
31.104.2. Constructor & Destructor Documentation	813
31.104.2.1 FI_Repeat_Button()	813
31.104.3. Member Function Documentation	813
31.104.3.1 handle()	813
31.10 FI_Return_Button Class Reference	814
31.105. Detailed Description	814
31.105.2. Constructor & Destructor Documentation	815
31.105.2.1 FI_Return_Button()	815
31.105.3. Member Function Documentation	815
31.105.3.1 draw()	815
31.105.3.2 handle()	815
31.10 FI_RGB_Image Class Reference	816
31.106. Detailed Description	817

31.106.2 Constructor & Destructor Documentation	818
31.106.2.1 FI_RGB_Image() [1/2]	818
31.106.2.2 FI_RGB_Image() [2/2]	819
31.106.3 Member Function Documentation	819
31.106.3.1 as_svg_image()	819
31.106.3.2 color_average()	819
31.106.3.3 copy()	819
31.106.3.4 desaturate()	820
31.106.3.5 draw()	820
31.106.3.6 label() [1/2]	821
31.106.3.7 label() [2/2]	821
31.106.3.8 max_size() [1/2]	821
31.106.3.9 max_size() [2/2]	821
31.106.3.10 normalize()	822
31.106.3.11 uncache()	822
31.106.4 Member Data Documentation	822
31.106.4.1 array	822
31.107 FI_Roller Class Reference	823
31.107.1 Detailed Description	823
31.107.2 Constructor & Destructor Documentation	824
31.107.2.1 FI_Roller()	824
31.107.3 Member Function Documentation	824
31.107.3.1 draw()	824
31.107.3.2 handle()	824
31.108 FI_Round_Button Class Reference	825
31.108.1 Detailed Description	826
31.108.2 Constructor & Destructor Documentation	826
31.108.2.1 FI_Round_Button()	826
31.109 FI_Round_Clock Class Reference	827
31.109.1 Detailed Description	827

31.109.2~Constructor & Destructor Documentation	827
31.109.2.1FI_Round_Clock()	828
31.110FI_Scroll Class Reference	828
31.110.1Detailed Description	830
31.110.2~Constructor & Destructor Documentation	831
31.110.2.1FI_Scroll()	831
31.110.3Member Function Documentation	831
31.110.3.1bbox()	831
31.110.3.2clear()	831
31.110.3.3draw()	832
31.110.3.4handle()	832
31.110.3.5recalc_scrollbars()	833
31.110.3.6resize()	833
31.110.3.7scroll_to()	834
31.110.3.8scrollbar_size() [1/2]	834
31.110.3.9scrollbar_size() [2/2]	834
31.110.3.10position()	835
31.110.3.11yposition()	835
31.111FI_Scrollbar Class Reference	835
31.111.1Detailed Description	836
31.111.2~Constructor & Destructor Documentation	837
31.111.2.1FI_Scrollbar()	837
31.111.2.2~FI_Scrollbar()	837
31.111.3Member Function Documentation	837
31.111.3.1draw()	837
31.111.3.2handle()	837
31.111.3.3inesize()	838
31.111.3.4value() [1/3]	838
31.111.3.5value() [2/3]	838
31.111.3.6value() [3/3]	839

31.11 FI_Scroll::Scrollbar_Info::FI_Scrollbar_Data Struct Reference	839
31.112. Detailed Description	840
31.11 FI_Secret_Input Class Reference	840
31.113. Detailed Description	840
31.113.2. Constructor & Destructor Documentation	840
31.113.2.1 FI_Secret_Input()	841
31.113.3. Member Function Documentation	841
31.113.3.1 handle()	841
31.11 FI_Select_Browser Class Reference	842
31.114. Detailed Description	842
31.114.2. Constructor & Destructor Documentation	842
31.114.2.1 FI_Select_Browser()	843
31.11 FI_Shared_Image Class Reference	843
31.115. Detailed Description	845
31.115.2. Constructor & Destructor Documentation	845
31.115.2.1 FI_Shared_Image() [1/2]	845
31.115.2.2 FI_Shared_Image() [2/2]	846
31.115.2.3~ FI_Shared_Image()	846
31.115.3. Member Function Documentation	846
31.115.3.1 add()	846
31.115.3.2 as_shared_image()	846
31.115.3.3 color_average()	847
31.115.3.4 compare()	847
31.115.3.5 copy()	848
31.115.3.6 desaturate()	848
31.115.3.7 draw()	848
31.115.3.8 ind()	849
31.115.3.9 get() [1/2]	849
31.115.3.10 get() [2/2]	850
31.115.3.11 lum_images()	850

31.115.3.10 original()	850
31.115.3.10 fcnt()	851
31.115.3.10 release()	851
31.115.3.10 load()	851
31.115.3.10 move_handler()	851
31.115.3.10 uncache()	852
31.116.1Simple_Counter Class Reference	852
31.116.2Detailed Description	852
31.117.1Simple_Terminal Class Reference	853
31.117.2Detailed Description	854
31.117.2.1Member Function Documentation	856
31.117.2.1ansi() [1/2]	856
31.117.2.2ansi() [2/2]	857
31.117.2.3append()	858
31.117.2.4clear()	858
31.117.2.5current_style_index() [1/2]	858
31.117.2.6current_style_index() [2/2]	859
31.117.2.7draw()	859
31.117.2.8enforce_history_lines()	859
31.117.2.9enforce_stay_at_bottom()	859
31.117.2.10history_lines() [1/2]	859
31.117.2.11history_lines() [2/2]	860
31.117.2.12normal_style_index() [1/2]	860
31.117.2.13normal_style_index() [2/2]	860
31.117.2.14printf()	861
31.117.2.15 move_lines()	861
31.117.2.15stay_at_bottom() [1/2]	861
31.117.2.15stay_at_bottom() [2/2]	862
31.117.2.15style_table() [1/2]	862
31.117.2.15style_table() [2/2]	863

31.117.2.20style_table_size()	864
31.117.2.21text() [1/2]	864
31.117.2.22text() [2/2]	864
31.117.2.23printf()	864
31.118.1Single_Window Class Reference	865
31.118.1.1Detailed Description	865
31.118.1.2Member Function Documentation	866
31.118.2.1show()	866
31.119.1Slider Class Reference	866
31.119.1.1Detailed Description	867
31.119.1.2Constructor & Destructor Documentation	868
31.119.2.1FI_Slider()	868
31.119.3Member Function Documentation	868
31.119.3.1bounds()	868
31.119.3.2draw()	869
31.119.3.3handle()	869
31.119.3.4scrollvalue()	870
31.119.3.5slider() [1/2]	870
31.119.3.6slider() [2/2]	870
31.119.3.7slider_size()	870
31.120.1Spinner Class Reference	871
31.120.1.1Detailed Description	872
31.120.1.2Constructor & Destructor Documentation	873
31.120.2.1FI_Spinner()	873
31.120.3Member Function Documentation	873
31.120.3.1format() [1/2]	873
31.120.3.2format() [2/2]	873
31.120.3.3handle()	874
31.120.3.4maximum() [1/2]	874
31.120.3.5maximum() [2/2]	874

31.120.3.6minimum() [1/2]	875
31.120.3.7minimum() [2/2]	875
31.120.3.8range()	875
31.120.3.9resize()	875
31.120.3.10step() [1/2]	876
31.120.3.11step() [2/2]	876
31.120.3.12textcolor() [1/2]	876
31.120.3.13textcolor() [2/2]	876
31.120.3.14textfont() [1/2]	876
31.120.3.15textfont() [2/2]	877
31.120.3.16textsize() [1/2]	877
31.120.3.17textsize() [2/2]	877
31.120.3.18type() [1/2]	877
31.120.3.19type() [2/2]	877
31.120.3.20value() [1/2]	878
31.120.3.21value() [2/2]	878
31.120.3.22wrap() [1/2]	878
31.120.3.23wrap() [2/2]	879
31.12FI_Spinner::FI_Spinner_Input Class Reference	879
31.121.1Member Function Documentation	879
31.121.1.1handle()	880
31.121.2Surface_Device Class Reference	880
31.122.1Detailed Description	881
31.122.2Constructor & Destructor Documentation	881
31.122.2.1FI_Surface_Device()	882
31.122.2.2~FI_Surface_Device()	882
31.122.3Member Function Documentation	882
31.122.3.1driver() [1/2]	882
31.122.3.2driver() [2/2]	882
31.122.3.3pop_current()	882

31.122.3.4push_current()	883
31.122.3.5set_current()	883
31.122.3.6surface()	883
31.123FI_SVG_File_Surface Class Reference	883
31.123.1Detailed Description	884
31.123.2Constructor & Destructor Documentation	884
31.123.2.1FI_SVG_File_Surface()	885
31.123.2.2~FI_SVG_File_Surface()	886
31.123.3Member Function Documentation	886
31.123.3.1close()	886
31.123.3.2origin()	886
31.123.3.3printable_rect()	887
31.123.3.4translate()	887
31.124FI_SVG_Image Class Reference	888
31.124.1Detailed Description	889
31.124.2Constructor & Destructor Documentation	890
31.124.2.1FI_SVG_Image()	890
31.124.2.2~FI_SVG_Image()	891
31.124.3Member Function Documentation	891
31.124.3.1as_svg_image()	891
31.124.3.2color_average()	891
31.124.3.3copy()	891
31.124.3.4desaturate()	892
31.124.3.5draw()	892
31.124.3.6normalize()	892
31.124.3.7resize()	893
31.124.4Member Data Documentation	893
31.124.4.1proportional	893
31.125FI_Sys_Menu_Bar Class Reference	893
31.125.1Detailed Description	895

31.125.1Member Enumeration Documentation	896
31.125.2.1window_menu_style_enum	896
31.125.3Constructor & Destructor Documentation	897
31.125.3.1FI_Sys_Menu_Bar()	897
31.125.4Member Function Documentation	897
31.125.4.1about()	897
31.125.4.2add() [1/3]	897
31.125.4.3add() [2/3]	899
31.125.4.4add() [3/3]	899
31.125.4.5clear()	900
31.125.4.6clear_submenu()	900
31.125.4.7create_window_menu()	900
31.125.4.8draw()	901
31.125.4.9insert() [1/2]	901
31.125.4.10insert() [2/2]	902
31.125.4.11menu()	902
31.125.4.12node()	902
31.125.4.13remove()	903
31.125.4.14replace()	903
31.125.4.15setonly()	903
31.125.4.16update()	903
31.125.4.17window_menu_style()	904
31.126FI_Table Class Reference	904
31.126.1Detailed Description	910
31.126.2Member Enumeration Documentation	912
31.126.2.1TableContext	912
31.126.3Constructor & Destructor Documentation	912
31.126.3.1FI_Table()	912
31.126.3.2~FI_Table()	913
31.126.4Member Function Documentation	913

31.126.4.1array()	913
31.126.4.2callback()	913
31.126.4.3callback_col()	914
31.126.4.4callback_context()	914
31.126.4.5callback_row()	914
31.126.4.6child()	915
31.126.4.7children()	915
31.126.4.8clear()	915
31.126.4.9col_header()	916
31.126.4.10col_resize()	916
31.126.4.11col_resize_min()	916
31.126.4.12col_width()	916
31.126.4.13col_width_all()	916
31.126.4.14cursor2rowcol()	917
31.126.4.15damage_zone()	917
31.126.4.16o_callback()	917
31.126.4.17draw()	917
31.126.4.18draw_cell()	918
31.126.4.19end_cell()	919
31.126.4.20get_selection()	920
31.126.4.21hit_sizes()	920
31.126.4.22insert()	920
31.126.4.23interactive_resize()	920
31.126.4.24selected()	921
31.126.4.25move_cursor()	921
31.126.4.26calc_dimensions()	921
31.126.4.27redraw_range()	922
31.126.4.28resize()	922
31.126.4.29w_col_clamp()	922
31.126.4.30w_header()	922

31.126.4.3 bw_height()	923
31.126.4.3 bw_height_all()	923
31.126.4.3 bw_resize()	923
31.126.4.3 bw_resize_min()	923
31.126.4.3 scrollbar_size() [1/2]	923
31.126.4.3 scrollbar_size() [2/2]	924
31.126.4.3 set_selection()	924
31.126.4.3 tab_cell_nav() [1/2]	925
31.126.4.3 tab_cell_nav() [2/2]	925
31.126.4.4 table_box()	925
31.126.4.4 table_resized()	925
31.126.4.4 table_scrolled()	926
31.126.4.4 top_row() [1/2]	926
31.126.4.4 top_row() [2/2]	926
31.126.4.4 visible_cells()	926
31.126.4.4 when()	926
31.127FI_Table_Row Class Reference	927
31.127. Detailed Description	928
31.127. Constructor & Destructor Documentation	928
31.127.2.1 FI_Table_Row()	928
31.127.2.2~ FI_Table_Row()	929
31.127.3 Member Function Documentation	929
31.127.3.1 clear()	929
31.127.3.2 row_selected()	929
31.127.3.3 select_all_rows()	929
31.127.3.4 select_row()	930
31.127.3.5 type()	930
31.128FI_Tabs Class Reference	930
31.128. Detailed Description	932
31.128. Constructor & Destructor Documentation	935

31.128.2.1FI_Tabs()	935
31.128.3Member Function Documentation	935
31.128.3.1client_area()	936
31.128.3.2draw()	936
31.128.3.3handle()	937
31.128.3.4push() [1/2]	937
31.128.3.5push() [2/2]	938
31.128.3.6tab_align() [1/2]	938
31.128.3.7tab_align() [2/2]	938
31.128.3.8value() [1/2]	938
31.128.3.9value() [2/2]	939
31.128.3.10high()	939
31.129FI_Text_Buffer Class Reference	939
31.129.Detailed Description	944
31.129.Constructor & Destructor Documentation	945
31.129.2.1FI_Text_Buffer()	945
31.129.3Member Function Documentation	945
31.129.3.1add_modify_callback()	945
31.129.3.2address() [1/2]	945
31.129.3.3address() [2/2]	946
31.129.3.4append()	946
31.129.3.5appendfile()	946
31.129.3.6byte_at()	947
31.129.3.7char_at()	947
31.129.3.8copy()	947
31.129.3.9count_displayed_characters()	948
31.129.3.10count_lines()	948
31.129.3.11findchar_backward()	948
31.129.3.12findchar_forward()	949
31.129.3.13highlight_text()	949

31.129.3.1 <i>4</i> assert()	949
31.129.3.1 <i>5</i> sert_()	950
31.129.3.1 <i>6</i> sertfile()	950
31.129.3.1 <i>7</i> _word_separator()	951
31.129.3.1 <i>8</i> ength()	951
31.129.3.1 <i>9</i> ne_end()	951
31.129.3.2 <i>0</i> ne_start()	951
31.129.3.2 <i>1</i> ne_text()	952
31.129.3.2 <i>2</i> adfile()	952
31.129.3.2 <i>3</i> ext_char()	952
31.129.3.2 <i>4</i> putfile()	953
31.129.3.2 <i>5</i> rev_char()	953
31.129.3.2 <i>6</i> printf()	953
31.129.3.2 <i>7</i> move()	954
31.129.3.2 <i>8</i> move_()	954
31.129.3.2 <i>9</i> place()	955
31.129.3.3 <i>0</i> wind_lines()	956
31.129.3.3 <i>1</i> avefile()	956
31.129.3.3 <i>2</i> earch_backward()	956
31.129.3.3 <i>3</i> earch_forward()	957
31.129.3.3 <i>4</i> secondary_selection_text()	957
31.129.3.3 <i>5</i> selection_text()	958
31.129.3.3 <i>6</i> kip_displayed_characters()	958
31.129.3.3 <i>7</i> ab_distance()	958
31.129.3.3 <i>8</i> xt() [1/2]	958
31.129.3.3 <i>9</i> xt() [2/2]	959
31.129.3.4 <i>0</i> xt_range()	960
31.129.3.4 <i>1</i> printf()	960
31.129.3.4 <i>2</i> ord_end()	961
31.129.3.4 <i>3</i> ord_start()	961

31.129.4Member Data Documentation	961
31.129.4.1file_encoding_warning_message	961
31.129.4.2mPredeleteProcs	962
31.129.4.3nTabDist	962
31.129.4.4transcoding_warning_action	962
31.130FI_Text_Display Class Reference	962
31.130.1Detailed Description	969
31.130.2Member Enumeration Documentation	970
31.130.2.1anonymous enum	970
31.130.2.2anonymous enum	971
31.130.3Constructor & Destructor Documentation	971
31.130.3.1FI_Text_Display()	971
31.130.3.2~FI_Text_Display()	972
31.130.4Member Function Documentation	972
31.130.4.1absolute_top_line_number()	972
31.130.4.2buffer() [1/3]	972
31.130.4.3buffer() [2/3]	973
31.130.4.4buffer() [3/3]	973
31.130.4.5buffer_modified_cb()	973
31.130.4.6buffer_predelete_cb()	974
31.130.4.7calc_last_char()	974
31.130.4.8calc_line_starts()	974
31.130.4.9clear_rect()	976
31.130.4.10l_to_x()	976
31.130.4.11count_lines()	976
31.130.4.12cursor_color() [1/2]	977
31.130.4.13cursor_color() [2/2]	977
31.130.4.14cursor_style()	977
31.130.4.15display_insert()	978
31.130.4.16raw()	978

31.130.4.1 <i>draw_cursor()</i>	978
31.130.4.1 <i>draw_line_numbers()</i>	979
31.130.4.1 <i>draw_range()</i>	979
31.130.4.2 <i>draw_string()</i>	979
31.130.4.2 <i>draw_text()</i>	980
31.130.4.2 <i>draw_vline()</i>	980
31.130.4.2 <i>empty_vlines()</i>	981
31.130.4.2 <i>extend_range_for_styles()</i>	981
31.130.4.2 <i>find_line_end()</i>	981
31.130.4.2 <i>find_wrap_range()</i>	982
31.130.4.2 <i>find_x()</i>	982
31.130.4.2 <i>get_absolute_top_line_number()</i>	983
31.130.4.2 <i>handle_vline()</i>	983
31.130.4.3 <i>highlight_data()</i>	984
31.130.4.3 <i>init_selection()</i>	985
31.130.4.3 <i>assert()</i>	985
31.130.4.3 <i>assert_position() [1/2]</i>	986
31.130.4.3 <i>assert_position() [2/2]</i>	986
31.130.4.3 <i>line_end()</i>	986
31.130.4.3 <i>line_start()</i>	987
31.130.4.3 <i>renumber_align()</i>	987
31.130.4.3 <i>renumber bgcolor()</i>	988
31.130.4.3 <i>renumber fgcolor()</i>	988
31.130.4.4 <i>renumber_font()</i>	988
31.130.4.4 <i>renumber_format()</i>	988
31.130.4.4 <i>renumber_size()</i>	989
31.130.4.4 <i>renumber_width()</i>	989
31.130.4.4 <i>longest_vline()</i>	989
31.130.4.4 <i>maintain_absolute_top_line_number()</i>	989
31.130.4.4 <i>maintaining_absolute_top_line_number()</i>	990

31.130.4.47measure_deleted_lines()	990
31.130.4.48measure_proportional_character()	990
31.130.4.49measure_vline()	991
31.130.4.50move_down()	991
31.130.4.51move_left()	991
31.130.4.52move_right()	992
31.130.4.53move_up()	992
31.130.4.54offset_line_starts()	992
31.130.4.55overstrike()	992
31.130.4.56position_style()	993
31.130.4.57position_to_line()	993
31.130.4.58position_to_linecol()	995
31.130.4.59position_to_xy()	995
31.130.4.60display_range()	996
31.130.4.61reset_absolute_top_line_number()	996
31.130.4.62size()	997
31.130.4.63wind_lines()	997
31.130.4.64scroll()	997
31.130.4.65scroll_()	998
31.130.4.66scroll_timer_cb()	998
31.130.4.67scrollbar_align() [1/2]	998
31.130.4.68scrollbar_align() [2/2]	999
31.130.4.69scrollbar_size() [1/2]	999
31.130.4.70scrollbar_size() [2/2]	999
31.130.4.71scrollbar_width() [1/2]	1000
31.130.4.72scrollbar_width() [2/2]	1000
31.130.4.73shortcut() [1/2]	1000
31.130.4.74shortcut() [2/2]	1000
31.130.4.75show_cursor()	1001
31.130.4.76show_insert_position()	1001

31.130.4.7 skip_lines()	1001
31.130.4.7 string_width()	1002
31.130.4.7 textcolor() [1/2]	1002
31.130.4.8 textcolor() [2/2]	1002
31.130.4.8 textfont() [1/2]	1003
31.130.4.8 textfont() [2/2]	1003
31.130.4.8 textsize() [1/2]	1003
31.130.4.8 textsize() [2/2]	1003
31.130.4.8 update_h_scrollbar()	1004
31.130.4.8 update_line_starts()	1004
31.130.4.8 update_v_scrollbar()	1004
31.130.4.8 line_length()	1005
31.130.4.8 ord_end()	1005
31.130.4.9 ord_start()	1005
31.130.4.9 wrap_mode()	1006
31.130.4.9 wrap_uses_character()	1006
31.130.4.9 wrapped_column()	1007
31.130.4.9 wrapped_line_counter()	1007
31.130.4.9 wrapped_row()	1008
31.130.4.9 y_to_col()	1009
31.130.4.9 y_to_position()	1009
31.130.4.9 y_to_rowcol()	1010
31.13 FI_Text_Editor Class Reference	1010
31.131. Detailed Description	1013
31.131. Member Typedef Documentation	1013
31.131.2. Key_Func	1013
31.131. Constructor & Destructor Documentation	1013
31.131.3. FI_Text_Editor()	1013
31.131. Member Function Documentation	1013
31.131.4. add_default_key_bindings()	1014

31.131.4.2 add_key_binding() [1/2]	1014
31.131.4.3 add_key_binding() [2/2]	1014
31.131.4.4 bound_key_function() [1/2]	1014
31.131.4.5 bound_key_function() [2/2]	1014
31.131.4.6 default_key_function()	1015
31.131.4.7 insert_mode() [1/2]	1015
31.131.4.8 insert_mode() [2/2]	1015
31.131.4.9 f_backspace()	1015
31.131.4.10 c_s_move()	1015
31.131.4.11 f_copy()	1016
31.131.4.12 ctrl_move()	1016
31.131.4.13 cut()	1016
31.131.4.14 default()	1016
31.131.4.15 delete()	1017
31.131.4.16 down()	1017
31.131.4.17 end()	1017
31.131.4.18 enter()	1017
31.131.4.19 home()	1017
31.131.4.20 ignore()	1018
31.131.4.21 f_insert()	1018
31.131.4.22 left()	1018
31.131.4.23 m_s_move()	1018
31.131.4.24 meta_move()	1019
31.131.4.25 move()	1019
31.131.4.26 page_down()	1019
31.131.4.27 page_up()	1019
31.131.4.28 paste()	1020
31.131.4.29 right()	1020
31.131.4.30 select_all()	1020
31.131.4.31 f_shift_move()	1020

31.131.4.3 1 <u>undo()</u>	1020
31.131.4.3 2 <u>up()</u>	1021
31.131.4.3 3 <u>remove_all_key_bindings()</u> [1/2]	1021
31.131.4.3 4 <u>remove_all_key_bindings()</u> [2/2]	1021
31.131.4.3 5 <u>remove_key_binding()</u> [1/2]	1021
31.131.4.3 6 <u>remove_key_binding()</u> [2/2]	1021
31.131.4.3 7 <u>ab_nav()</u> [1/2]	1022
31.131.4.3 8 <u>ab_nav()</u> [2/2]	1023
31.131.5Member Data Documentation	1023
31.131.5.1global_key_bindings	1024
31.132FI_Text_Selection Class Reference	1024
31.132.1Detailed Description	1025
31.132.2Member Function Documentation	1025
31.132.2.1end()	1026
31.132.2.2includes()	1026
31.132.2.3length()	1026
31.132.2.4position()	1026
31.132.2.5selected() [1/2]	1027
31.132.2.6selected() [2/2]	1027
31.132.2.7set()	1028
31.132.2.8start()	1028
31.132.2.9update()	1028
31.132FI_Tile Class Reference	1029
31.133.1Detailed Description	1029
31.133.2Constructor & Destructor Documentation	1031
31.133.2.1FI_Tile()	1031
31.133.3Member Function Documentation	1031
31.133.3.1handle()	1031
31.133.3.2position()	1032
31.133.3.3resize()	1032

31.13 FI_Tiled_Image Class Reference	1033
31.134. Detailed Description	1033
31.134. Constructor & Destructor Documentation	1034
31.134.2.1 FI_Tiled_Image()	1034
31.134. Member Function Documentation	1034
31.134.3.1 color_average()	1034
31.134.3.2 copy()	1035
31.134.3.3 desaturate()	1035
31.134.3.4 draw()	1035
31.13 FI_Timer Class Reference	1036
31.135. Detailed Description	1037
31.135. Constructor & Destructor Documentation	1037
31.135.2.1 FI_Timer()	1037
31.135. Member Function Documentation	1037
31.135.3.1 direction() [1/2]	1037
31.135.3.2 direction() [2/2]	1038
31.135.3.3 draw()	1038
31.135.3.4 handle()	1038
31.135.3.5 suspended() [1/2]	1039
31.135.3.6 suspended() [2/2]	1039
31.13 FI_Toggle_Button Class Reference	1039
31.136. Detailed Description	1040
31.136. Constructor & Destructor Documentation	1040
31.136.2.1 FI_Toggle_Button()	1040
31.13 FI_Tooltip Class Reference	1041
31.137. Detailed Description	1042
31.137. Member Function Documentation	1042
31.137.2.1 color() [1/2]	1043
31.137.2.2 color() [2/2]	1043
31.137.2.3 current()	1043

31.137.2.4delay() [1/2]	1043
31.137.2.5delay() [2/2]	1043
31.137.2.6disable()	1043
31.137.2.7enable()	1044
31.137.2.8enabled()	1044
31.137.2.9enter_area()	1044
31.137.2.10exit() [1/2]	1044
31.137.2.11exit() [2/2]	1044
31.137.2.12dedelay() [1/2]	1045
31.137.2.13dedelay() [2/2]	1045
31.137.2.14overdelay() [1/2]	1045
31.137.2.15overdelay() [2/2]	1045
31.137.2.16margin_height() [1/2]	1045
31.137.2.17margin_height() [2/2]	1045
31.137.2.18margin_width() [1/2]	1046
31.137.2.19margin_width() [2/2]	1046
31.137.2.20size() [1/2]	1046
31.137.2.21size() [2/2]	1046
31.137.2.22extcolor() [1/2]	1046
31.137.2.23extcolor() [2/2]	1046
31.137.2.24wrap_width() [1/2]	1047
31.137.2.25wrap_width() [2/2]	1047
31.138.1_Tree Class Reference	1047
31.138.2.Detailed Description	1054
31.138.2.Member Function Documentation	1058
31.138.2.1add() [1/2]	1059
31.138.2.2add() [2/2]	1059
31.138.2.3calc_dimensions()	1060
31.138.2.4calc_tree()	1060
31.138.2.5callback_item() [1/2]	1061

31.138.2.6callback_item() [2/2]	1061
31.138.2.7callback_reason() [1/2]	1061
31.138.2.8callback_reason() [2/2]	1061
31.138.2.9clear()	1062
31.138.2.10clear_children()	1062
31.138.2.11close() [1/2]	1062
31.138.2.12close() [2/2]	1063
31.138.2.13closeicon() [1/2]	1063
31.138.2.14closeicon() [2/2]	1063
31.138.2.15connectorstyle()	1064
31.138.2.16deselect() [1/2]	1064
31.138.2.17deselect() [2/2]	1064
31.138.2.18deselect_all()	1065
31.138.2.19display()	1066
31.138.2.20displayed()	1066
31.138.2.21extend_selection()	1066
31.138.2.22extend_selection_dir()	1067
31.138.2.23end_clicked()	1068
31.138.2.24end_item()	1068
31.138.2.25first()	1069
31.138.2.26first_selected_item()	1069
31.138.2.27first_visible()	1070
31.138.2.28first_visible_item()	1070
31.138.2.29get_selected_items()	1070
31.138.2.30handle()	1071
31.138.2.31position() [1/2]	1071
31.138.2.32position() [2/2]	1071
31.138.2.33assert()	1072
31.138.2.34assert_above()	1073
31.138.2.35_close() [1/2]	1073

31.138.2.36_close() [2/2]	1074
31.138.2.37_hscroll_visible()	1074
31.138.2.38_open() [1/2]	1075
31.138.2.39_open() [2/2]	1075
31.138.2.40_scrollbar()	1076
31.138.2.41_selected() [1/2]	1076
31.138.2.42_selected() [2/2]	1076
31.138.2.43_vscroll_visible()	1077
31.138.2.44_item_clicked() [1/2]	1077
31.138.2.45_item_clicked() [2/2]	1078
31.138.2.46_item_draw_mode() [1/3]	1078
31.138.2.47_item_draw_mode() [2/3]	1078
31.138.2.48_item_draw_mode() [3/3]	1079
31.138.2.49_item_labelbgcolor() [1/2]	1079
31.138.2.50_item_labelbgcolor() [2/2]	1079
31.138.2.51_item_labelfgcolor()	1079
31.138.2.52_item_labelfont()	1079
31.138.2.53_item_labelsize()	1080
31.138.2.54_item_pathname()	1080
31.138.2.55_item_reselect_mode() [1/2]	1080
31.138.2.56_item_reselect_mode() [2/2]	1081
31.138.2.57_labelmarginleft() [1/2]	1081
31.138.2.58_labelmarginleft() [2/2]	1081
31.138.2.59_last()	1081
31.138.2.60_last_selected_item()	1082
31.138.2.61_last_visible()	1082
31.138.2.62_last_visible_item()	1082
31.138.2.63_ad()	1083
31.138.2.64_ext()	1083
31.138.2.65_ext_item()	1083

31.138.2.66text_selected_item()	1085
31.138.2.67text_visible_item()	1085
31.138.2.68open() [1/2]	1086
31.138.2.69open() [2/2]	1087
31.138.2.70open_toggle()	1088
31.138.2.71openicon() [1/2]	1088
31.138.2.72openicon() [2/2]	1088
31.138.2.73prev()	1089
31.138.2.74calc_tree()	1089
31.138.2.75move()	1089
31.138.2.76size()	1090
31.138.2.77dot()	1090
31.138.2.78dot_label()	1090
31.138.2.79scrollbar_size() [1/2]	1091
31.138.2.80scrollbar_size() [2/2]	1091
31.138.2.81select() [1/2]	1091
31.138.2.82select() [2/2]	1092
31.138.2.83select_all()	1093
31.138.2.84select_only()	1093
31.138.2.85select_toggle()	1094
31.138.2.86selectbox() [1/2]	1094
31.138.2.87selectbox() [2/2]	1095
31.138.2.88selectmode() [1/2]	1095
31.138.2.89selectmode() [2/2]	1095
31.138.2.90set_item_focus()	1095
31.138.2.91show_item() [1/2]	1095
31.138.2.92show_item() [2/2]	1096
31.138.2.93show_item_bottom()	1096
31.138.2.94show_item_middle()	1097
31.138.2.95show_item_top()	1097

31.138.2.9 show_self()	1097
31.138.2.9 showcollapse() [1/2]	1097
31.138.2.9 showcollapse() [2/2]	1098
31.138.2.9 showroot()	1098
31.138.2.10 torder()	1098
31.138.2.10 tericon() [1/2]	1098
31.138.2.10 tericon() [2/2]	1098
31.138.2.10 tericonmarginleft() [1/2]	1099
31.138.2.10 tericonmarginleft() [2/2]	1099
31.138.2.10 position() [1/2]	1099
31.138.2.10 position() [2/2]	1099
31.138.2.10 Widgetmarginleft() [1/2]	1100
31.138.2.10 Widgetmarginleft() [2/2]	1100
31.138 FI_Tree_Item Class Reference	1100
31.139.1 Detailed Description	1105
31.139.2 Constructor & Destructor Documentation	1106
31.139.2.1 FI_Tree_Item() [1/2]	1106
31.139.2.2 FI_Tree_Item() [2/2]	1106
31.139.3 Member Function Documentation	1107
31.139.3.1 activate()	1107
31.139.3.2 add() [1/4]	1107
31.139.3.3 add() [2/4]	1107
31.139.3.4 add() [3/4]	1108
31.139.3.5 add() [4/4]	1108
31.139.3.6 calc_item_height()	1108
31.139.3.7 child()	1109
31.139.3.8 deactivate()	1109
31.139.3.9 parent()	1109
31.139.3.10 depth()	1109
31.139.3.11 deselect_all()	1109

31.139.3.10 <i>draw()</i>	1110
31.139.3.10 <i>draw_horizontal_connector()</i>	1110
31.139.3.10 <i>draw_item_content()</i>	1111
31.139.3.10 <i>draw_vertical_connector()</i>	1112
31.139.3.10 <i>drawbgcolor()</i>	1112
31.139.3.10 <i>drawfgcolor()</i>	1113
31.139.3.10 <i>end_child() [1/2]</i>	1113
31.139.3.10 <i>end_child() [2/2]</i>	1113
31.139.3.20 <i>end_child_item() [1/2]</i>	1114
31.139.3.21 <i>end_child_item() [2/2]</i>	1114
31.139.3.22 <i>end_clicked()</i>	1114
31.139.3.23 <i>end_item()</i>	1115
31.139.3.24 <i>end_widgets()</i>	1115
31.139.3.25 <i>insert()</i>	1115
31.139.3.26 <i>insert_above()</i>	1116
31.139.3.27 <i>label()</i>	1116
31.139.3.28 <i>label_h()</i>	1116
31.139.3.29 <i>label_w()</i>	1116
31.139.3.30 <i>label_x()</i>	1117
31.139.3.31 <i>label_y()</i>	1117
31.139.3.32 <i>labelbgcolor() [1/2]</i>	1117
31.139.3.33 <i>labelbgcolor() [2/2]</i>	1117
31.139.3.34 <i>move() [1/2]</i>	1118
31.139.3.35 <i>move() [2/2]</i>	1118
31.139.3.36 <i>move_above()</i>	1119
31.139.3.37 <i>move_below()</i>	1119
31.139.3.38 <i>move_into()</i>	1119
31.139.3.39 <i>next()</i>	1120
31.139.3.40 <i>next_displayed()</i>	1120
31.139.3.41 <i>next_sibling()</i>	1120

31.139.3.42ext_visible()	1120
31.139.3.43parent()	1121
31.139.3.44refs()	1121
31.139.3.45rev()	1121
31.139.3.46rev_displayed()	1121
31.139.3.47rev_sibling()	1122
31.139.3.48rev_visible()	1122
31.139.3.49calc_tree()	1122
31.139.3.50move_child() [1/2]	1123
31.139.3.51move_child() [2/2]	1123
31.139.3.52parent()	1123
31.139.3.53place()	1124
31.139.3.54place_child()	1124
31.139.3.55select()	1125
31.139.3.56select_all()	1125
31.139.3.57show_self()	1125
31.139.3.58show_widgets()	1125
31.139.3.59swap_children() [1/2]	1125
31.139.3.60swap_children() [2/2]	1126
31.139.3.61tree() [1/2]	1126
31.139.3.62tree() [2/2]	1126
31.139.3.63update_prev_next()	1127
31.139.3.64serdeicon() [1/2]	1127
31.139.3.65serdeicon() [2/2]	1127
31.139.3.66sericon()	1128
31.139.3.67visible_r()	1128
31.139.3.68()	1128
31.146I_Tree_Item_Array Class Reference	1128
31.140.Detailed Description	1129
31.140.Constructor & Destructor Documentation	1129

31.140.2.1FI_Tree_Item_Array()	1130
31.140.3Member Function Documentation	1130
31.140.3.1add()	1130
31.140.3.2clear()	1130
31.140.3.3parent()	1130
31.140.3.4insert()	1131
31.140.3.5manage_item_destroy()	1131
31.140.3.6move()	1131
31.140.3.7remove() [1/2]	1131
31.140.3.8remove() [2/2]	1132
31.140.3.9reparent()	1132
31.140.3.10place()	1132
31.14FI_Tree_Prefs Class Reference	1132
31.141.Detailed Description	1135
31.141.1Member Function Documentation	1135
31.141.2.1closedeicon()	1136
31.141.2.2closeicon()	1136
31.141.2.3item_draw_mode()	1136
31.141.2.4item_labelbgcolor() [1/2]	1136
31.141.2.5item_labelbgcolor() [2/2]	1136
31.141.2.6marginbottom() [1/2]	1137
31.141.2.7marginbottom() [2/2]	1137
31.141.2.8pendeicon()	1137
31.141.2.9openicon() [1/2]	1137
31.141.2.10openicon() [2/2]	1137
31.141.2.11selectmode()	1138
31.141.2.12showcollapse()	1138
31.141.2.13showroot()	1138
31.141.2.14sortorder()	1138
31.141.2.15userdeicon()	1139

31.14 FI_Valuator Class Reference	1139
31.142. Detailed Description	1141
31.142. Constructor & Destructor Documentation	1141
31.142.2.1 FI_Valuator()	1141
31.142.3. Member Function Documentation	1142
31.142.3.1 bounds()	1142
31.142.3.2 clamp()	1142
31.142.3.3 format()	1142
31.142.3.4 handle_drag()	1142
31.142.3.5 handle_release()	1143
31.142.3.6 increment()	1143
31.142.3.7 maximum() [1/2]	1143
31.142.3.8 maximum() [2/2]	1143
31.142.3.9 minimum() [1/2]	1143
31.142.3.10 minimum() [2/2]	1143
31.142.3.11 precision()	1144
31.142.3.12 range()	1144
31.142.3.13 round()	1144
31.142.3.14 set_value()	1144
31.142.3.15 step()	1145
31.142.3.16 value() [1/2]	1145
31.142.3.17 value() [2/2]	1145
31.14 FI_Value_Input Class Reference	1145
31.143. Detailed Description	1147
31.143. Constructor & Destructor Documentation	1147
31.143.2.1 FI_Value_Input()	1147
31.143.3. Member Function Documentation	1147
31.143.3.1 cursor_color() [1/2]	1148
31.143.3.2 cursor_color() [2/2]	1148
31.143.3.3 draw()	1148

31.143.3.4handle()	1148
31.143.3.5resize()	1149
31.143.3.6shortcut() [1/2]	1150
31.143.3.7shortcut() [2/2]	1150
31.143.3.8soft()	1150
31.143.3.9textcolor() [1/2]	1150
31.143.3.10textcolor() [2/2]	1150
31.143.3.11textfont() [1/2]	1151
31.143.3.12textfont() [2/2]	1151
31.143.3.13textsize() [1/2]	1151
31.143.3.14textsize() [2/2]	1151
31.144FI_Value_Output Class Reference	1151
31.144.1Detailed Description	1152
31.144.2Constructor & Destructor Documentation	1152
31.144.2.1FI_Value_Output()	1153
31.144.3Member Function Documentation	1153
31.144.3.1draw()	1153
31.144.3.2handle()	1153
31.144.3.3soft() [1/2]	1154
31.144.3.4soft() [2/2]	1154
31.144.3.5textcolor() [1/2]	1154
31.144.3.6textcolor() [2/2]	1154
31.144.3.7textfont() [1/2]	1155
31.144.3.8textfont() [2/2]	1155
31.144.3.9textsize()	1155
31.145FI_Value_Slider Class Reference	1155
31.145.1Detailed Description	1156
31.145.2Constructor & Destructor Documentation	1156
31.145.2.1FI_Value_Slider()	1157
31.145.3Member Function Documentation	1157

31.145.3.1draw()	1157
31.145.3.2handle()	1157
31.145.3.3textcolor() [1/2]	1158
31.145.3.4textcolor() [2/2]	1158
31.145.3.5textfont() [1/2]	1158
31.145.3.6textfont() [2/2]	1158
31.145.3.7textsize() [1/2]	1159
31.145.3.8textsize() [2/2]	1159
31.146 FI_Widget Class Reference	1159
31.146.1 Detailed Description	1165
31.146.2 Member Enumeration Documentation	1165
31.146.2.1anonymous enum	1165
31.146.3 Constructor & Destructor Documentation	1166
31.146.3.1FI_Widget()	1166
31.146.3.2~FI_Widget()	1167
31.146.4 Member Function Documentation	1167
31.146.4.1activate()	1167
31.146.4.2active()	1167
31.146.4.3active_r()	1168
31.146.4.4align() [1/2]	1168
31.146.4.5align() [2/2]	1168
31.146.4.6argument() [1/2]	1169
31.146.4.7argument() [2/2]	1169
31.146.4.8as_gl_window()	1169
31.146.4.9as_group()	1170
31.146.4.10s_window()	1171
31.146.4.11box() [1/2]	1171
31.146.4.12box() [2/2]	1171
31.146.4.13callback() [1/5]	1172
31.146.4.14callback() [2/5]	1172

31.146.4.15callback() [3/5]	1172
31.146.4.16callback() [4/5]	1173
31.146.4.17callback() [5/5]	1173
31.146.4.18changed()	1173
31.146.4.19clear_active()	1174
31.146.4.20clear_changed()	1174
31.146.4.21clear_damage()	1174
31.146.4.22clear_output()	1175
31.146.4.23clear_visible()	1175
31.146.4.24clear_visible_focus()	1175
31.146.4.25color() [1/3]	1176
31.146.4.26color() [2/3]	1176
31.146.4.27color() [3/3]	1176
31.146.4.28color2() [1/2]	1177
31.146.4.29color2() [2/2]	1177
31.146.4.30contains()	1177
31.146.4.31copy_label()	1178
31.146.4.32copy_tooltip()	1178
31.146.4.33damage() [1/3]	1179
31.146.4.34damage() [2/3]	1179
31.146.4.35damage() [3/3]	1179
31.146.4.36damage_resize()	1180
31.146.4.37deactivate()	1180
31.146.4.38default_callback()	1180
31.146.4.39reimage() [1/4]	1181
31.146.4.40reimage() [2/4]	1181
31.146.4.41reimage() [3/4]	1181
31.146.4.42reimage() [4/4]	1182
31.146.4.43b_callback() [1/3]	1182
31.146.4.44b_callback() [2/3]	1182

31.146.4.45 <code>to_callback()</code> [3/3]	1183
31.146.4.46 <code>draw()</code>	1183
31.146.4.47 <code>draw_box()</code> [1/2]	1184
31.146.4.48 <code>draw_box()</code> [2/2]	1184
31.146.4.49 <code>draw_focus()</code>	1184
31.146.4.50 <code>draw_label()</code> [1/3]	1185
31.146.4.51 <code>draw_label()</code> [2/3]	1185
31.146.4.52 <code>draw_label()</code> [3/3]	1185
31.146.4.53 <code>()</code> [1/2]	1185
31.146.4.54 <code>()</code> [2/2]	1185
31.146.4.55 <code>handle()</code>	1186
31.146.4.56 <code>nde()</code>	1186
31.146.4.57 <code>image()</code> [1/4]	1187
31.146.4.58 <code>image()</code> [2/4]	1187
31.146.4.59 <code>image()</code> [3/4]	1187
31.146.4.60 <code>image()</code> [4/4]	1187
31.146.4.61 <code>inside()</code>	1188
31.146.4.62 <code>_label_copied()</code>	1188
31.146.4.63 <code>label()</code> [1/3]	1188
31.146.4.64 <code>label()</code> [2/3]	1189
31.146.4.65 <code>label()</code> [3/3]	1189
31.146.4.66 <code>label_shortcut()</code>	1189
31.146.4.67 <code>labelcolor()</code> [1/2]	1190
31.146.4.68 <code>labelcolor()</code> [2/2]	1190
31.146.4.69 <code>labelfont()</code> [1/2]	1190
31.146.4.70 <code>labelfont()</code> [2/2]	1191
31.146.4.71 <code>labelsize()</code> [1/2]	1191
31.146.4.72 <code>labelsize()</code> [2/2]	1191
31.146.4.73 <code>labeltype()</code> [1/2]	1192
31.146.4.74 <code>labeltype()</code> [2/2]	1192

31.146.4.75measure_label()	1193
31.146.4.76output()	1193
31.146.4.77parent() [1/2]	1193
31.146.4.78parent() [2/2]	1194
31.146.4.79position()	1194
31.146.4.80draw()	1194
31.146.4.81redraw_label()	1194
31.146.4.82size()	1195
31.146.4.83selection_color() [1/2]	1195
31.146.4.84selection_color() [2/2]	1195
31.146.4.85set_active()	1196
31.146.4.86set_changed()	1196
31.146.4.87set_output()	1196
31.146.4.88set_visible()	1197
31.146.4.89set_visible_focus()	1197
31.146.4.90shortcut_label() [1/2]	1197
31.146.4.91shortcut_label() [2/2]	1197
31.146.4.92show()	1198
31.146.4.93size()	1198
31.146.4.94take_focus()	1198
31.146.4.95akesevents()	1199
31.146.4.96test_shortcut() [1/2]	1199
31.146.4.97test_shortcut() [2/2]	1199
31.146.4.98oltip() [1/2]	1200
31.146.4.99oltip() [2/2]	1200
31.146.4.100_window()	1201
31.146.4.101_window_offset()	1201
31.146.4.102e() [1/2]	1201
31.146.4.103e() [2/2]	1202
31.146.4.104er_data() [1/2]	1202

31.146.4.10 er_data() [2/2]	1202
31.146.4.10 sible()	1202
31.146.4.10 sible_focus() [1/2]	1203
31.146.4.10 sible_focus() [2/2]	1203
31.146.4.10 sible_r()	1203
31.146.4.11 0 [1/2]	1204
31.146.4.11 l() [2/2]	1204
31.146.4.11 len() [1/2]	1204
31.146.4.11 len() [2/2]	1205
31.146.4.11 Window()	1205
31.146.4.11 x(5 [1/2]	1206
31.146.4.11 x(6 [2/2]	1206
31.146.4.11 y(7 [1/2]	1206
31.146.4.11 y(8 [2/2]	1206
31.14 F _Widget_Surface Class Reference	1207
31.147.1Detailed Description	1208
31.147.2Constructor & Destructor Documentation	1208
31.147.2.1 Widget_Surface()	1208
31.147.3Member Function Documentation	1208
31.147.3.1 draw()	1208
31.147.3.2 draw_decorated_window()	1209
31.147.3.3 origin() [1/2]	1209
31.147.3.4 origin() [2/2]	1209
31.147.3.5 print_window_part()	1210
31.147.3.6 printable_rect()	1210
31.147.3.7 translate()	1211
31.14 F _Widget_Tracker Class Reference	1211
31.148.1Detailed Description	1212
31.148.2Member Function Documentation	1212
31.148.2.1 deleted()	1212

31.148.2. <code>exists()</code>	1213
31.148.3. <code>widget()</code>	1213
31.149. <code>FI_Window</code> Class Reference	1213
31.149.1.Detailed Description	1217
31.149.2.Constructor & Destructor Documentation	1217
31.149.2.1. <code>FI_Window()</code> [1/2]	1218
31.149.2.2. <code>FI_Window()</code> [2/2]	1218
31.149.2.3. <code>~FI_Window()</code>	1218
31.149.3.Member Function Documentation	1219
31.149.3.1. <code>as_window()</code>	1219
31.149.3.2. <code>border()</code>	1219
31.149.3.3. <code>clear_border()</code>	1219
31.149.3.4. <code>clear_modal_states()</code>	1220
31.149.3.5. <code>current()</code>	1220
31.149.3.6. <code>cursor()</code> [1/3]	1221
31.149.3.7. <code>cursor()</code> [2/3]	1221
31.149.3.8. <code>cursor()</code> [3/3]	1221
31.149.3.9. <code>decorated_h()</code>	1222
31.149.3.10. <code>decorated_w()</code>	1222
31.149.3.11. <code>default_cursor()</code> [1/2]	1222
31.149.3.12. <code>default_cursor()</code> [2/2]	1222
31.149.3.13. <code>default_icon()</code>	1222
31.149.3.14. <code>default_icons()</code>	1223
31.149.3.15. <code>default_xclass()</code> [1/2]	1223
31.149.3.16. <code>default_xclass()</code> [2/2]	1224
31.149.3.17. <code>draw()</code>	1224
31.149.3.18. <code>push()</code>	1225
31.149.3.19. <code>force_position()</code> [1/2]	1225
31.149.3.20. <code>force_position()</code> [2/2]	1225
31.149.3.21. <code>free_icons()</code>	1225

31.149.3.21 <code>dee_position()</code>	1226
31.149.3.22 <code>fullscreen()</code>	1226
31.149.3.23 <code>fullscreen_screens()</code>	1226
31.149.3.24 <code>handle()</code>	1227
31.149.3.25 <code>nde()</code>	1228
31.149.3.26 <code>hotspot()</code>	1228
31.149.3.27 <code>on()</code> [1/3]	1229
31.149.3.28 <code>on()</code> [2/3]	1230
31.149.3.29 <code>on()</code> [3/3]	1230
31.149.3.30 <code>tonize()</code>	1230
31.149.3.31 <code>onlabel()</code>	1231
31.149.3.32 <code>ons()</code>	1231
31.149.3.33 <code>abel()</code> [1/2]	1231
31.149.3.34 <code>abel()</code> [2/2]	1231
31.149.3.35 <code>ake_current()</code>	1232
31.149.3.36 <code>menu_window()</code>	1232
31.149.3.37 <code>odal()</code>	1232
31.149.3.38 <code>on_modal()</code>	1232
31.149.3.39 <code>override()</code>	1232
31.149.3.40 <code>esize()</code>	1233
31.149.3.41 <code>screen_num()</code>	1233
31.149.3.42 <code>set_menu_window()</code>	1233
31.149.3.43 <code>et_modal()</code>	1234
31.149.3.44 <code>et_non_modal()</code>	1234
31.149.3.45 <code>et_tooltip_window()</code>	1234
31.149.3.46 <code>shape()</code> [1/2]	1235
31.149.3.47 <code>shape()</code> [2/2]	1236
31.149.3.48 <code>show()</code> [1/2]	1236
31.149.3.49 <code>show()</code> [2/2]	1236
31.149.3.50 <code>shown()</code>	1237

31.149.3.5 size _range()	1237
31.149.3.5 oltip _window()	1238
31.149.3.5 wait_for_expose ()	1238
31.149.3.5 class () [1/2]	1239
31.149.3.5 class () [2/2]	1239
31.149.4Member Data Documentation	1239
31.149.4.1current_	1240
31.15FI_Wizard Class Reference	1240
31.150.Detailed Description	1241
31.150.CConstructor & Destructor Documentation	1241
31.150.2.1FI_Wizard()	1241
31.150.3Member Function Documentation	1241
31.150.3.1draw()	1241
31.150.3.2next()	1242
31.150.3.3prev()	1242
31.150.3.4value() [1/2]	1242
31.150.3.5value() [2/2]	1242
31.15FI_XBM_Image Class Reference	1242
31.151.Detailed Description	1243
31.151.CConstructor & Destructor Documentation	1243
31.151.2.1FI_XBM_Image()	1243
31.15FI_XColor Struct Reference	1243
31.15FI_XPM_Image Class Reference	1244
31.153.Detailed Description	1244
31.153.CConstructor & Destructor Documentation	1244
31.153.2.1FI_XPM_Image()	1244
31.15FI_Text_Editor::Key_Binding Struct Reference	1245
31.154.Detailed Description	1245
31.15FI_Preferences::Name Class Reference	1245
31.155.Detailed Description	1246
31.155.CConstructor & Destructor Documentation	1246
31.155.2.1Name() [1/2]	1246
31.155.2.2Name() [2/2]	1246
31.15FI_Preferences::Node Class Reference	1247
31.15FI_Paged_Device::page_format Struct Reference	1247
31.157.Detailed Description	1248
31.15FI_Preferences::RootNode Class Reference	1248
31.15FI_Scroll::ScrollInfo Struct Reference	1248
31.159.Detailed Description	1249
31.16FI_Text_Display::Style_Table_Entry Struct Reference	1249
31.160.Detailed Description	1250

32 File Documentation	1251
32.1 Enumerations.H File Reference	1251
32.1.1 Detailed Description	1263
32.1.2 Macro Definition Documentation	1263
32.1.2.1 FL_ABI_VERSION	1263
32.1.2.2 FL_API_VERSION	1263
32.1.2.3 FL_IMAGE_LABEL	1264
32.1.2.4 FL_MAJOR_VERSION	1264
32.1.2.5 FL_MINOR_VERSION	1264
32.1.2.6 FL_MULTI_LABEL	1264
32.1.2.7 FL_PATCH_VERSION	1264
32.1.2.8 FL_SYMBOL_LABEL	1265
32.1.2.9 FL_VERSION	1265
32.1.3 Typedef Documentation	1265
32.1.3.1 Fl_Align	1265
32.1.3.2 Fl_Font	1266
32.1.3.3 Fl_Fontsize	1266
32.1.4 Enumeration Type Documentation	1266
32.1.4.1 anonymous enum	1266
32.1.4.2 Fl_Boxtype	1266
32.1.4.3 Fl_Cursor	1268
32.1.4.4 Fl_Damage	1269
32.1.4.5 Fl_Event	1269
32.1.4.6 Fl_Labeltype	1272
32.1.4.7 Fl_When	1273
32.1.5 Function Documentation	1274
32.1.5.1 fl_box()	1274
32.1.5.2 fl_color_cube()	1274
32.1.5.3 fl_darker()	1274
32.1.5.4 fl_define_FL_EMBOSSLED_LABEL()	1274

32.1.5.5 fl_define_FL_ENGRAVED_LABEL()	1275
32.1.5.6 fl_define_FL_ICON_LABEL()	1275
32.1.5.7 fl_define_FL_IMAGE_LABEL()	1275
32.1.5.8 fl_define_FL_MULTI_LABEL()	1275
32.1.5.9 fl_define_FL_SHADOW_LABEL()	1275
32.1.5.10 fl_down()	1275
32.1.5.11 fl_frame()	1276
32.1.5.12 fl_gray_ramp()	1276
32.1.5.13 fl_lighter()	1276
32.1.5.14 fl_rgb_color() [1/2]	1276
32.1.5.15 fl_rgb_color() [2/2]	1276
32.1.6 Variable Documentation	1277
32.1.6.1 FL_ALIGN_BOTTOM	1277
32.1.6.2 FL_ALIGN_CENTER	1277
32.1.6.3 FL_ALIGN_CLIP	1277
32.1.6.4 FL_ALIGN_IMAGE_BACKDROP	1277
32.1.6.5 FL_ALIGN_IMAGE_MASK	1277
32.1.6.6 FL_ALIGN_IMAGE_NEXT_TO_TEXT	1278
32.1.6.7 FL_ALIGN_IMAGE_OVER_TEXT	1278
32.1.6.8 FL_ALIGN_INSIDE	1278
32.1.6.9 FL_ALIGN_LEFT	1278
32.1.6.10 FL_ALIGN_LEFT_BOTTOM	1278
32.1.6.11 FL_ALIGN_LEFT_TOP	1278
32.1.6.12 FL_ALIGN_NOWRAP	1279
32.1.6.13 FL_ALIGN_POSITION_MASK	1279
32.1.6.14 FL_ALIGN_RIGHT	1279
32.1.6.15 FL_ALIGN_RIGHT_BOTTOM	1279
32.1.6.16 FL_ALIGN_RIGHT_TOP	1279
32.1.6.17 FL_ALIGN_TEXT_NEXT_TO_IMAGE	1279
32.1.6.18 FL_ALIGN_TEXT_OVER_IMAGE	1280

32.1.6.19 FL_ALIGN_TOP	1280
32.1.6.20 FL_ALIGN_WRAP	1280
32.1.6.21 FL_NORMAL_SIZE	1280
32.2 filename.H File Reference	1280
32.2.1 Detailed Description	1281
32.3 Fl.hxx File Reference	1281
32.3.1 Detailed Description	1282
32.3.2 Function Documentation	1282
32.3.2.1 fl_open_display()	1282
32.4 Fl.H File Reference	1283
32.4.1 Detailed Description	1284
32.5 fl_arc.hxx File Reference	1284
32.5.1 Detailed Description	1284
32.6 fl_ask.hxx File Reference	1284
32.6.1 Detailed Description	1286
32.7 fl_ask.H File Reference	1286
32.7.1 Detailed Description	1287
32.7.2 Enumeration Type Documentation	1287
32.7.2.1 FL_Beep	1287
32.7.3 Function Documentation	1288
32.7.3.1 fl_message_position()	1288
32.8 fl_boxtype.hxx File Reference	1288
32.8.1 Detailed Description	1289
32.8.2 Function Documentation	1290
32.8.2.1 fl_internal_boxtype()	1290
32.8.2.2 fl_rectbound()	1290
32.9 fl_color.hxx File Reference	1290
32.9.1 Detailed Description	1291
32.10 Fl_Color_Chooser.H File Reference	1291
32.10.1 Detailed Description	1291

32.11 Fl_compose.cxx File Reference	1291
32.11.1 Detailed Description	1291
32.12 fl_curve.cxx File Reference	1292
32.12.1 Detailed Description	1292
32.13 Fl_Device.H File Reference	1292
32.13.1 Detailed Description	1292
32.14 Fl_Double_Window.cxx File Reference	1292
32.14.1 Detailed Description	1293
32.15 fl_draw.H File Reference	1293
32.15.1 Detailed Description	1298
32.16 Fl_Image.H File Reference	1298
32.16.1 Detailed Description	1299
32.16.2 Enumeration Type Documentation	1299
32.16.2.1 Fl_RGB_Scaling	1299
32.17 Fl_Menu_Item.H File Reference	1299
32.17.1 Enumeration Type Documentation	1300
32.17.1.1 anonymous enum	1300
32.18 Fl_Native_File_Chooser.H File Reference	1300
32.18.1 Detailed Description	1300
32.19 Fl_Paged_Device.cxx File Reference	1301
32.19.1 Detailed Description	1301
32.20 Fl_Paged_Device.H File Reference	1301
32.20.1 Detailed Description	1301
32.21 Fl_PostScript.H File Reference	1301
32.21.1 Detailed Description	1302
32.22 Fl_Printer.H File Reference	1302
32.22.1 Detailed Description	1302
32.23 fl_rect.cxx File Reference	1302
32.23.1 Detailed Description	1302
32.24 Fl_Shared_Image.H File Reference	1303

32.24.1 Detailed Description	1303
32.24.2 Function Documentation	1303
32.24.2.1 fl_register_images()	1303
32.25fl_show_colormap.H File Reference	1303
32.25.1 Detailed Description	1304
32.26fl_string.h File Reference	1304
32.26.1 Detailed Description	1304
32.27FI_Sys_Menu_Bar.H File Reference	1304
32.27.1 Detailed Description	1305
32.28FI_Tree.H File Reference	1305
32.28.1 Detailed Description	1305
32.28.2 Enumeration Type Documentation	1305
32.28.2.1 FI_Tree_Reason	1305
32.29FI_Tree_Item.H File Reference	1306
32.29.1 Detailed Description	1306
32.30FI_Tree_Item_Array.H File Reference	1306
32.30.1 Detailed Description	1307
32.31FI_Tree_Prefs.H File Reference	1307
32.31.1 Detailed Description	1308
32.31.2 Enumeration Type Documentation	1308
32.31.2.1 FI_Tree_Connector	1308
32.31.2.2 FI_Tree_Item_Draw_Mode	1308
32.31.2.3 FI_Tree_Item_Reselect_Mode	1308
32.31.2.4 FI_Tree_Select	1309
32.31.2.5 FI_Tree_Sort	1309
32.32fl_types.h File Reference	1309
32.32.1 Detailed Description	1310
32.32.2 Typedef Documentation	1310
32.32.2.1 FI_Shortcut	1310
32.33fl_utf8.h File Reference	1310

32.33.1 Detailed Description	1312
32.34fl_vertex.cxx File Reference	1313
32.34.1 Detailed Description	1313
32.35Fl_Widget.H File Reference	1313
32.35.1 Detailed Description	1314
32.35.2 Macro Definition Documentation	1314
32.35.2.1 FL_RESERVED_TYPE	1314
32.36Fl_Window.H File Reference	1314
32.36.1 Detailed Description	1314
32.37gl.h File Reference	1315
32.37.1 Detailed Description	1316
32.37.2 Function Documentation	1316
32.37.2.1 gl_color()	1316
32.37.2.2 gl_draw() [1/7]	1316
32.37.2.3 gl_draw() [2/7]	1316
32.37.2.4 gl_draw() [3/7]	1317
32.37.2.5 gl_draw() [4/7]	1317
32.37.2.6 gl_draw() [5/7]	1317
32.37.2.7 gl_draw() [6/7]	1318
32.37.2.8 gl_draw() [7/7]	1318
32.37.2.9 gl_font()	1318
32.37.2.10gl_rect()	1318
32.37.2.11gl_rectf()	1319
32.37.2.12gl_texture_pile_height() [1/2]	1319
32.37.2.13gl_texture_pile_height() [2/2]	1319
32.38mac.H File Reference	1319
32.38.1 Detailed Description	1320
32.39numericsort.c File Reference	1320
32.39.1 Function Documentation	1320
32.39.1.1 fl_casenumericssort()	1321
32.39.1.2 fl_numericsort()	1321
32.40platform_types.h File Reference	1322
32.40.1 Detailed Description	1323
32.40.2 Typedef Documentation	1323
32.40.2.1 fl_intptr_t	1323
32.40.2.2 fl_uintptr_t	1323
Index	1325

Chapter 1

FLTK Programming Manual



FLTK 1.4.0 Programming Manual

Revision 11 by F. Costantini, D. Gibson, M. Melcher, A. Schlosser,
B. Spitzak and M. Sweet.

Copyright 1998-2020 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.

<p>Preface Introduction to FLTK FLTK Basics Common Widgets and Attributes</p> <ul style="list-style-type: none">• Colors• Box Types• Labels and Label Types• Drawing Images <p>How does resizing work? Designing a Simple Text Editor Drawing Things in FLTK Handling Events</p> <ul style="list-style-type: none">• Fl::event_*() methods• Event Propagation <p>Adding and Extending Widgets Using OpenGL Programming with FLUID</p> <ul style="list-style-type: none">• GUI Attributes• Selecting and Moving Widgets• Image Labels	<p>Advanced FLTK Unicode and UTF-8 Support</p> <p>Appendices:</p> <ul style="list-style-type: none">• FLTK Enumerations• GLUT Compatibility<ul style="list-style-type: none">– class Fl_Glut_Window• Forms Compatibility• Operating System Issues• Migrating Code from FLTK 1.3 to 1.4• Developer Information• Software License• Example Source Code• FAQ (Frequently Asked Questions)
--	---

Chapter 2

Preface

This manual describes the Fast Light Tool Kit ("FLTK") version 1.4.0, a C++ Graphical User Interface ("GUI") toolkit for UNIX, Microsoft Windows and Apple OS X.

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](#).

2.1 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](#)
- [Programming with FLUID](#)
- [Advanced FLTK](#)
- [Unicode and UTF-8 Support](#)
- [FLTK Enumerations](#)
- [GLUT Compatibility](#)
- [Forms Compatibility](#)
- [Operating System Issues](#)
- [Migrating Code from FLTK 1.3 to 1.4](#)
- [Developer Information](#)
- [Software License](#)
- [Example Source Code](#)

2.2 Conventions

This manual was generated using Doxygen (see <http://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 <http://www.stack.nl/~dimitri/doxygen/>

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

2.3 Abbreviations

The following abbreviations are used in this manual:

X11

The X Window System version 11.

Xlib

The X Window System interface library.

MS 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 MS Windows API.

OS X, __APPLE__

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

2.4 Copyrights and Trademarks

FLTK is Copyright 1998-2020 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, and Mac OS X are registered trademarks of Apple Computer, Inc.

Chapter 3

Introduction to FLTK

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

FLTK provides modern GUI functionality without the 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 in the US.

3.1 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. That is the version you have now.

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.

3.2 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 now included with several Linux distributions.

Here are some of the core features unique to FLTK:

- `sizeof(Fl_Widget) == 64 to 92.`
- The "core" (the "hello" program compiled & linked with a static FLTK library using `gcc` on a 486 and then stripped) is 114K.
- The FLUID program (which includes every widget) is 538k.
- Written directly atop core libraries (Xlib, Windows or Cocoa) for maximum speed, and carefully optimized for code size and performance.
- Precise low-level compatibility between the X11, Windows and MacOS versions - only about 10% of the code is different.
- Interactive user interface builder program. Output is human-readable and editable C++ source code.
- Support for overlay hardware, with emulation if none is available.
- Very small & fast portable 2-D drawing library to hide Xlib, Windows, or QuickDraw.
- 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.

3.3 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!

3.4 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".

3.5 Building and Installing FLTK Under UNIX and Apple macOS

In most cases you can just type "make". This will run configure with the default of no options and then compile everything.

FLTK uses GNU autoconf to configure itself for your UNIX platform. The main things that the configure script will look for are the X11 and OpenGL (or Mesa) header and library files. If these cannot be found in the standard include/library locations you'll need to define the `CFLAGS`, `CXXFLAGS`, and `LDFLAGS` environment variables. For the Bourne and Korn shells you'd use:

```
CFLAGS=-Iincludedir; export CFLAGS  
CXXFLAGS=-Iincludedir; export CXXFLAGS  
LDFLAGS=-Llibdir; export LDFLAGS
```

For C shell and tcsh, use:

```
setenv CFLAGS "-Iincludedir"  
setenv CXXFLAGS "-Iincludedir"  
setenv LDFLAGS "-Llibdir"
```

By default configure will look for a C++ compiler named `CC`, `c++`, `g++`, or `gcc` in that order. To use another compiler you need to set the `CXX` environment variable:

```
CXX=xlc; export CXX  
setenv CXX "xlc"
```

The `CC` environment variable can also be used to override the default C compiler (`cc` or `gcc`), which is used for a few FLTK source files.

You can run configure yourself to get the exact setup you need. Type `./configure <options>`, where options are:

-enable-cygwin

Enable the Cygwin libraries under Windows

-enable-debug

Enable debugging code & symbols

-disable-gl

Disable OpenGL support

-disable-print

Disable print support for an X11 platform

-enable-shared

Enable generation of shared libraries

-enable-threads

Enable multithreading support

-enable-xdbe

Enable the X double-buffer extension

-enable-xft

Enable the Xft library for anti-aliased fonts under X11

-enable-pango

Enable the pango library for drawing any text in any script under X11.

-enable-x11

When targeting cygwin, build with X11 GUI instead of windows GDI. Also applicable to macOS platforms supplemented with XQuartz.

-enable-cp936

Under X11, enable use of the GB2312 locale

-bindir=/path

Set the location for executables [default = \$prefix/bin]

-datadir=/path

Set the location for data files. [default = \$prefix/share]

-libdir=/path

Set the location for libraries [default = \$prefix/lib]

-includedir=/path

Set the location for include files. [default = \$prefix/include]

-mandir=/path

Set the location for man pages. [default = \$prefix/man]

-prefix=/dir

Set the directory prefix for files [default = /usr/local]

When the configure script is done you can just run the "make" command. This will build the library, FLUID tool, and all of the test programs.

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

3.6 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 file "README.Windows.txt" and you should read that file to determine if there are changes that may be applicable to your build environment.

FLTK 1.3 is officially supported on Windows (2000,) 2003, XP, and later. Older Windows versions prior to Windows 2000 are not officially supported, but may still work. The main reason is that the OS version needs to support UTF-8. FLTK 1.3 is known to work on recent versions of Windows such as Windows 7, Windows 8/8.1 and Windows 10 and has been reported to work in both 32-bit and 64-bit versions of these.

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

CAUTION: Libraries built by any one of these build environments can not be mixed with object files from any of the other environments! (They use incompatible C++ conventions internally.)

Free Microsoft Visual C++ 2008 Express and Visual C++ 2010 Express or later versions using 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. Be sure to get your service packs!

Since FLTK 1.4 the project files MUST be generated with CMake. Please read "README.CMake.txt" for more information about this.

3.6.1 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 in the section on *Building and Installing FLTK Under UNIX and Apple macOS* apply, in particular the descriptions of using the "configure" script and its related options.

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:

```
./configure  
make
```

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:

```
make install
```

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:

```
sh configure --prefix=C:/FLTK  
make
```

Where the value passed to "prefix" is the path at which you would like FLTK to be installed.

A subsequent invocation of "make install" will then place the FLTK libraries and header files into that path.

The other options to "configure" may also be used to tailor the build to suit your environment.

3.6.2 Using the Visual C++ DLL Library

The "fltkdll.dsp" project file builds a DLL-version of the FLTK library. Because of name mangling differences between PC compilers (even between different versions of Visual C++!) you can only use the DLL that is generated with the same version compiler that you built it with.

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

3.7 Internet Resources

FLTK is available on the 'net in a bunch of locations:

WWW

<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]

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]

3.8 Reporting Bugs

To report a bug in FLTK, or for feature requests, please use the form at <https://www.fltk.org/bugs.php>, and click on "Submit Bug or Feature Request".

You'll be prompted for the FLTK version, operating system & version, and compiler that you are using. We will be unable to provide any kind of help without that basic information.

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>.

Chapter 4

FLTK Basics

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

4.1 Writing Your First FLTK Program

All programs must include the file <FL/Fl.H>. 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 window in Figure 4.1. You can quit the program by closing the window or pressing the `ESCAPE` key.

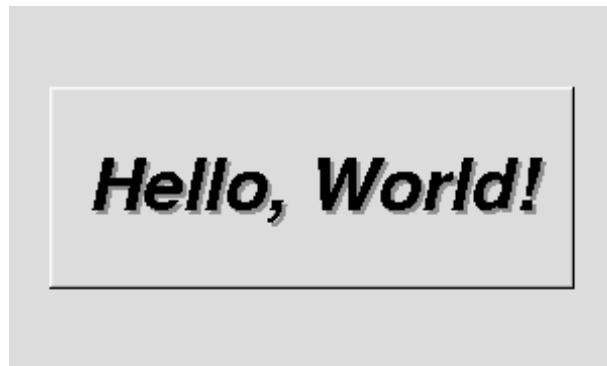


Figure 4.1 The Hello, World! Window

4.1.1 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`) and the units are in pixels.

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

`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.

4.1.2 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 `myGroup->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.

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(...)`.

4.1.3 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 boxtyle in 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".

4.1.4 Redrawing After Changing Attributes

Almost all of the set/get 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.

4.1.5 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`. You can also specify typefaces directly.

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

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](#).

4.1.6 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.

4.1.7 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.

4.2 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:

```
CC -I/usr/local/include ...
gcc -I/usr/local/include ...
```

The `fltk-config` script included with FLTK can be used to get the options that are required by your compiler:

```
CC `fltk-config --cxxflags` ...
```

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

```
CC ... -L/usr/local/lib -lfltk -lXext -lX11 -lm
gcc ... -L/usr/local/lib -lfltk -lXext -lX11 -lm
```

Aside from the "fltk" library, there is also a "fltk_forms" library for the XForms compatibility classes, "fltk_gl" for the OpenGL and GLUT classes, and "fltk_images" for the image file classes, [Fl_Help_Dialog](#) widget, and system icon support.

Note

The libraries are named "fltk.lib", "fltkgl.lib", "fltkforms.lib", and "fltkimages.lib", respectively under Windows.

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

```
CC ... `fltk-config --ldflags`
```

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

```
CC ... `fltk-config --use-forms --ldflags`
CC ... `fltk-config --use-gl --ldflags`
CC ... `fltk-config --use-images --ldflags`
CC ... `fltk-config --use-forms --use-gl --use-images --ldflags`
```

Finally, you can use the `fltk-config` script to compile a single source file as a FLTK program:

```
fltk-config --compile filename.cpp
fltk-config --use-forms --compile filename.cpp
fltk-config --use-gl --compile filename.cpp
fltk-config --use-images --compile filename.cpp
fltk-config --use-forms --use-gl --use-images --compile filename.cpp
```

Any of these will create an executable named `filename`.

4.3 Compiling Programs with Makefiles

The previous section described how to use `fltk-config` to build a program consisting of a single source file 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 programs out of multiple source files:

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

$(TARGET): $(OBJS)
CubeMain.o: CubeMain.cxx CubeViewUI.h
CubeView.o: CubeView.cxx CubeView.h CubeViewUI.h
CubeViewUI.o: CubeViewUI.cxx CubeView.h

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

4.4 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 FLTK (`FLTK.LIB` or `FLTKD.LIB`) and the Windows Common Controls (`COMCTL32.LIB`) libraries to the "Link" settings.

You must also define `_WIN32` if the compiler doesn't do this. Currently all known Windows compilers define `_WIN32` - unless you use Cygwin. 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.

4.5 Naming

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/...>`.

4.6 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>
```

Chapter 5

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.

5.1 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.

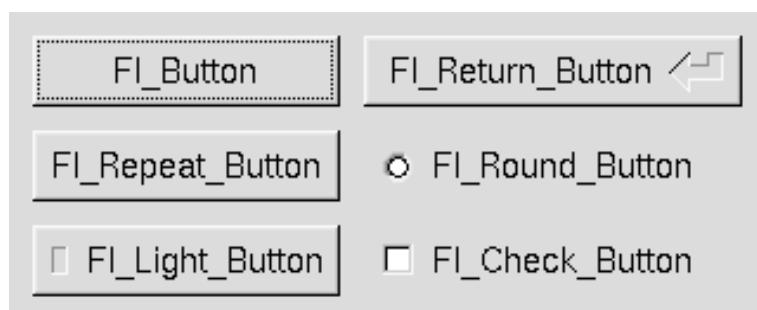


Figure 5.1 FLTK Button Widgets

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.

5.2 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.

5.3 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.

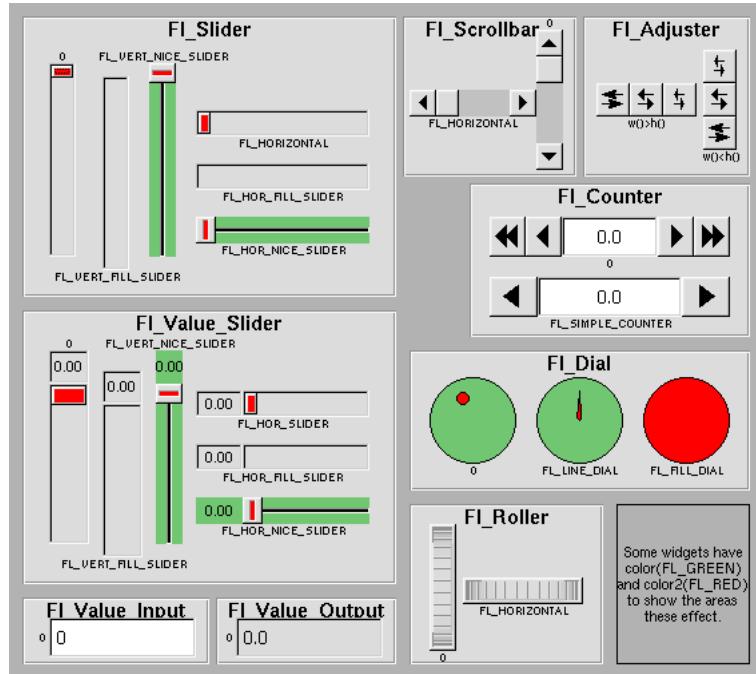


Figure 5.2 FLTK valuator widgets

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.

5.4 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_GL_Window](#) - An OpenGL window on the screen.
- [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.

5.5 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.

5.6 Colors

FLTK stores the colors of widgets as an 32-bit unsigned number that is either an index into a color palette of 256 colors or a 24-bit RGB color. The color palette is *not* the X or MS 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
- 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(0xffff0000);                  // RGB: red
button->color(0x00ff0000);                  // RGB: green
button->color(0x0000ff00);                  // RGB: blue
button->color(0xffffffff);                  // RGB: white
```

Note

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

5.7 Box Types

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

Figure 5.3 shows the standard box types included with FLTK.

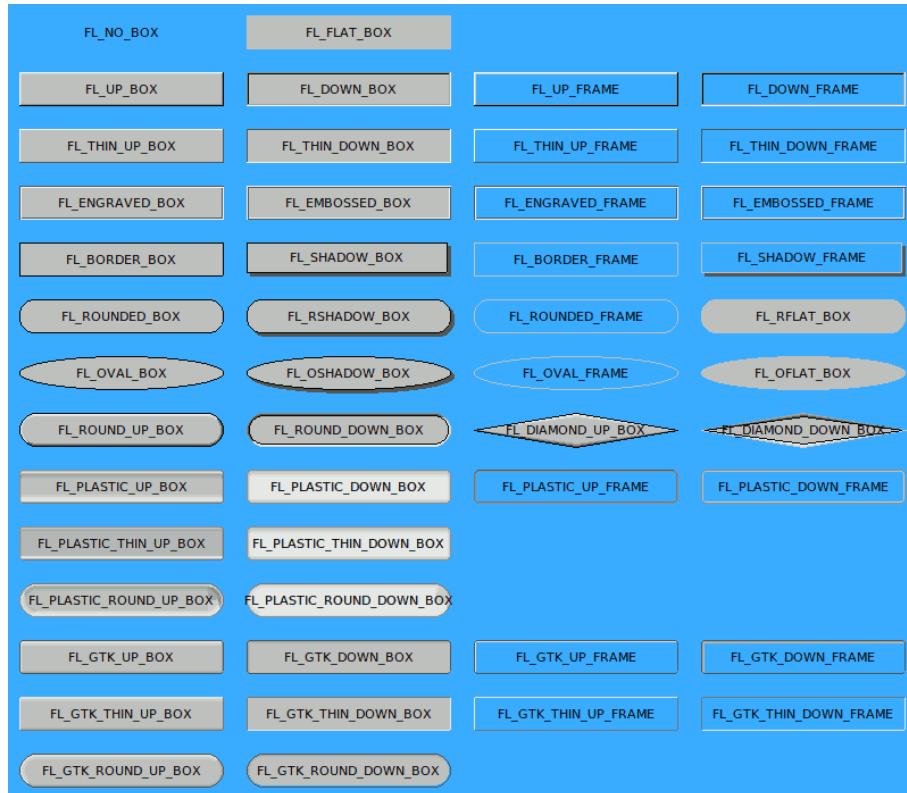


Figure 5.3 FLTK box types

`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 Figure 5.3 is the area that is not drawn by the frame types.

5.7.1 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 boxtyle with a numerical value evenly dividable by two. [fl_frame\(Fl_Boxtype\)](#) expects the UP_BOX design at a value dividable by four.

5.8 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. Figure 5.4 shows the available symbols.



Figure 5.4 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.

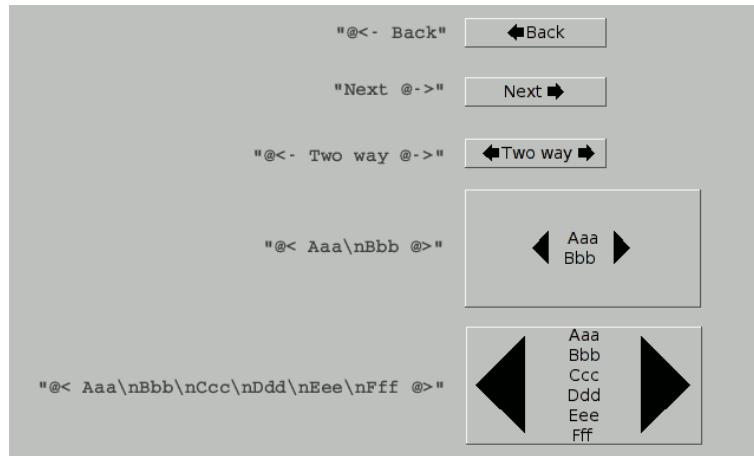


Figure 5.5 FLTK symbols and text

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_EMBOSS_LABEL` - draws edges as though the text is raised.
- `FL_ICON_LABEL` - draws the icon ([FI_Image](#)) associated with the text.
- `FL_IMAGE_LABEL` - draws the image ([FI_Image](#)) associated with the text.
- `FL_MULTI_LABEL` - draws multiple parts side by side, see [FI_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 [FI_Menu_Item](#)'s to support icons or small images, typically left of the menu text.

As of this writing (FLTK 1.4.0, Sep 2017) [FI_Menu_Items](#) support only one label part (text **or** image), but using [FI_Multi_Label](#) as the label can extend this to more than one part.

See also

class [FI_Multi_Label](#), [FI_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 [FI_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_FREELABELTYPE
Fl::set_labeltype(XYZ_LABEL, xyz_draw, xyz_measure);
```

The label type number `n` can be any integer value starting at the constant `FL_FREELABELTYPE`. 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.

5.9 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);
```

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);
```

Note:

You cannot delete a widget inside a callback, as the widget may still be accessed by FLTK after your callback is completed. Instead, use the [Fl::delete_widget\(\)](#) method to mark your widget for deletion when it is safe to do so.

Hint:

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);
```

5.10 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.

Chapter 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.

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(0); // no resizing  
group->end()
```

The `resizable` may be set to zero, 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.

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.

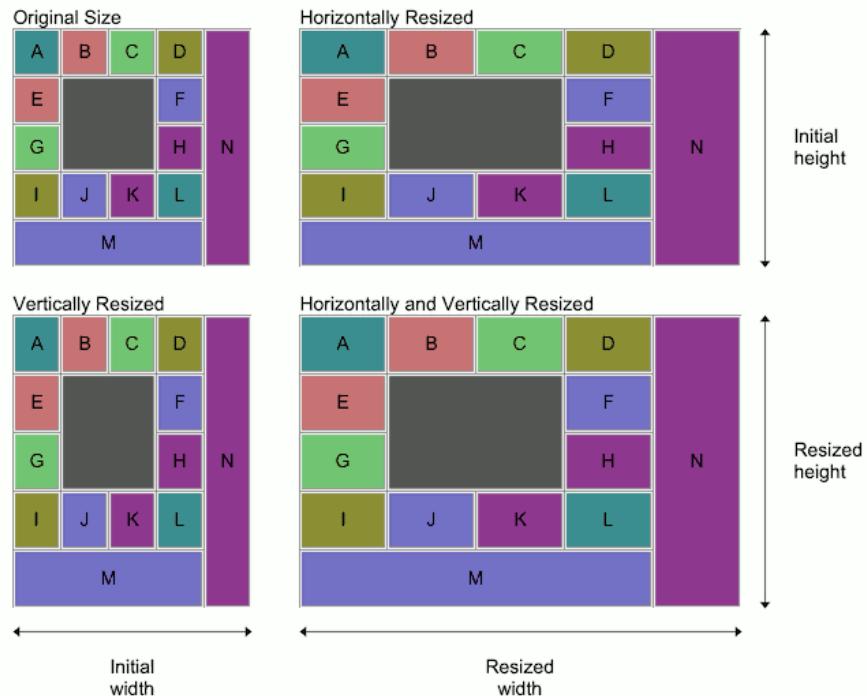


Figure 6.1 Proportional resizing example

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.

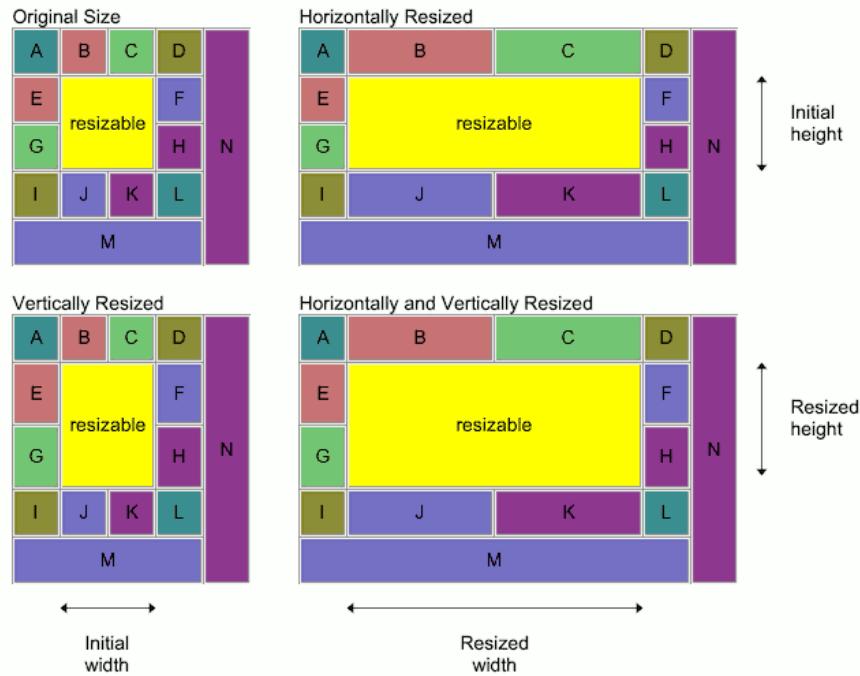


Figure 6.2 Complex resizing example

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:

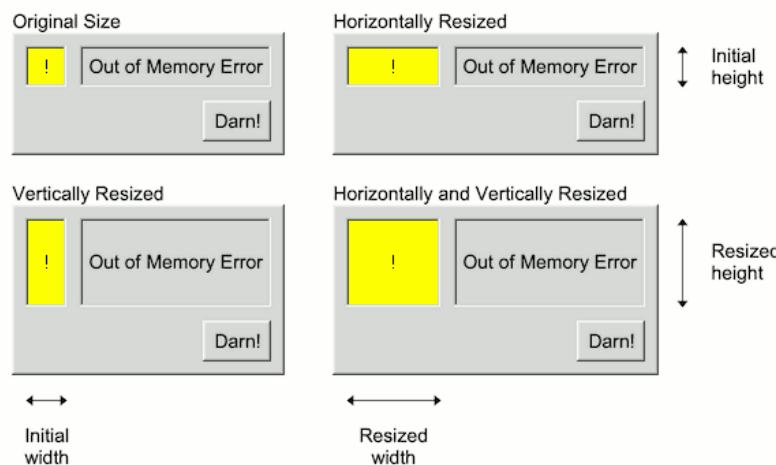


Figure 6.3 Resizing dialog example (a)

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 behaviour shown in the diagram below.

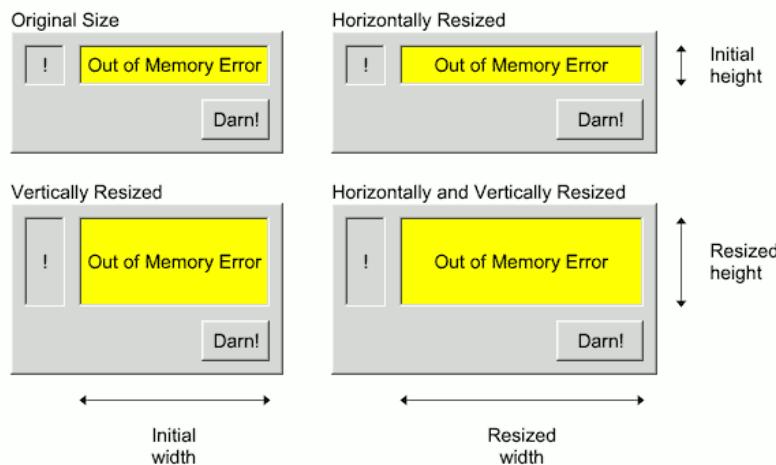


Figure 6.4 Resizing dialog example (b)

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.

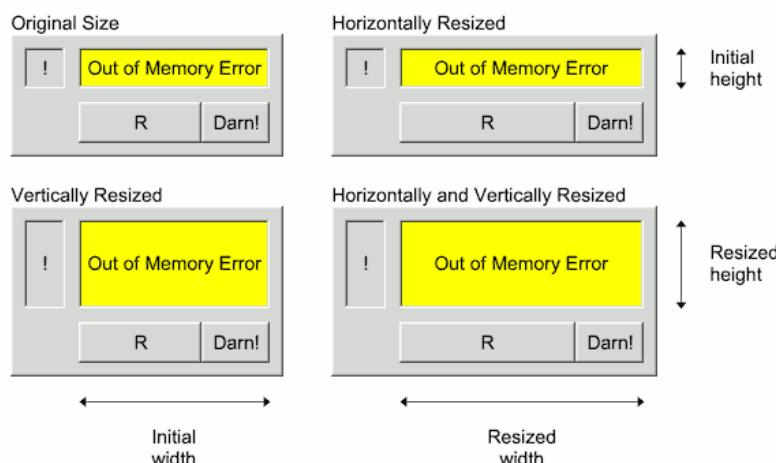


Figure 6.5 Resizing dialog example (c)

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");
bttns = 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
bttns->resizable(R); // make "R" parent group resizable
bttns->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 `resizable` leaves the other one fixed, as shown below:

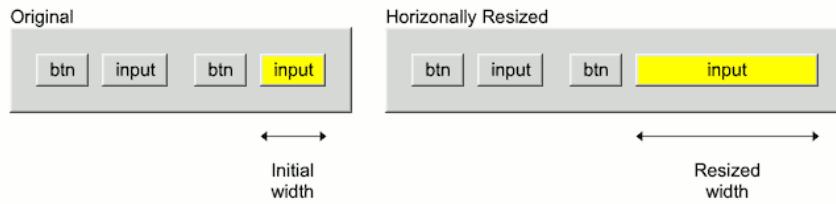


Figure 6.6 Resizing input fields example (b)

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!

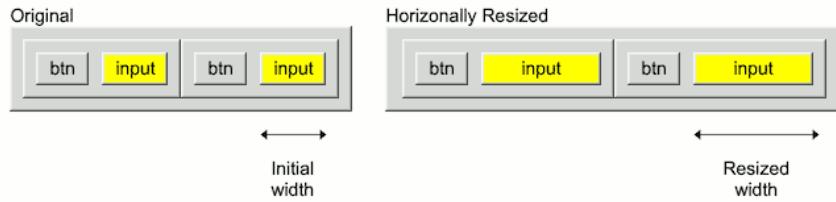


Figure 6.7 Resizing input fields example (b)

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.

Chapter 7

Designing a Simple Text Editor

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

7.1 Determining the Goals of the Text Editor

Since this will be the first big project you'll be doing with FLTK, let's define what we want our text editor to do:

1. Provide a menubar/menus for all functions.
2. Edit a single text file, possibly with multiple views.
3. Load from a file.
4. Save to a file.
5. Cut/copy/delete/paste functions.
6. Search and replace functions.
7. Keep track of when the file has been changed.

7.2 Designing the Main Window

Now that we've outlined the goals for our editor, we can begin with the design of our GUI. Obviously the first thing that we need is a window, which we'll place inside a class called `EditorWindow`:

```
class EditorWindow : public Fl_Double_Window {  
public:  
    EditorWindow(int w, int h, const char* t);  
    ~EditorWindow();  
  
    Fl_Window      *replace_dlg;  
    Fl_Input       *replace_find;  
    Fl_Input       *replace_with;  
    Fl_Button      *replace_all;  
    Fl_Return_Button *replace_next;  
    Fl_Button      *replace_cancel;  
  
    Fl_Text_Editor *editor;  
    char           search[256];  
};
```

7.3 Variables

Our text editor will need some global variables to keep track of things:

```
int          changed = 0;
char        filename[256] = "";
F1_Text_Buffer *textbuf;
```

The `textbuf` variable is the text editor buffer for our window class described previously. We'll cover the other variables as we build the application.

7.4 Menubars and Menus

The first goal requires us to use a menubar and menus that define each function the editor needs to perform. The `F1_Menu_Item` structure is used to define the menus and items in a menubar:

```
F1_Menu_Item menuitems[] = {
{ "&File",           0, 0, 0, FL_SUBMENU },
{ "&New File",        0, (F1_Callback *)new_cb },
{ "&Open File...",   FL_COMMAND + 'o', (F1_Callback *)open_cb },
{ "&Insert File...", FL_COMMAND + 'i', (F1_Callback *)insert_cb, 0,
  FL_MENU_DIVIDER },
{ "&Save File",       FL_COMMAND + 's', (F1_Callback *)save_cb },
{ "Save File &As...", FL_COMMAND + FL_SHIFT + 's', (
  F1_Callback *)saveas_cb, 0, FL_MENU_DIVIDER },
{ "New &View",        FL_ALT + 'v', (F1_Callback *)view_cb, 0 },
{ "&Close View",      FL_COMMAND + 'w', (F1_Callback *)close_cb, 0,
  FL_MENU_DIVIDER },
{ "E&xit",            FL_COMMAND + 'q', (F1_Callback *)quit_cb, 0 },
{ 0 },

{ "&Edit",            0, 0, 0, FL_SUBMENU },
{ "&Undo",             FL_COMMAND + 'z', (F1_Callback *)undo_cb, 0,
  FL_MENU_DIVIDER },
{ "Cu&t",              FL_COMMAND + 'x', (F1_Callback *)cut_cb },
{ "&Copy",              FL_COMMAND + 'c', (F1_Callback *)copy_cb },
{ "&Paste",              FL_COMMAND + 'v', (F1_Callback *)paste_cb },
{ "&Delete",            0, (F1_Callback *)delete_cb },
{ 0 },

{ "&Search",           0, 0, 0, FL_SUBMENU },
{ "&Find...",          FL_COMMAND + 'f', (F1_Callback *)find_cb },
{ "F&ind Again",        FL_COMMAND + 'g', find2_cb },
{ "&Replace...",        FL_COMMAND + 'r', replace_cb },
{ "Re&place Again",     FL_COMMAND + 't', replace2_cb },
{ 0 },

{ 0 }
};
```

Once we have the menus defined we can create the `F1_Menu_Bar` widget and assign the menus to it with:

```
F1_Menu_Bar *m = new F1_Menu_Bar(0, 0, 640, 30);
m->copy(menuitems);
```

We'll define the callback functions later.

7.5 Editing the Text

To keep things simple our text editor will use the [Fl_Text_Editor](#) widget to edit the text:

```
w->editor = new Fl_Text_Editor(0, 30, 640, 370);
w->editor->buffer(textbuf);
```

So that we can keep track of changes to the file, we also want to add a "modify" callback:

```
textbuf->add_modify_callback(changed_cb, w);
textbuf->call_modify_callbacks();
```

Finally, we want to use a mono-spaced font like [FL_COURIER](#):

```
w->editor->textfont(Fl_COURIER);
```

7.6 The Replace Dialog

We can use the FLTK convenience functions for many of the editor's dialogs, however the replace dialog needs its own custom window. To keep things simple we will have a "find" string, a "replace" string, and "replace all", "replace next", and "cancel" buttons. The strings are just [Fl_Input](#) widgets, the "replace all" and "cancel" buttons are [Fl_Button](#) widgets, and the "replace next" button is a [Fl_Return_Button](#) widget:

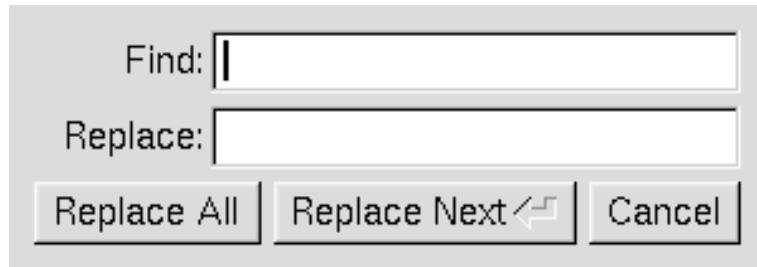


Figure 7.1 The search and replace dialog

```
Fl_Window *replace_dlg = new Fl_Window(300, 105, "Replace");
Fl_Input *replace_find = new Fl_Input(70, 10, 200, 25, "Find:");
Fl_Input *replace_with = new Fl_Input(70, 40, 200, 25, "Replace:");
Fl_Button *replace_all = new Fl_Button(10, 70, 90, 25, "Replace All");
Fl_Button *replace_next = new Fl_Button(105, 70, 120, 25, "Replace Next");
Fl_Button *replace_cancel = new Fl_Button(230, 70, 60, 25, "Cancel");
```

7.7 Callbacks

Now that we've defined the GUI components of our editor, we need to define our callback functions.

7.7.1 changed_cb()

This function will be called whenever the user changes any text in the `editor` widget:

```
void changed_cb(int, int nInserted, int nDeleted, const char*, void* v) {
    if ((nInserted || nDeleted) && !loading) changed = 1;
    EditorWindow *w = (EditorWindow *)v;
    set_title(w);
    if (loading) w->editor->show_insert_position();
}
```

The `set_title()` function is one that we will write to set the changed status on the current file. We're doing it this way because we want to show the changed status in the window's title bar.

7.7.2 copy_cb()

This callback function will call [Fl_Text_Editor::kf_copy\(\)](#) to copy the currently selected text to the clipboard:

```
void copy_cb(Fl_Widget*, void* v) {
    EditorWindow* e = (EditorWindow*)v;
    Fl_Text_Editor::kf_copy(0, e->editor);
}
```

7.7.3 cut_cb()

This callback function will call [Fl_Text_Editor::kf_cut\(\)](#) to cut the currently selected text to the clipboard:

```
void cut_cb(Fl_Widget*, void* v) {
    EditorWindow* e = (EditorWindow*)v;
    Fl_Text_Editor::kf_cut(0, e->editor);
}
```

7.7.4 delete_cb()

This callback function will call [Fl_Text_Buffer::remove_selection\(\)](#) to delete the currently selected text to the clipboard:

```
void delete_cb(Fl_Widget*, void* v) {
    textbuf->remove_selection();
}
```

7.7.5 find_cb()

This callback function asks for a search string using the [fl_input\(\)](#) convenience function and then calls the `find2_cb()` function to find the string:

```
void find_cb(Fl_Widget* w, void* v) {
    EditorWindow* e = (EditorWindow*)v;
    const char *val;

    val = fl_input("Search String:", e->search);
    if (val != NULL) {
        // User entered a string - go find it!
        strcpy(e->search, val);
        find2_cb(w, v);
    }
}
```

7.7.6 find2_cb()

This function will find the next occurrence of the search string. If the search string is blank then we want to pop up the search dialog:

```
void find2_cb(Fl_Widget* w, void* v) {
    EditorWindow* e = (EditorWindow*)v;
    if (e->search[0] == '\0') {
        // Search string is blank; get a new one...
        find_cb(w, v);
        return;
    }

    int pos = e->editor->insert_position();
    int found = textbuf->search_forward(pos, e->search, &pos);
    if (found) {
        // Found a match; select and update the position...
        textbuf->select(pos, pos+strlen(e->search));
        e->editor->insert_position(pos+strlen(e->search));
        e->editor->show_insert_position();
    }
    else fl_alert("No occurrences of '%s' found!", e->search);
}
```

If the search string cannot be found we use the `fl_alert()` convenience function to display a message to that effect.

7.7.7 new_cb()

This callback function will clear the editor widget and current filename. It also calls the `check_save()` function to give the user the opportunity to save the current file first as needed:

```
void new_cb(Fl_Widget*, void*) {
    if (!check_save()) return;

    filename[0] = '\0';
    textbuf->select(0, textbuf->length());
    textbuf->remove_selection();
    changed = 0;
    textbuf->call_modify_callbacks();
}
```

7.7.8 open_cb()

This callback function will ask the user for a filename and then load the specified file into the input widget and current filename. It also calls the `check_save()` function to give the user the opportunity to save the current file first as needed:

```
void open_cb(Fl_Widget*, void*) {
    if (!check_save()) return;

    char *newfile = fl_file_chooser("Open File?", "*", filename);
    if (newfile != NULL) load_file(newfile, -1);
}
```

We call the `load_file()` function to actually load the file.

7.7.9 paste_cb()

This callback function will call `Fl_Text_Editor::kf_paste()` to paste the clipboard at the current position:

```
void paste_cb(Fl_Widget*, void* v) {
    EditorWindow* e = (EditorWindow*)v;
    Fl_Text_Editor::kf_paste(0, e->editor);
}
```

7.7.10 quit_cb()

The quit callback will first see if the current file has been modified, and if so give the user a chance to save it. It then exits from the program:

```
void quit_cb(Fl_Widget*, void*) {
    if (changed && !check_save())
        return;
    exit(0);
}
```

7.7.11 replace_cb()

The replace callback just shows the replace dialog:

```
void replace_cb(Fl_Widget*, void* v) {
    EditorWindow* e = (EditorWindow*)v;
    e->replace_dlg->show();
}
```

7.7.12 replace2_cb()

This callback will replace the next occurrence of the replacement string. If nothing has been entered for the replacement string, then the replace dialog is displayed instead:

```
void replace2_cb(Fl_Widget*, void* v) {
    EditorWindow* e = (EditorWindow*)v;
    const char *find = e->replace_find->value();
    const char *replace = e->replace_with->value();

    if (find[0] == '\0') {
        // Search string is blank; get a new one...
        e->replace_dlg->show();
        return;
    }

    e->replace_dlg->hide();

    int pos = e->editor->insert_position();
    int found = textbuf->search_forward(pos, find, &pos);

    if (found) {
        // Found a match; update the position and replace text...
        textbuf->select(pos, pos+strlen(find));
        textbuf->remove_selection();
        textbuf->insert(pos, replace);
        textbuf->select(pos, pos+strlen(replace));
        e->editor->insert_position(pos+strlen(replace));
        e->editor->show_insert_position();
    }
    else fl_alert("No occurrences of '%s' found!", find);
}
```

7.7.13 replall_cb()

This callback will replace all occurrences of the search string in the file:

```

void replall_cb(Fl_Widget*, void* v) {
    EditorWindow* e = (EditorWindow*)v;
    const char *find = e->replace_find->value();
    const char *replace = e->replace_with->value();

    find = e->replace_find->value();
    if (find[0] == '\0') {
        // Search string is blank; get a new one...
        e->replace_dlg->show();
        return;
    }

    e->replace_dlg->hide();
    e->editor->insert_position(0);
    int times = 0;

    // Loop through the whole string
    for (int found = 1; found;) {
        int pos = e->editor->insert_position();
        found = textbuf->search_forward(pos, find, &pos);

        if (found) {
            // Found a match; update the position and replace text...
            textbuf->select(pos, pos+strlen(find));
            textbuf->remove_selection();
            textbuf->insert(pos, replace);
            e->editor->insert_position(pos+strlen(replace));
            e->editor->show_insert_position();
            times++;
        }
    }

    if (times) fl_message("Replaced %d occurrences.", times);
    else fl_alert("No occurrences of '%s' found!", find);
}

```

7.7.14 replican_cb()

This callback just hides the replace dialog:

```

void replican_cb(Fl_Widget*, void* v) {
    EditorWindow* e = (EditorWindow*)v;
    e->replace_dlg->hide();
}

```

7.7.15 save_cb()

This callback saves the current file. If the current filename is blank it calls the "save as" callback:

```

void save_cb(void) {
    if (filename[0] == '\0') {
        // No filename - get one!
        saveas_cb();
        return;
    }
    else save_file(filename);
}

```

The `save_file()` function saves the current file to the specified filename.

7.7.16 saveas_cb()

This callback asks the user for a filename and saves the current file:

```

void saveas_cb(void) {
    char *newfile;

    newfile = fl_file_chooser("Save File As?", "*", filename);
    if (newfile != NULL) save_file(newfile);
}

```

The `save_file()` function saves the current file to the specified filename.

7.8 Other Functions

Now that we've defined the callback functions, we need our support functions to make it all work:

7.8.1 check_save()

This function checks to see if the current file needs to be saved. If so, it asks the user if they want to save it:

```
int check_save(void) {
    if (!changed) return 1;

    int r = fl_choice("The current file has not been saved.\n"
                      "Would you like to save it now?",
                      "Cancel", "Save", "Discard");

    if (r == 1) {
        save_cb(); // Save the file...
        return !changed;
    }

    return (r == 2) ? 1 : 0;
}
```

7.8.2 load_file()

This function loads the specified file into the `textbuf` variable:

```
int loading = 0;
void load_file(char *newfile, int ipos) {
    loading = 1;
    int insert = (ipos != -1);
    changed = insert;
    if (!insert) strcpy(filename, "");
    int r;
    if (!insert) r = textbuf->loadfile(newfile);
    else r = textbuf->insertfile(newfile, ipos);
    if (r)
        fl_alert("Error reading from file \'%s\':\n%s.", newfile, strerror(errno));
    else
        if (!insert) strcpy(filename, newfile);
    loading = 0;
    textbuf->call_modify_callbacks();
}
```

When loading the file we use the `Fl_Text_Buffer::loadfile()` method to "replace" the text in the buffer, or the `Fl_Text_Buffer::insertfile()` method to insert text in the buffer from the named file.

7.8.3 save_file()

This function saves the current buffer to the specified file:

```
void save_file(char *newfile) {
    if (textbuf->savefile(newfile))
        fl_alert("Error writing to file \'%s\':\n%s.", newfile, strerror(errno));
    else
        strcpy(filename, newfile);
    changed = 0;
    textbuf->call_modify_callbacks();
}
```

7.8.4 set_title()

This function checks the `changed` variable and updates the window label accordingly:

```
void set_title(Fl_Window* w) {
    if (filename[0] == '\0') strcpy(title, "Untitled");
    else {
        char *slash;
        slash = strrchr(filename, '/');
#ifndef _WIN32
        if (slash == NULL) slash = strrchr(filename, '\\');
#endif
        if (slash != NULL) strcpy(title, slash + 1);
        else strcpy(title, filename);
    }

    if (changed) strcat(title, " (modified)");

    w->label(title);
}
```

7.9 The main() Function

Once we've created all of the support functions, the only thing left is to tie them all together with the `main()` function. The `main()` function creates a new text buffer, creates a new view (window) for the text, shows the window, loads the file on the command-line (if any), and then enters the FLTK event loop:

```
int main(int argc, char **argv) {
    textbuf = new Fl_Text_Buffer;

    Fl_Window* window = new_view();

    window->show(1, argv);

    if (argc > 1) load_file(argv[1], -1);

    return Fl::run();
}
```

7.10 Compiling the Editor

The complete source for our text editor can be found in the `test/editor.cxx` source file. Both the Makefile and Visual C++ workspace include the necessary rules to build the editor. You can also compile it using a standard compiler with:

```
CC -o editor editor.cxx -lfltk -lxext -lx11 -lm
```

or by using the `fltk-config` script with:

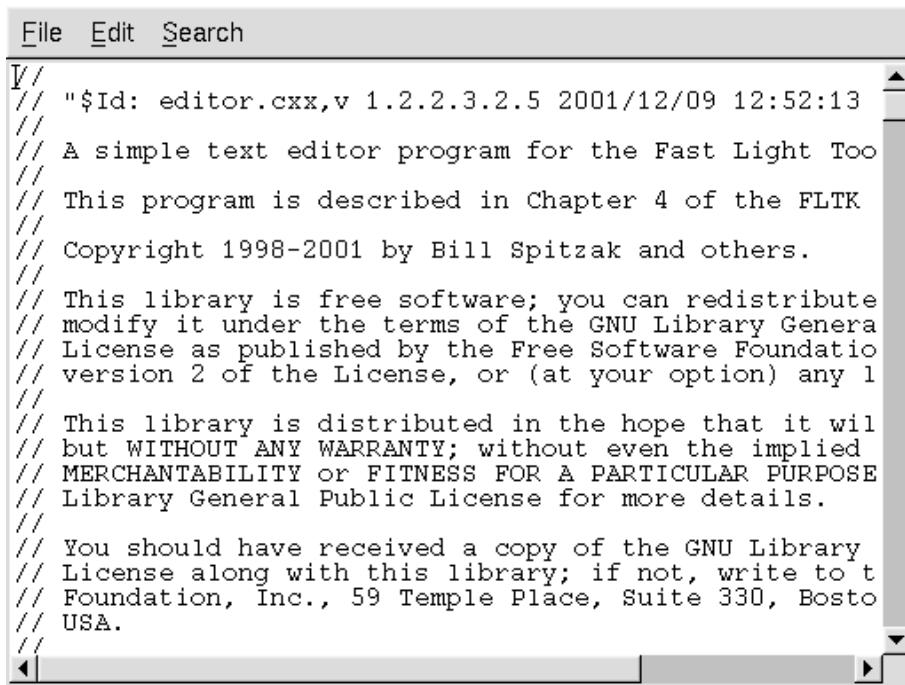
```
fltk-config --compile editor.cxx
```

As noted in [Compiling Programs with Standard Compilers](#), you may need to include compiler and linker options to tell them where to find the FLTK library. Also, the `CC` command may also be called `gcc` or `c++` on your system.

Congratulations, you've just built your own text editor!

7.11 The Final Product

The final editor window should look like the image in Figure 7.2.



A screenshot of a text editor window titled "File Edit Search". The main content area contains the text of the GNU General Public License. The text is as follows:

```

/*
// $Id: editor.cxx,v 1.2.2.3.2.5 2001/12/09 12:52:13
// A simple text editor program for the Fast Light Too
// This program is described in Chapter 4 of the FLTK
// Copyright 1998-2001 by Bill Spitzak and others.
// This library is free software; you can redistribute
// modify it under the terms of the GNU Library Genera
// License as published by the Free Software Foundatio
// version 2 of the License, or (at your option) any l
//
// This library is distributed in the hope that it wil
// but WITHOUT ANY WARRANTY; without even the implied
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE
// Library General Public License for more details.
//
// You should have received a copy of the GNU Library
// License along with this library; if not, write to t
// Foundation, Inc., 59 Temple Place, Suite 330, Bosto
// USA.
*/

```

Figure 7.2 The completed editor window

7.12 Advanced Features

Now that we've implemented the basic functionality, it is time to show off some of the advanced features of the [Fl_Text_Editor](#) widget.

7.12.1 Syntax Highlighting

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_Editor.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:

```

F1_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:

```

F1_Text_Buffer *stylebuf;

w->editor->highlight_data(stylebuf, 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:

```
textbuf->add_modify_callback(style_update, w->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) {
        stylebuf->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';

        stylebuf->replace(pos, pos + nDeleted, style);
        delete[] style;
    } else {
        // Just delete characters in the style buffer...
        stylebuf->remove(pos, pos + nDeleted);
    }

    // Select the area that was just updated to avoid unnecessary
    // callbacks...
    stylebuf->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 = textbuf->line_start(pos);
end = textbuf->line_end(pos + nInserted - nDeleted);
text = textbuf->text_range(start, end);
style = stylebuf->text_range(start, end);
last = style[end - start - 1];

style_parse(text, style, end - start);

stylebuf->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 = textbuf->length();
    text = textbuf->text_range(start, end);
    style = stylebuf->text_range(start, end);

    style_parse(text, style, end - start);

    stylebuf->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';
}
}
```


Chapter 8

Drawing Things in FLTK

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

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()`.

8.1.1 What Drawing Unit do FLTK drawing functions use?

Before version 1.4, all graphical quantities used by FLTK are in pixel units: a window of width 500 units is 500-pixel wide, a line of length 10 units is 10-pixel long, lines of text written using a 14-point font are 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-pixel wide, a line of length 10 units is 10*scale-pixel 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 data visible by the public API (e.g., window widths, line lengths, font sizes, clipping regions) are in **FLTK units** which are both system- and DPI-independent.
- 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 Mac OS X and for retina displays, where one drawing unit corresponds to two pixels.

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](#) and [Windows](#) 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 Mac OS X, the corresponding GUI scaling shortcuts are `/+/-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) independently from the size of the image data. 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\(\)](#).

8.2 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](#)
- [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](#)

8.2.1 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 boxtyle `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.

8.2.2 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)
void fl_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.

The `fl_clip()` version is deprecated and will be removed from future releases.

```
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.

8.3 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 1.3.x 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:

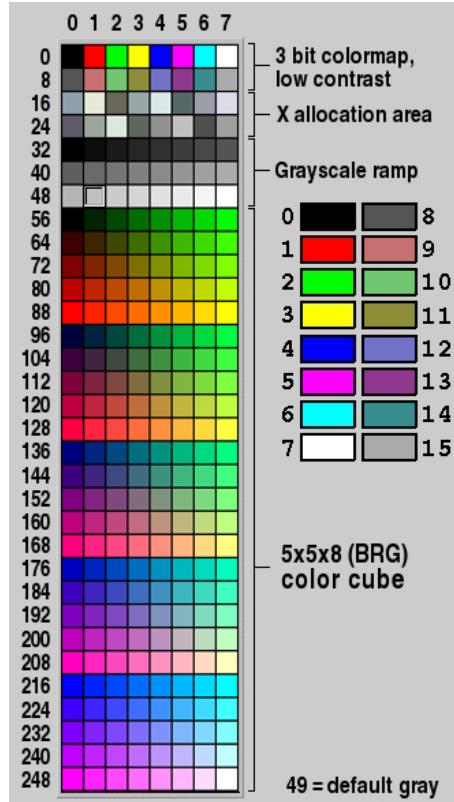


Figure 8.1 FLTK default colormap (Fl_Color 0x00 - 0xff)

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 1.3.x colormap or a direct RGB value on TrueColor screens.

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

Generate FI_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(0xfffffff0);                                // RGB: white
```

Note

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

8.3.1 Line Dashes and Thickness

FLTK supports drawing of lines with different styles and widths. Full functionality is not available under Windows 95, 98, and Me due to the reduced drawing functionality these operating systems provide.

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

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)`.

Note: Because of how line styles are implemented on MS 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!

`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 -----
- FL_DASH - - - -
- FL_DOT
- FL_DASHDOT - . - .

- `FL_DASHDOTDOT` - . . -
- `FL_CAP_FLAT`
- `FL_CAP_ROUND`
- `FL_CAP_SQUARE` (extends past end point 1/2 line width)
- `FL_JOIN_MITER` (pointed)
- `FL_JOIN_ROUND`
- `FL_JOIN_BEVEL` (flat)

`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: The dashes array does not work under Windows 95, 98, or Me, since those operating systems do not support complex line styles.

8.3.2 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_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 MS 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, a_2 must be greater or equal to a_1 .

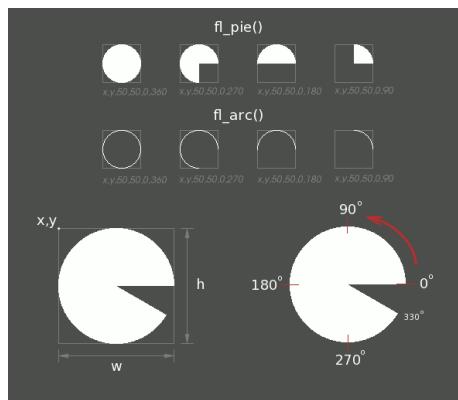


Figure 8.2 `fl_pie()` and `fl_arc()`

`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,

8.3.3 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 MS 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_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 Bezier curve to the path. The curve ends (and two of the points are) at X_0, Y_0 and X_3, Y_3 .

```
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.

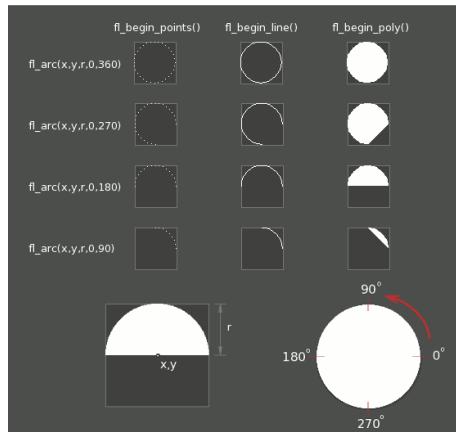


Figure 8.3 `fl_arc(x,y,r,a1,a2)`

```
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.

8.3.4 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)
```

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.

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.

8.3.5 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.

8.3.6 Character Encoding

FLTK 1.3 expects 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 `OPTION_USE_PANGO` CMake option turned On (or with configure `--enable-pango`).

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](#).

8.3.7 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.

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.

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.

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.

8.4.3 Image Classes

FLTK provides a base image class called [FI_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 [FI_Image::scale\(\)](#)).

The [FI_Image](#) class does almost nothing by itself, but is instead supported by three basic image types:

- [FI_Bitmap](#)
- [FI_Pixmap](#)
- [FI_RGB_Image](#)

The [FI_Bitmap](#) class encapsulates a mono-color bitmap image. The `draw()` method draws the image using the current drawing color.

The [FI_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 [FI_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:

- [FI_GIF_Image](#)
- [FI_JPEG_Image](#)

- [FI_PNG_Image](#)
- [FI_PNM_Image](#)
- [FI_XBM_Image](#)
- [FI_XPM_Image](#)
- [FI_SVG_Image](#)

Each of these image classes loads a named file of the corresponding format. The [FI_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 [FI_Tiled_Image](#) to tile another image object in the specified area. This class can be used to tile a background image in a [FI_Group](#) widget, for example.

```
virtual void FI_Image::copy()  
virtual FI_Image* FI_Image::copy(int w, int h)
```

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 [FI_Image::RGB_scaling\(FI_RGB_Scaling method\)](#)

```
virtual void FI_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 [FI_Tiled_Image::draw\(\)](#) regarding arguments `ox, oy, w, and h`.

```
virtual void FI_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 [FI_Image](#) type.

8.4.4 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.

`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. FLTK can draw into a buffer at any time. There is no need to wait for an `Fl_Widget::draw()` to occur.

`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.

Chapter 9

Handling Events

This chapter discusses the FLTK event model and how to handle events in your program or widget.

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.

Event numbers can be converted to their actual names using the `fl_eventnames[]` array 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.

9.2 Mouse Events

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.

9.2.2 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`.

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`.

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`.

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 and vertically.

9.3 Focus Events

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.

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`.

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.

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.

9.4 Keyboard Events

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.

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.

9.5 Widget Events

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).

9.5.2 FL_ACTIVATE

This widget is now active, due to [activate\(\)](#) being called on it or one of its parents.

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. Please note that although [visible\(\)](#) may still return true for this widget after receiving this event, it is only truly visible if [visible\(\)](#) is true for both it and all of its parents. (You can use [visible_r\(\)](#) to check this).

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.

9.6 Clipboard Events

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\(\)](#).

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.

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. Files are received as a list of full path and file names, separated by newline.

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://).

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](#).

9.7.1 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.

9.7.2 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.

9.7.3 FL_DND_LEAVE

The mouse has moved out of the widget.

9.7.4 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.

9.8 Other events

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.

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.

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\(\)](#)

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::release\(\)](#) (deprecated, see [Fl::grab\(0\)](#))
- [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.

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.

Chapter 10

Adding and Extending Widgets

This chapter describes how to add your own widgets or extend existing widgets in FLTK.

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](#), and [Fl_Window](#)) 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.

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.

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);
```

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:

```
MyClass::handle(int event) {
    ...
    if (change_to_part1) damage(1);
    if (change_to_part2) damage(2);
    if (change_to_part3) damage(4);
}

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 behaviour 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.

Todo Clarify `FL_Widget::test_shortcut()` explanations. `FL_Widget.h` says Internal Use only, but subclassing chapter gives details!

```
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).

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, 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)"
[...]
```

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](#).

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.

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:

```

int 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.

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.

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`

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_left_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.

Chapter 11

Using OpenGL

This chapter discusses using FLTK for your OpenGL applications.

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).

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()`. For double-buffered windows you will need to surround the drawing code with the following code to make sure that both buffers are redrawn:

```
#ifndef MESA
glDrawBuffer(GL_FRONT_AND_BACK);
#endif // !MESA
... draw stuff here ...
#ifndef MESA
glDrawBuffer(GL_BACK);
#endif // !MESA
```

Note:

If you are using the Mesa graphics library, the call to `glDrawBuffer()` is not required and will slow down drawing considerably. The preprocessor instructions shown above will optimize your code based upon the graphics library used.

11.2.1 Defining the Subclass

To define the subclass you just subclass the `F1_Gl_Window` class:

```
class MyWindow : public F1_Gl_Window {
    void draw();
    int handle(int);

public:
    MyWindow(int X, int Y, int W, int H, const char *L)
        : F1_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.)

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 ...
}
```

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 F1::event_x() and F1::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 F1::event_key(), ascii in F1::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 OpenGL window.

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. Conversely, 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 also be useful in this context.

Note

A further coding rule is necessary to properly support retina displays and OpenGL under macOS (see [OpenGL and 'retina' displays](#))

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 `boxtyle` 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` !

11.5 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()`.

11.6 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.

11.7 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...
    csCamera *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...

#ifndef _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 {
#ifndef _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.

11.8 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 Unix/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 with it.

GLEW installation (Unix/Linux and Windows platforms)

GLEW is available as a package for most Linux distributions and in source form at <http://glew.sourceforge.net/>. For the Windows platform, a Visual Studio static library (`glew32.lib`) can be downloaded from the same web site; a MinGW-style static library (`libglew32.a`) can be built from source with the `make` command.

Source-level changes for OpenGL 3:

- Put this in all OpenGL-using source files (instead of `#include <FL/gl.h>`, and before `#include <FL/glut.h>` if you use GLUT):

```
#if defined(__APPLE__)
# include <OpenGL/g13.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.

Changes in the build process

Link with `libGLEW.so` (on Unix/Linux), `libglew32.a` (with MinGW) or `glew32.lib` (with MS Visual Studio); no change is needed on the Mac OS platform.

Chapter 12

Programming with FLUID

This chapter shows how to use the Fast Light User-Interface Designer ("FLUID") to create your GUIs.

Subchapters:

- [What is FLUID?](#)
- [Running FLUID Under UNIX](#)
- [Running FLUID Under Microsoft Windows](#)
- [Compiling .fl Files](#)
- [A Short Tutorial](#)
- [FLUID Reference](#)
- [FLUID Templates](#)
- [Internationalization with FLUID](#)
- [Known Limitations](#)

12.1 What is FLUID?

The Fast Light User Interface Designer, or FLUID, is a graphical editor that is used to produce FLTK source code. FLUID edits and saves its state in `.fl` files. These files are text, and you can (with care) edit them in a text editor, perhaps to get some special effects.

FLUID can "compile" the `.fl` file into a `.cxx` and a `.h` file. The `.cxx` file defines all the objects from the `.fl` file and the `.h` file declares all the global ones. FLUID also supports localization ([Internationalization](#)) of label strings using message files and the GNU gettext or POSIX catgets interfaces.

A simple program can be made by putting all your code (including a `main()` function) into the `.fl` file and thus making the `.cxx` file a single source file to compile. Most programs are more complex than this, so you write other `.cxx` files that call the FLUID functions. These `.cxx` files must `#include` the `.h` file or they can `#include` the `.cxx` file so it still appears to be a single source file.

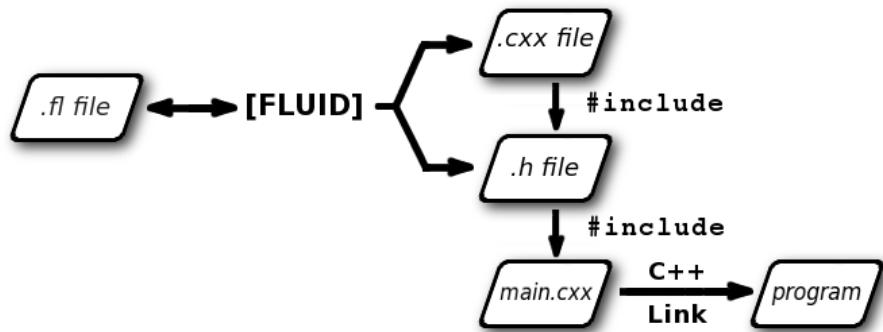


Figure 12.1 FLUID organization

Normally the FLUID file defines one or more functions or classes which output C++ code. Each function defines one or more FLTK windows and all the widgets that go inside those windows.

Widgets created by FLUID are either "named", "complex named" or "unnamed". A named widget has a legal C++ variable identifier as its name (i.e. only alphanumeric and underscore). In this case FLUID defines a global variable or class member that will point at the widget after the function defining it is called. A complex named object has punctuation such as '.' or '->' or any other symbols in its name. In this case FLUID assigns a pointer to the widget to the name, but does not attempt to declare it. This can be used to get the widgets into structures. An unnamed widget has a blank name and no pointer is stored.

Widgets may either call a named callback function that you write in another source file, or you can supply a small piece of C++ source and FLUID will write a private callback function into the .cxx file.

12.2 Running FLUID Under UNIX

To run FLUID under UNIX, type:

```
fluid filename.fl &
```

to edit the .fl file filename.fl. If the file does not exist you will get an error pop-up, but if you dismiss it you will be editing a blank file of that name. You can run FLUID without any name, in which case you will be editing an unnamed blank setup (but you can use save-as to write it to a file).

You can provide any of the standard FLTK switches before the filename:

```
-display host:n.n
-geometry WxH+X+Y
-title windowtitle
-name classname
-iconic
-fg color
-bg color
-bg2 color
-scheme schemename
```

Changing the colors may be useful to see what your interface will look at if the user calls it with the same switches. Similarly, using "-scheme plastic" will show how the interface will look using the "plastic" scheme.

In the current version, if you don't put FLUID into the background with '&' then you will be able to abort FLUID by typing CTRL-C on the terminal. It will exit immediately, losing any changes.

12.3 Running FLUID Under Microsoft Windows

To run FLUID under Windows, double-click on the *FLUID.exe* file. You can also run FLUID from the Command Prompt window. FLUID always runs in the background under Windows.

12.4 Compiling .fl Files

FLUID can also be called as a command-line "compiler" to create the *.cxx* and *.h* file from a *.fl* file. To do this type:

```
fluid -c filename.fl
```

This is the same as the menu 'File/Write Code...'. It will read the *filename.fl* file and write *filename.cxx* and *filename.h*. Any leading directory on *filename.fl* will be stripped, so they are always written to the current directory. If there are any errors reading or writing the files, FLUID will print the error and exit with a non-zero code. You can use the following lines in a makefile to automate the creation of the source and header files:

```
my_panels.h my_panels.hxx: my_panels.fl  
    fluid -c my_panels.fl
```

Most versions of make support rules that cause *.fl* files to be compiled:

```
.SUFFIXES: .fl .cxx .h  
.fl.h .fl.hxx:  
    fluid -c $<
```

If you use

```
fluid -cs filename.fl
```

FLUID will also write the "strings" for internationalization in file 'filename.txt' (menu: 'File/Write Strings...').

Finally there is another option which is useful for program developers who have many *.fl* files and want to upgrade them to the current FLUID version. FLUID will read the *filename.fl* file, save it, and exit immediately. This writes the file with current syntax and options and the current FLTK version in the header of the file. Use

```
fluid -u filename.fl
```

to 'upgrade' *filename.fl*. You may combine this with '-c' or '-cs'.

Note

All these commands overwrite existing files w/o warning. You should particularly take care when running 'fluid -u' since this overwrites the original *.fl* source file.

12.5 A Short Tutorial

FLUID is an amazingly powerful little program. However, this power comes at a price as it is not always obvious how to accomplish seemingly simple tasks with it. This tutorial will show you how to generate a complete user interface class with FLUID that is used for the CubeView program provided with FLTK.

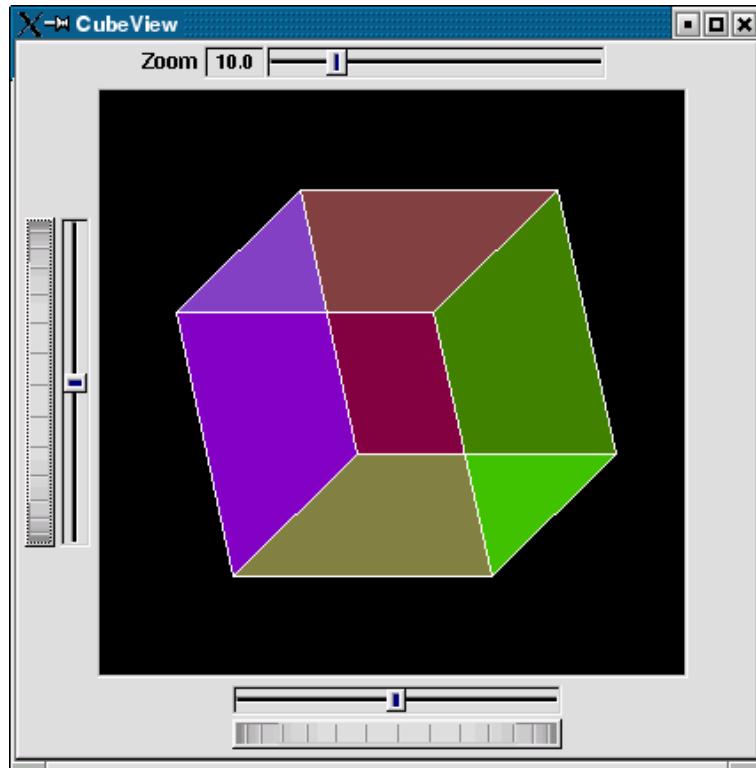


Figure 12.2 CubeView demo

The window is of class `CubeViewUI`, and is completely generated by FLUID, including class member functions. The central display of the cube is a separate subclass of [FL_GI_Window](#) called `CubeView`. `CubeViewUI` manages `CubeView` using callbacks from the various sliders and rollers to manipulate the viewing angle and zoom of `CubeView`.

At the completion of this tutorial you will (hopefully) understand how to:

1. Use FLUID to create a complete user interface class, including constructor and any member functions necessary.
2. Use FLUID to set callback member functions of custom widget classes.
3. Subclass an [FL_GI_Window](#) to suit your purposes.

12.5.1 The CubeView Class

The `CubeView` class is a subclass of [FL_GI_Window](#). It has methods for setting the zoom, the x and y pan, and the rotation angle about the x and y axes.

You can safely skip this section as long as you realize that `CubeView` is a subclass of [FL_GI_Window](#) and will respond to calls from `CubeViewUI`, generated by FLUID.

The CubeView Class Definition

Here is the CubeView class definition, as given by its header file "test/CubeView.h":

```
class CubeView : public Fl_Gl_Window {
public:
    CubeView(int x,int y,int w,int h,const char *l=0);
    // this value determines the scaling factor used to draw the cube.
    double size;
    /* Set the rotation about the vertical (y) axis.

     * This function is called by the horizontal roller in CubeViewUI
     * and the initialize button in CubeViewUI.
     */
    void v_angle(float angle){vAng=angle;};
    // Return the rotation about the vertical (y) axis.
    float v_angle(){return vAng;};
    /* Set the rotation about the horizontal (x) axis.

     * This function is called by the vertical roller in CubeViewUI
     * and the initialize button in CubeViewUI.
     */
    void h_angle(float angle){hAng=angle;};
    // the rotation about the horizontal (x) axis.
    float h_angle(){return hAng;};
    /* Sets the x shift of the cube view camera.

     * This function is called by the slider in CubeViewUI and the
     * initialize button in CubeViewUI.
     */
    void panx(float x){xshift=x;};
    /* Sets the y shift of the cube view camera.

     * This function is called by the slider in CubeViewUI and the
     * initialize button in CubeViewUI.
     */
    void pany(float y){yshift=y;};
    /* The widget class draw() override.
     * The draw() function initialize Gl for another round of
     * drawing then calls specialized functions for drawing each
     * of the entities displayed in the cube view.
     */
    void draw();
private:
    /* Draw the cube boundaries
     * Draw the faces of the cube using the boxv[] vertices, using
     * GL_LINE_LOOP for the faces. The color is #defined by
     * CUBECOLOR.
     */
    void drawCube();

    float vAng,hAng; float xshift,yshift;

    float boxv0[3];float boxv1[3]; float boxv2[3];float boxv3[3];
    float boxv4[3];float boxv5[3]; float boxv6[3];float boxv7[3];
};
```

The CubeView Class Implementation

Here is the CubeView implementation. It is very similar to the "cube" demo included with FLTK.

```
#include "CubeView.h"
#include <math.h>

CubeView::CubeView(int x,int y,int w,int h,const char *l)
    : Fl_Gl_Window(x,y,w,h,l)
{
    vAng = 0.0; hAng=0.0; size=10.0;
    /* The cube definition. These are the vertices of a unit cube
     * centered on the origin.*/
    boxv0[0] = -0.5; boxv0[1] = -0.5; boxv0[2] = -0.5;
    boxv1[0] = 0.5; boxv1[1] = -0.5; boxv1[2] = -0.5;
    boxv2[0] = 0.5; boxv2[1] = 0.5; boxv2[2] = -0.5;
```

```

boxv3[0] = -0.5; boxv3[1] = 0.5; boxv3[2] = -0.5;
boxv4[0] = -0.5; boxv4[1] = -0.5; boxv4[2] = 0.5;
boxv5[0] = 0.5; boxv5[1] = -0.5; boxv5[2] = 0.5;
boxv6[0] = 0.5; boxv6[1] = 0.5; boxv6[2] = 0.5;
boxv7[0] = -0.5; boxv7[1] = 0.5; boxv7[2] = 0.5;
};

// The color used for the edges of the bounding cube.
#define CUBECOLOR 255,255,255,255

void CubeView::drawCube() {
/* Draw a colored cube */
#define ALPHA 0.5
    glShadeModel(GL_FLAT);

    glBegin(GL_QUADS);
        glColor4f(0.0, 0.0, 1.0, ALPHA);
        glVertex3fv(boxv0);
        glVertex3fv(boxv1);
        glVertex3fv(boxv2);
        glVertex3fv(boxv3);

        glColor4f(1.0, 1.0, 0.0, ALPHA);
        glVertex3fv(boxv0);
        glVertex3fv(boxv4);
        glVertex3fv(boxv5);
        glVertex3fv(boxv1);

        glColor4f(0.0, 1.0, 1.0, ALPHA);
        glVertex3fv(boxv2);
        glVertex3fv(boxv6);
        glVertex3fv(boxv7);
        glVertex3fv(boxv3);

        glColor4f(1.0, 0.0, 0.0, ALPHA);
        glVertex3fv(boxv4);
        glVertex3fv(boxv5);
        glVertex3fv(boxv6);
        glVertex3fv(boxv7);

        glColor4f(1.0, 0.0, 1.0, ALPHA);
        glVertex3fv(boxv0);
        glVertex3fv(boxv3);
        glVertex3fv(boxv7);
        glVertex3fv(boxv4);

        glColor4f(0.0, 1.0, 0.0, ALPHA);
        glVertex3fv(boxv1);
        glVertex3fv(boxv5);
        glVertex3fv(boxv6);
        glVertex3fv(boxv2);
    glEnd();

    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_LINES);
        glVertex3fv(boxv0);
        glVertex3fv(boxv1);

        glVertex3fv(boxv1);
        glVertex3fv(boxv2);

        glVertex3fv(boxv2);
        glVertex3fv(boxv3);

        glVertex3fv(boxv3);
        glVertex3fv(boxv0);

        glVertex3fv(boxv4);
        glVertex3fv(boxv5);

        glVertex3fv(boxv5);
        glVertex3fv(boxv6);

        glVertex3fv(boxv6);
        glVertex3fv(boxv7);

        glVertex3fv(boxv7);
        glVertex3fv(boxv4);

        glVertex3fv(boxv0);
        glVertex3fv(boxv4);

        glVertex3fv(boxv1);
        glVertex3fv(boxv5);

        glVertex3fv(boxv2);
        glVertex3fv(boxv6);

        glVertex3fv(boxv3);
        glVertex3fv(boxv7);
    glEnd();
}

```

```

glVertex3fv(boxv3);
glVertex3fv(boxv7);
glEnd();
};//drawCube

void CubeView::draw() {
    if (!valid()) {
        glLoadIdentity(); glViewport(0,0,w(),h());
        glOrtho(-10,10,-10,10,-20000,10000); glEnable(GL_BLEND);
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    }

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPushMatrix(); glTranslatef(xshift, yshift, 0);
    glRotatef(hAng,0,1,0); glRotatef(vAng,1,0,0);
    glScalef(float(size),float(size),float(size)); drawCube();
    glPopMatrix();
}

```

12.5.2 The CubeViewUI Class

We will completely construct a window to display and control the CubeView defined in the previous section using FLUID.

Defining the CubeViewUI Class

Once you have started FLUID, the first step in defining a class is to create a new class within FLUID using the **New->Code->Class** menu item. Name the class "CubeViewUI" and leave the subclass blank. We do not need any inheritance for this window. You should see the new class declaration in the FLUID browser window.

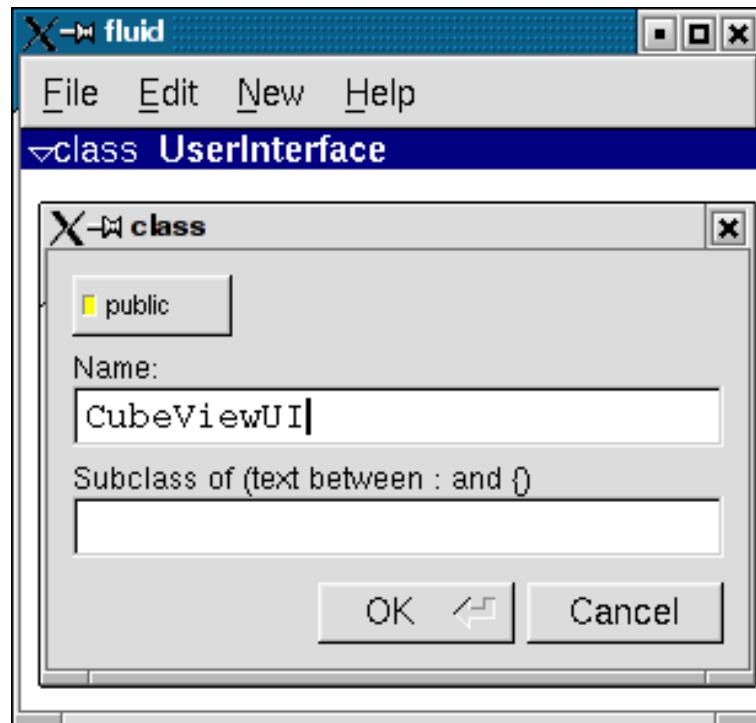


Figure 12.3 FLUID file for CubeView

Adding the Class Constructor

Click on the CubeViewUI class in the FLUID window and add a new method by selecting **New->Code->Function/Method**. The name of the function will also be CubeViewUI. FLUID will understand that this will be the constructor for the class and will generate the appropriate code. Make sure you declare the constructor public.

Then add a window to the CubeViewUI class. Highlight the name of the constructor in the FLUID browser window and click on **New->Group->Window**. In a similar manner add the following to the CubeViewUI constructor:

- A horizontal roller named `hrot`
- A vertical roller named `vrot`
- A horizontal slider named `xpan`
- A vertical slider named `ypan`
- A horizontal value slider named `zoom`

None of these additions need be public. And they shouldn't be unless you plan to expose them as part of the interface for CubeViewUI.

When you are finished you should have something like this:

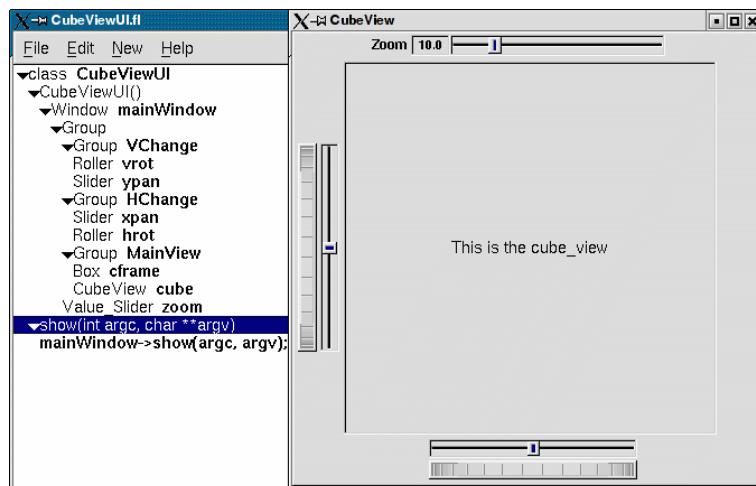


Figure 12.4 FLUID window containing CubeView demo

We will talk about the `show()` method that is highlighted shortly.

Adding the CubeView Widget

What we have is nice, but does little to show our cube. We have already defined the CubeView class and we would like to show it within the CubeViewUI.

The CubeView class inherits the `FI_GI_Window` class, which is created in the same way as an `FI_Box` widget. Use **New->Other->Box** to add a square box to the main window. This will be no ordinary box, however.

The Box properties window will appear. The key to letting CubeViewUI display CubeView is to enter CubeView in the **Class:** text entry box. This tells FLUID that it is not an `FI_Box`, but a similar widget with the same constructor.

In the **Extra Code:** field enter `#include "CubeView.h"`

This `#include` is important, as we have just included `CubeView` as a member of `CubeViewUI`, so any public `CubeView` methods are now available to `CubeViewUI`.

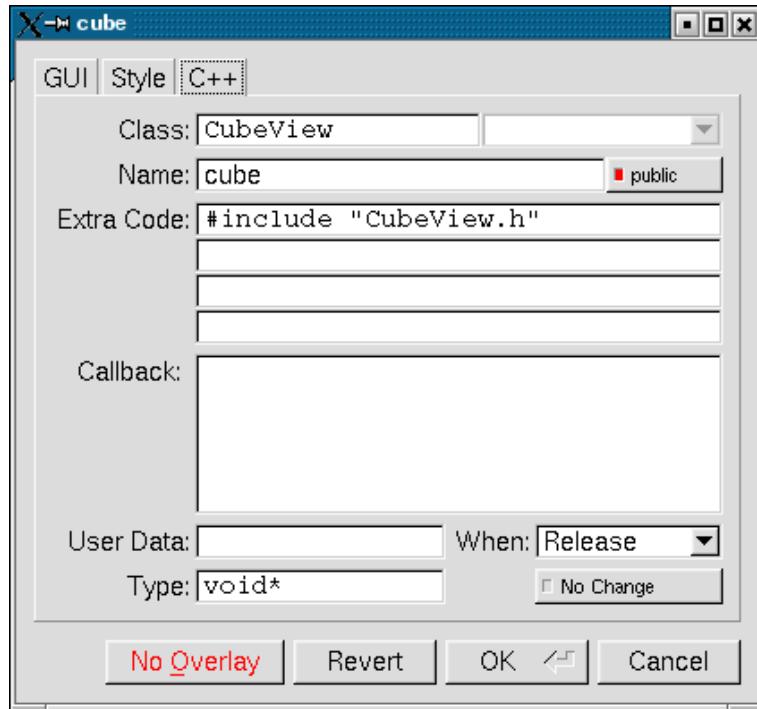


Figure 12.5 CubeView methods

Defining the Callbacks

Each of the widgets we defined before adding `CubeView` can have callbacks that call `CubeView` methods. You can call an external function or put a short amount of code in the **Callback** field of the widget panel. For example, the callback for the `ypan` slider is:

```
cube->pany(((F1_Slider *)o)->value());
cube->redraw();
```

We call `cube->redraw()` after changing the value to update the `CubeView` window. `CubeView` could easily be modified to do this, but it is nice to keep this exposed. In the case where you may want to do more than one view change only redrawing once saves a lot of time.

There is no reason to wait until after you have added `CubeView` to enter these callbacks. FLUID assumes you are smart enough not to refer to members or functions that don't exist.

Adding a Class Method

You can add class methods within FLUID that have nothing to do with the GUI. As an example add a `show` function so that `CubeViewUI` can actually appear on the screen.

Make sure the top level `CubeViewUI` is selected and select **New->Code->Function/Method**. Just use the name `show()`. We don't need a return value here, and since we will not be adding any widgets to this method FLUID will assign it a return type of `void`.

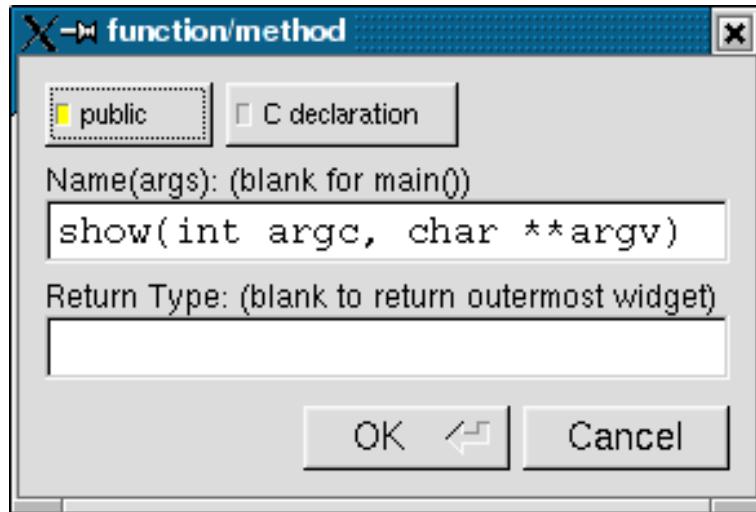


Figure 12.6 CubeView constructor

Once the new method has been added, highlight its name and select **New->Code->Code**. Enter the method's code in the code window.

12.5.3 Adding Constructor Initialization Code

If you need to add code to initialize a class, for example setting initial values of the horizontal and vertical angles in the CubeView, you can simply highlight the constructor and select **New->Code->Code**. Add any required code.

12.5.4 Generating the Code

Now that we have completely defined the CubeViewUI, we have to generate the code. There is one last trick to ensure this all works. Open the preferences dialog from **Edit->Preferences**.

At the bottom of the preferences dialog box is the key: "**Include Header from Code**". Select that option and set your desired file extensions and you are in business. You can include the CubeViewUI.h (or whatever extension you prefer) as you would any other C++ class.

12.6 FLUID Reference

The following sections describe each of the windows in FLUID.

12.6.1 The Widget Browser

The main window shows a menu bar and a scrolling browser of all the defined widgets. The name of the .f1 file being edited is shown in the window title.

The widgets are stored in a hierarchy. You can open and close a level by clicking the "triangle" at the left of a widget. The leftmost widgets are the *parents*, and all the widgets listed below them are their *children*. Parents don't have to have any children.

The top level of the hierarchy is composed of *functions* and *classes*. Each of these will produce a single C++ public function or class in the output `.cxx` file. Calling the function or instantiating the class will create all of the child widgets.

The second level of the hierarchy contains the *windows*. Each of these produces an instance of class `FI_Window`.

Below that are either *widgets* (subclasses of `FI_Widget`) or *groups* of widgets (including other groups). Plain groups are for layout, navigation, and resize purposes. *Tab groups* provide the well-known file-card tab interface.

Widgets are shown in the browser by either their *name* (such as "main_panel" in the example), or by their *type* and *label* (such as "Button "the green"").

You *select* widgets by clicking on their names, which highlights them (you can also select widgets from any displayed window). You can select many widgets by dragging the mouse across them, or by using Shift+Click to toggle them on and off. To select no widgets, click in the blank area under the last widget. Note that hidden children may be selected even when there is no visual indication of this.

You *open* widgets by double-clicking on them, or (to open several widgets you have picked) by typing the F1 key. A control panel will appear so you can change the widget(s).

12.6.2 Menu Items

The menu bar at the top is duplicated as a pop-up menu on any displayed window. The shortcuts for all the menu items work in any window. The menu items are:

File/Open... (Ctrl+o)

Discards the current editing session and reads in a different `.fl` file. You are asked for confirmation if you have changed the current file.

FLUID can also read `.fd` files produced by the Forms and XForms "fdesign" programs. It is best to File/Merge them instead of opening them. FLUID does not understand everything in a `.fd` file, and will print a warning message on the controlling terminal for all data it does not understand. You will probably need to edit the resulting setup to fix these errors. Be careful not to save the file without changing the name, as FLUID will write over the `.fd` file with its own format, which fdesign cannot read!

File/Insert... (Ctrl+i)

Inserts the contents of another `.fl` file, without changing the name of the current `.fl` file. All the functions (even if they have the same names as the current ones) are added, and you will have to use cut/paste to put the widgets where you want.

File/Save (Ctrl+s)

Writes the current data to the `.fl` file. If the file is unnamed then FLUID will ask for a filename.

File/Save As... (Ctrl+Shift+S)

Asks for a new filename and saves the file.

File/Write Code (Ctrl+Shift+C)

"Compiles" the data into a .cxx and .h file. These are exactly the same as the files you get when you run FLUID with the -c switch.

The output file names are the same as the .fl file, with the leading directory and trailing ".fl" stripped, and ".h" or ".cxx" appended.

File/Write Strings (Ctrl+Shift+W)

Writes a message file for all of the text labels defined in the current file.

The output file name is the same as the .fl file, with the leading directory and trailing ".fl" stripped, and ".txt", ".po", or ".msg" appended depending on the [Internationalization Mode](#).

File/Quit (Ctrl+q)

Exits FLUID. You are asked for confirmation if you have changed the current file.

Edit/Undo (Ctrl+z)

This isn't implemented yet. You should do save often so you can recover from any mistakes you make.

Edit/Cut (Ctrl+x)

Deletes the selected widgets and all of their children. These are saved to a "clipboard" file and can be pasted back into any FLUID window.

Edit/Copy (Ctrl+c)

Copies the selected widgets and all of their children to the "clipboard" file.

Edit/Paste (Ctrl+v)

Pastes the widgets from the clipboard file.

If the widget is a window, it is added to whatever function is selected, or contained in the current selection.

If the widget is a normal widget, it is added to whatever window or group is selected. If none is, it is added to the window or group that is the parent of the current selection.

To avoid confusion, it is best to select exactly one widget before doing a paste.

Cut/paste is the only way to change the parent of a widget.

Edit>Select All (Ctrl+a)

Selects all widgets in the same group as the current selection.

If they are all selected already then this selects all widgets in that group's parent. Repeatedly typing Ctrl+a will select larger and larger groups of widgets until everything is selected.

Edit/Open... (F1 or double click)

Displays the current widget in the attributes panel. If the widget is a window and it is not visible then the window is shown instead.

Edit/Sort

Sorts the selected widgets into left to right, top to bottom order. You need to do this to make navigation keys in FLTK work correctly. You may then fine-tune the sorting with "Earlier" and "Later". This does not affect the positions of windows or functions.

Edit/Earlier (F2)

Moves all of the selected widgets one earlier in order among the children of their parent (if possible). This will affect navigation order, and if the widgets overlap it will affect how they draw, as the later widget is drawn on top of the earlier one. You can also use this to reorder functions, classes, and windows within functions.

Edit/Later (F3)

Moves all of the selected widgets one later in order among the children of their parent (if possible).

Edit/Group (F7)

Creates a new [FI_Group](#) and make all the currently selected widgets children of it.

Edit/Ungroup (F8)

Deletes the parent group if all the children of a group are selected.

Edit/Overlays on/off (Ctrl+Shift+O)

Toggles the display of the red overlays off, without changing the selection. This makes it easier to see box borders and how the layout looks. The overlays will be forced back on if you change the selection.

Edit/Project Settings... (Alt+p)

Displays the project settings panel.

Under the "Output" tab you control the extensions or names of the files that are generated by FLUID. If you check the "Include Header from Code" button the code file will include the header file automatically.

Under the "Internationalization" tab are the [internationalization](#) options, described later in this chapter.

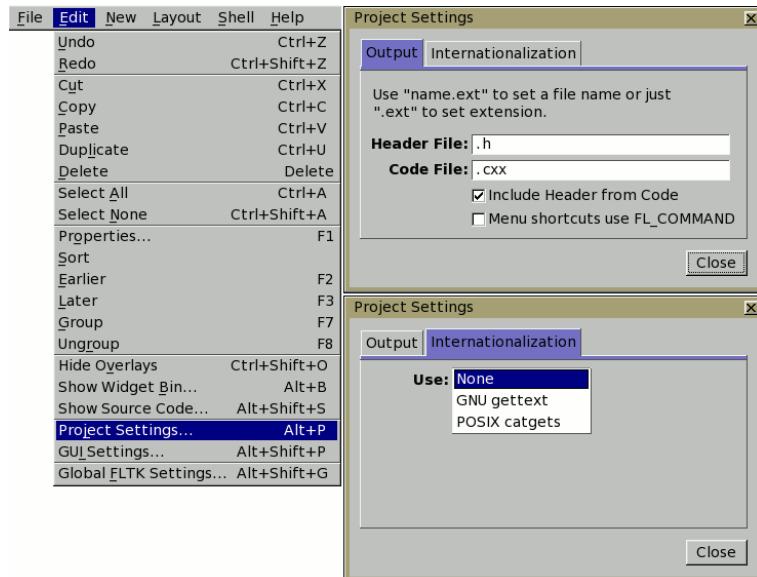


Figure 12.7 FLUID Project Settings Window

Edit/GUI Settings... (Shift+Alt+p)

Displays the GUI Settings panel, used to control the user interface settings.

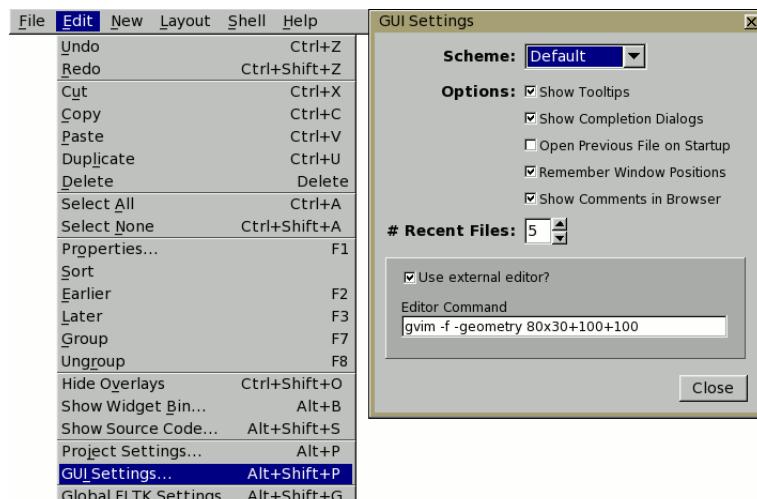


Figure 12.8 FLUID GUI Settings Window

Edit/Global FLTK Settings... (Shift+Alt+g)

Displays the FLTK Global Settings ("Preferences") panel, used to control fluid's user specific and/or system wide settings.

Tooltips provide descriptions of each option.

At the lower-right, "User Settings" causes changes to only affect the current user, "System Settings" causes changes to be applied to all users on the current machine.

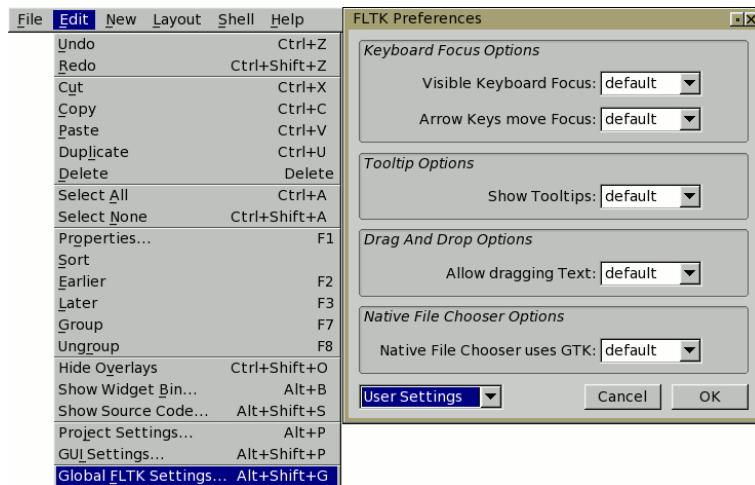


Figure 12.9 FLUID Global Settings Window

New/Code/Function

Creates a new C function. You will be asked for a name for the function. This name should be a legal C++ function template, without the return type. You can pass arguments which can be referred to by code you type into the individual widgets.

If the function contains any unnamed windows, it will be declared as returning an [Fl_Window](#) pointer. The unnamed window will be returned from it (more than one unnamed window is useless). If the function contains only named windows, it will be declared as returning nothing (`void`).

It is possible to make the `.cxx` output be a self-contained program that can be compiled and executed. This is done by deleting the function name so `main(argc, argv)` is used. The function will call `show()` on all the windows it creates and then call [`Fl::run\(\)`](#). This can also be used to test resize behavior or other parts of the user interface.

You can change the function name by double-clicking on the function.

New/Window

Creates a new [Fl_Window](#) widget. The window is added to the currently selected function, or to the function containing the currently selected item. The window will appear, sized to 100x100. You can resize it to whatever size you require.

The widget panel will also appear and is described later in this chapter.

New/...

All other items on the New menu are subclasses of [Fl_Widget](#). Creating them will add them to the currently selected group or window, or the group or window containing the currently selected widget. The initial dimensions and position are chosen by copying the current widget, if possible.

When you create the widget you will get the widget's control panel, which is described later in this chapter.

Layout/Align/...

Align all selected widgets to the first widget in the selection.

Layout/Space Evenly/...

Space all selected widgets evenly inside the selected space. Widgets will be sorted from first to last.

Layout/Make Same Size/...

Make all selected widgets the same size as the first selected widget.

Layout/Center in Group/...

Center all selected widgets relative to their parent widget

Layout/Grid and Size Settings... (Ctrl+g)

Displays the grid settings panel.

This panel controls the grid that all widgets snap to when you move and resize them, and for the "snap" which is how far a widget has to be dragged from its original position to actually change.

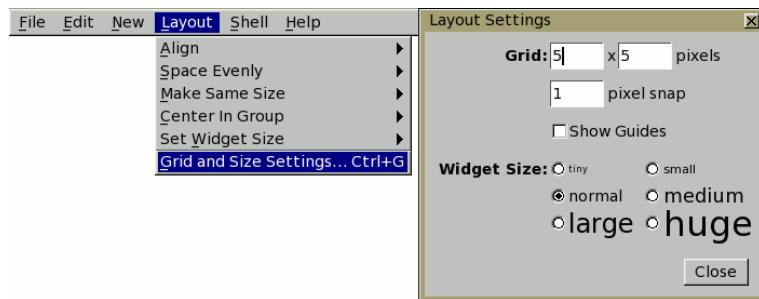


Figure 12.10 FLUID Layout/Grid Settings Window

Shell/Execute Command... (Alt+x)

Displays the shell command panel. The shell command is commonly used to run a 'make' script to compile the FLTK output.

Shell/Execute Again (Alt+g)

Run the shell command again.

Help/About FLUID

Pops up a panel showing the version of FLUID.

Help/On FLUID

Shows this chapter of the manual.

Help/Manual

Shows the contents page of the manual

12.6.3 The Widget Panel

When you double-click on a widget or a set of widgets you will get the "widget attribute panel".

When you change attributes using this panel, the changes are reflected immediately in the window. It is useful to hit the "no overlay" button (or type Ctrl+Shift+O) to hide the red overlay so you can see the widgets more accurately, especially when setting the box type.

If you have several widgets selected, they may have different values for the fields. In this case the value for *one* of the widgets is shown. But if you change this value, *all* of the selected widgets are changed to the new value.

Hitting "OK" makes the changes permanent. Selecting a different widget also makes the changes permanent. FLUID checks for simple syntax errors such as mismatched parenthesis in any code before saving any text.

"Revert" or "Cancel" put everything back to when you last brought up the panel or hit OK. However in the current version of FLUID, changes to "visible" attributes (such as the color, label, box) are not undone by revert or cancel. Changes to code like the callbacks are undone, however.

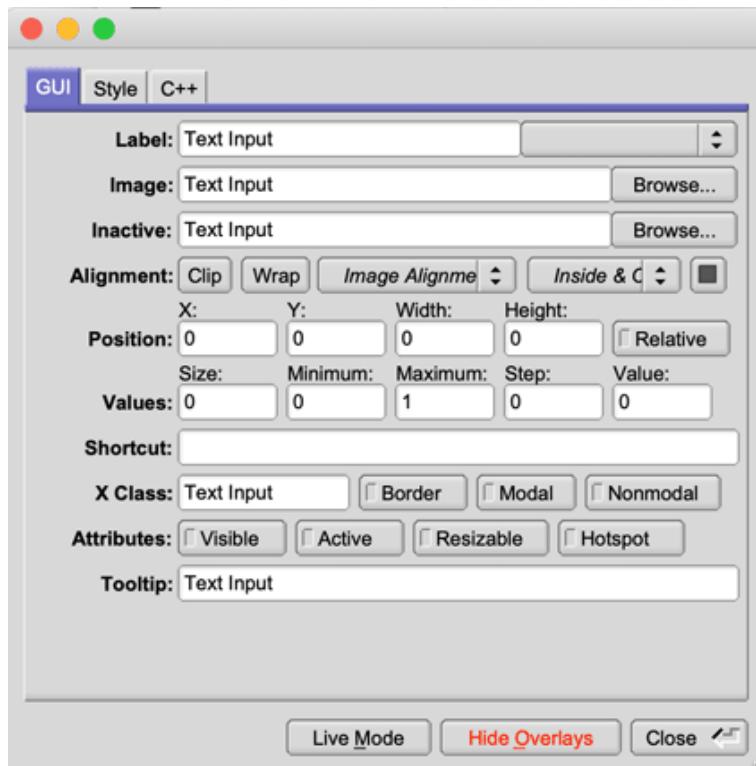


Figure 12.11 The FLUID widget GUI attributes

12.7 GUI Attributes

Not all fields in the Widget attributes dialog will be visible

for all types of widgets.

Label (text field)

String to print next to or inside the button. You can put newlines into the string to make multiple lines. The easiest way is by typing Ctrl+j.

Symbols can be added to the label using the at sign ("@").

Label (pull down menu)

How to draw the label. Normal, shadowed, engraved, and embossed change the appearance of the text.

Image

The active image for the widget. Click on the **Browse...** button to pick an image file using the file chooser.

Inactive

The inactive image for the widget. Click on the **Browse...** button to pick an image file using the file chooser.

Alignment (buttons)

Where to draw the label. The arrows put it on that side of the widget, you can combine them to put it in the corner. The "box" button puts the label inside the widget, rather than outside.

The **clip** button clips the label to the widget box, the **wrap** button wraps any text in the label, and the **text image** button puts the text over the image instead of under the image.

Position (text fields)

The position fields show the current position and size of the widget box. Enter new values to move and/or resize a widget.

Values (text fields)

The values and limits of the current widget. Depending on the type of widget, some or all of these fields may be inactive.

Shortcut

The shortcut key to activate the widget. Click on the shortcut button and press any key sequence to set the shortcut.

Attributes (buttons)

The **Visible** button controls whether the widget is visible (on) or hidden (off) initially. Don't change this for windows or for the immediate children of a Tabs group.

The **Active** button controls whether the widget is activated (on) or deactivated (off) initially. Most widgets appear greyed out when deactivated.

The **Resizable** button controls whether the window is resizeable. In addition all the size changes of a window or group will go "into" the resizable child. If you have a large data display surrounded by buttons, you probably want that data area to be resizable. You can get more complex behavior by making invisible boxes the resizable widget, or by using hierarchies of groups. Unfortunately the only way to test it is to compile the program. Resizing the FLUID window is *not* the same as what will happen in the user program.

The **Hotspot** button causes the parent window to be positioned with that widget centered on the mouse. This position is determined *when the FLUID function is called*, so you should call it immediately before showing the window. If you want the window to hide and then reappear at a new position, you should have your program set the hotspot itself just before `show()`.

The **Border** button turns the window manager border on or off. On most window managers you will have to close the window and reopen it to see the effect.

X Class (text field)

The string typed into here is passed to the X window manager as the class. This can change the icon or window decorations. On most (all?) window managers you will have to close the window and reopen it to see the effect.

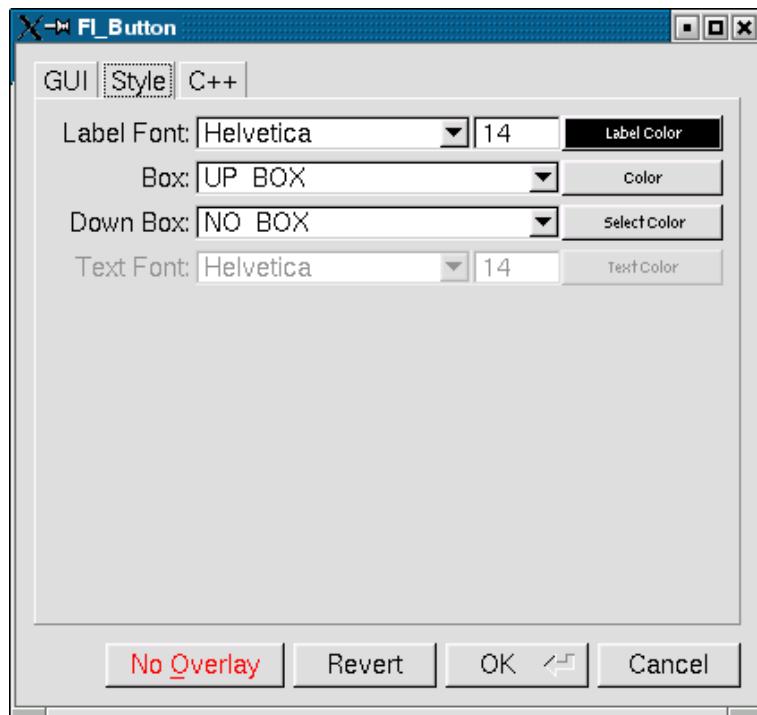


Figure 12.12 The FLUID widget Style attributes

12.7.1 Style Attributes

Label Font (pulldown menu)

Font to draw the label in. Ignored by symbols, bitmaps, and pixmaps. Your program can change the actual font used by these "slots" in case you want some font other than the 16 provided.

Label Size (pulldown menu)

Pixel size (height) for the font to draw the label in. Ignored by symbols, bitmaps, and pixmaps. To see the result without dismissing the panel, type the new number and then Tab.

Label Color (button)

Color to draw the label. Ignored by pixmaps (bitmaps, however, do use this color as the foreground color).

Box (pulldown menu)

The boxtyle to draw as a background for the widget.

Many widgets will work, and draw faster, with a "frame" instead of a "box". A frame does not draw the colored interior, leaving whatever was already there visible. Be careful, as FLUID may draw this ok but the real program may leave unwanted stuff inside the widget.

If a window is filled with child widgets, you can speed up redrawing by changing the window's box type to "`← NO_BOX`". FLUID will display a checkerboard for any areas that are not colored in by boxes. Note that this checkerboard is not drawn by the resulting program. Instead random garbage will be displayed.

Down Box (pulldown menu)

The boxtyle to draw when a button is pressed or for some parts of other widgets like scrollbars and valiators.

Color (button)

The color to draw the box with.

Select Color (button)

Some widgets will use this color for certain parts. FLUID does not always show the result of this: this is the color buttons draw in when pushed down, and the color of input fields when they have the focus.

Text Font, Size, and Color

Some widgets display text, such as input fields, pull-down menus, and browsers.

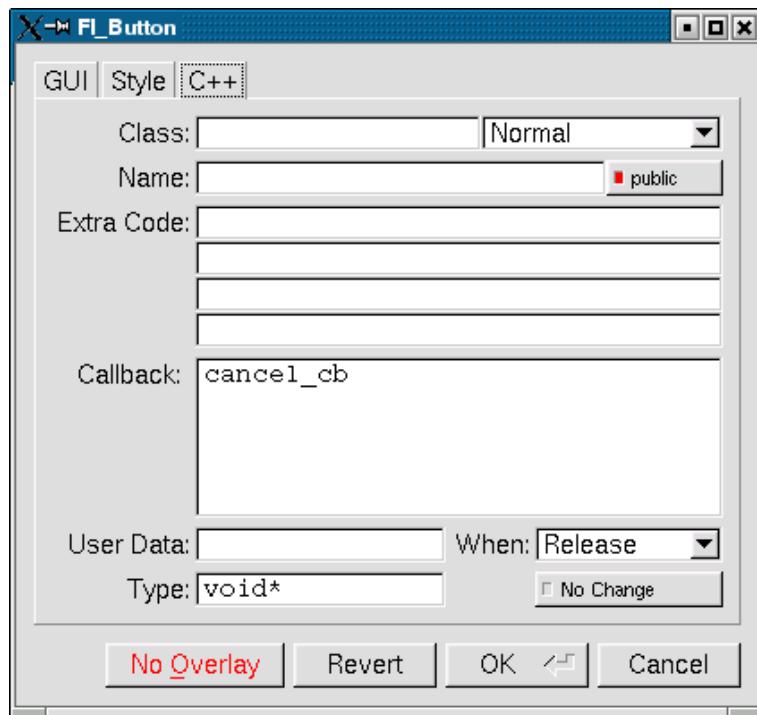


Figure 12.13 The FLUID widget C++ attributes

12.7.2 C++ Attributes

Class

This is how you use your own subclasses of [FL_Widget](#). Whatever identifier you type in here will be the class that is instantiated.

In addition, no `#include` header file is put in the `.h` file. You must provide a `#include` line as the first line of the "Extra Code" which declares your subclass.

The class must be similar to the class you are spoofing. It does not have to be a subclass. It is sometimes useful to change this to another FLTK class. For windows you can select either `Single` or `Double` in the drop-down box right to the "Class:" field to get a normal window ([FL_Window](#)) or a double-buffered window ([FL_Double_Window](#)), respectively.

Type (upper-right pulldown menu)

Some classes have subtypes that modify their appearance or behavior. You pick the subtype off of this menu.

Name (text field)

Name of a variable to declare, and to store a pointer to this widget into. This variable will be of type "<class>*". If the name is blank then no variable is created.

You can name several widgets with "name[0]", "name[1]", "name[2]", etc. This will cause FLUID to declare an array of pointers. The array is big enough that the highest number found can be stored. All widgets in the array must be the same type.

Public (button)

Controls whether the widget is publicly accessible. When embedding widgets in a C++ class, this controls whether the widget is `public` or `private` in the class. Otherwise it controls whether the widget is declared `static` or `global` (`extern`).

Extra Code (text fields)

These four fields let you type in literal lines of code to dump into the `.h` or `.cxx` files.

If the text starts with a `#` or the word `extern` then FLUID thinks this is an "include" line, and it is written to the `.h` file. If the same include line occurs several times then only one copy is written.

All other lines are "code" lines. The current widget is pointed to by the local variable `o`. The window being constructed is pointed to by the local variable `w`. You can also access any arguments passed to the function here, and any named widgets that are before this one.

FLUID will check for matching parenthesis, braces, and quotes, but does not do much other error checking. Be careful here, as it may be hard to figure out what widget is producing an error in the compiler. If you need more than four lines you probably should call a function in your own `.cxx` code.

Callback (text field)

This can either be the name of a function, or a small snippet of code. If you enter anything other than letters, numbers, and the underscore then FLUID treats it as code.

A name refers to a function in your own code. It must be declared as `void name(<class>*, void*)`.

A code snippet is inserted into a static function in the `.cxx` output file. The function prototype is `void name(class *o, void *v)` so that you can refer to the widget as `o` and the `user_data()` as `v`. FLUID will check for matching parenthesis, braces, and quotes, but does not do much other error checking. Be careful here, as it may be hard to figure out what widget is producing an error in the compiler.

If the callback is blank then no callback is set.

User Data (text field)

This is a value for the `user_data()` of the widget. If blank the default value of zero is used. This can be any piece of C code that can be cast to a `void` pointer.

Type (text field)

The `void*` in the callback function prototypes is replaced with this. You may want to use `long` for old XForms code. Be warned that anything other than `void*` is not guaranteed to work! However on most architectures other pointer types are ok, and `long` is usually ok, too.

When (pulldown menu)

When to do the callback. This can be **Never**, **Changed**, **Release**, or **Enter Key**. The value of **Enter Key** is only useful for text input fields.

There are other rare but useful values for the `when()` field that are not in the menu. You should use the extra code fields to put these values in.

No Change (button)

The **No Change** button means the callback is done on the matching event even if the data is not changed.

12.8 Selecting and Moving Widgets

Double-clicking a window name in the browser will display it, if not displayed yet. From this display you can select widgets, sets of widgets, and move or resize them. To close a window either double-click it or type `ESC`.

To select a widget, click it. To select several widgets drag a rectangle around them. Holding down shift will toggle the selection of the widgets instead.

You cannot pick hidden widgets. You also cannot choose some widgets if they are completely overlapped by later widgets. Use the browser to select these widgets.

The selected widgets are shown with a red "overlay" line around them. You can move the widgets by dragging this box. Or you can resize them by dragging the outer edges and corners. Hold down the Alt key while dragging the mouse to defeat the snap-to-grid effect for fine positioning.

If there is a tab box displayed you can change which child is visible by clicking on the file tabs. The child you pick is selected.

The arrow, tab, and shift+tab keys "navigate" the selection. Left, right, tab, or shift+tab move to the next or previous widgets in the hierarchy. Hit the right arrow enough and you will select every widget in the window. Up/down widgets move to the previous/next widgets that overlap horizontally. If the navigation does not seem to work you probably need to "Sort" the widgets. This is important if you have input fields, as FLTK uses the same rules when using arrow keys to move between input fields.

To "open" a widget, double click it. To open several widgets select them and then type F1 or pick "Edit/Open" off the pop-up menu.

Type Ctrl+o to temporarily toggle the overlay off without changing the selection, so you can see the widget borders.

You can resize the window by using the window manager border controls. FLTK will attempt to round the window size to the nearest multiple of the grid size and makes it big enough to contain all the widgets (it does this using illegal X methods, so it is possible it will barf with some window managers!). Notice that the actual window in your program may not be resizable, and if it is, the effect on child widgets may be different.

The panel for the window (which you get by double-clicking it) is almost identical to the panel for any other [FL_Widget](#). There are three extra items:

12.9 Image Labels

The *contents* of the image files in the **Image** and **Inactive** text fields are written to the `.cxx` file. If many widgets share the same image then only one copy is written. Since the image data is embedded in the generated source code, you need only distribute the C++ code and not the image files themselves.

However, the *filenames* are stored in the `.fl` file so you will need the image files as well to read the `.fl` file. Filenames are relative to the location of the `.fl` file and not necessarily the current directory. We recommend you either put the images in the same directory as the `.fl` file, or use absolute path names.

Notes for All Image Types

FLUID runs using the default visual of your X server. This may be 8 bits, which will give you dithered images. You may get better results in your actual program by adding the code "Fl::visual(FL_RGB)" to your code right before the first window is displayed.

All widgets with the same image on them share the same code and source X pixmap. Thus once you have put an image on a widget, it is nearly free to put the same image on many other widgets.

If you edit an image at the same time you are using it in FLUID, the only way to convince FLUID to read the image file again is to remove the image from all widgets that are using it or re-load the `.fl` file.

Don't rely on how FLTK crops images that are outside the widget, as this may change in future versions! The cropping of inside labels will probably be unchanged.

To more accurately place images, make a new "box" widget and put the image in that as the label.

XBM (X Bitmap) Files

FLUID reads X bitmap files which use C source code to define a bitmap. Sometimes they are stored with the ".h" or ".bm" extension rather than the standard ".xbm" extension.

FLUID writes code to construct an [Fl_Bitmap](#) image and use it to label the widget. The '1' bits in the bitmap are drawn using the label color of the widget. You can change this color in the FLUID widget attributes panel. The '0' bits are transparent.

The program "bitmap" on the X distribution does an adequate job of editing bitmaps.

XPM (X Pixmap) Files

FLUID reads X pixmap files as used by the `libxpm` library. These files use C source code to define a pixmap. The filenames usually have the ".xpm" extension.

FLUID writes code to construct an [Fl_Pixmap](#) image and use it to label the widget. The label color of the widget is ignored, even for 2-color images that could be a bitmap. XPM files can mark a single color as being transparent, and FLTK uses this information to generate a transparency mask for the image.

We have not found any good editors for small iconic pictures. For pixmaps we have used [XPaint](#) and the KDE icon editor.

BMP Files

FLUID reads Windows BMP image files which are often used in Windows applications for icons. FLUID converts BMP files into (modified) XPM format and uses an [Fl_BMP_Image](#) image to label the widget. Transparency is handled the same as for XPM files. All image data is uncompressed when written to the source file, so the code may be much bigger than the .bmp file.

GIF Files

FLUID reads GIF image files which are often used in HTML documents to make icons. FLUID converts GIF files into (modified) XPM format and uses an [Fl_GIF_Image](#) image to label the widget. Transparency is handled the same as for XPM files. All image data is uncompressed when written to the source file, so the code may be much bigger than the .gif file. Only the first image of an animated GIF file is used.

JPEG Files

If FLTK is compiled with JPEG support, FLUID can read JPEG image files which are often used for digital photos. FLUID uses an [Fl_JPEG_Image](#) image to label the widget, and writes uncompressed RGB or grayscale data to the source file.

PNG (Portable Network Graphics) Files

If FLTK is compiled with PNG support, FLUID can read PNG image files which are often used in HTML documents. FLUID uses a [Fl_PNG_Image](#) image to label the widget, and writes uncompressed RGB or grayscale data to the source file. PNG images can provide a full alpha channel for partial transparency, and FLTK supports this as best as possible on each platform.

12.10 FLUID Templates

Fluid can store a number of project templates. Project templates are great for storing often used boilerplate code for fast access. A common use would be projects with readily prepared copyright messages.

A sample template for FLTK projects is included with Fluid.

Choose "File > New From Template..." to create a new project based on a template file. In the template dialog, select one of the existing templates. All occurrences of the word "@INSTANCE@" in the template are replaced with the text in the "Instance" field. To create the new project click "New".

To add your current project as a new template, choose "File > Save As Template...", fill in a name, and click "Save".

To delete a template, open the template dialog using "New from Template" or "Save As Template", then select any existing template, and click "Delete Template".

12.11 Internationalization with FLUID

FLUID supports internationalization (I18N for short) of label strings used by widgets. The preferences window (**Ctrl+p**) provides access to the I18N options.

12.11.1 I18N Methods

FLUID supports three methods of I18N: use none, use GNU gettext, and use POSIX catgets. The "use none" method is the default and just passes the label strings as-is to the widget constructors.

The "GNU gettext" method uses GNU gettext (or a similar text-based I18N library) to retrieve a localized string before calling the widget constructor.

The "POSIX catgets" method uses the POSIX catgets function to retrieve a numbered message from a message catalog before calling the widget constructor.

12.11.2 Using GNU gettext for I18N

FLUID's code support for GNU gettext is limited to calling a function or macro to retrieve the localized label; you still need to call `setlocale()` and `textdomain()` or `bindtextdomain()` to select the appropriate language and message file.

To use GNU gettext for I18N, open the preferences window and choose "GNU gettext" from the **Use:** chooser. Two new input fields will then appear to control the include file and function/macro name to use when retrieving the localized label strings.

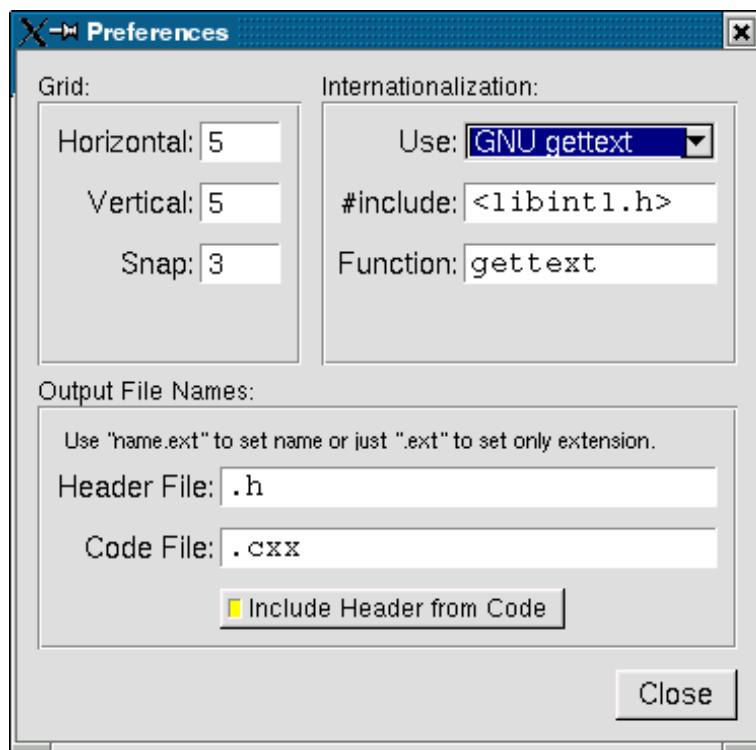


Figure 12.14 Internationalization using GNU gettext

The **#include** field controls the header file to include for I18N; by default this is `<libintl.h>`, the standard I18N file for GNU gettext.

The **Function:** field controls the function (or macro) that will retrieve the localized message; by default the `gettext` function will be called.

12.11.3 Using POSIX catgets for I18N

FLUID's code support for POSIX catgets allows you to use a global message file for all interfaces or a file specific to each .fl file; you still need to call `setlocale()` to select the appropriate language.

To use POSIX catgets for I18N, open the preferences window and choose "POSIX catgets" from the **Use:** chooser. Three new input fields will then appear to control the include file, catalog file, and set number for retrieving the localized label strings.

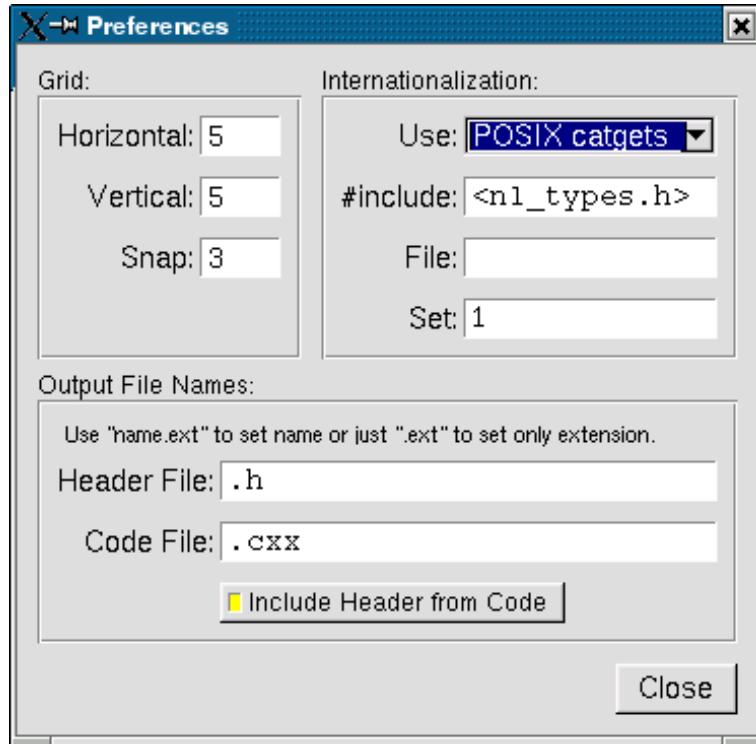


Figure 12.15 Internationalization using POSIX catgets

The **#include** field controls the header file to include for I18N; by default this is `<nl_types.h>`, the standard I18N file for POSIX catgets.

The **File:** field controls the name of the catalog file variable to use when retrieving localized messages; by default the file field is empty which forces a local (static) catalog file to be used for all of the windows defined in your .fl file.

The **Set:** field controls the set number in the catalog file. The default set is 1 and rarely needs to be changed.

12.12 Known Limitations

Declaration Blocks can be used to temporarily block out already designed code using `#if 0` and `#endif` type construction. This will effectively avoid compilation of blocks of code. However, static code and data generated by this segment (menu items, images, include statements, etc.) will still be generated and likely cause compile-time warnings.

Chapter 13

Advanced FLTK

This chapter explains advanced programming and design topics that will help you to get the most out of FLTK.

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/thread.h` and `test/thread.cxx`.

FLTK has been used with a variety of thread interfaces, so if the simple example shown in `test/thread.cxx` does not cover your needs, you might want to select a third-party library that provides the features you require.

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 `-enable-threads` set during the `configure` process. IDE-based versions of FLTK are automatically compiled with the locking mechanism incorporated if possible. Since version 1.3, the `configure` script that builds the FLTK library also sets `-enable-threads` by default.

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` thread messages

You can send messages from worker threads to the `main()` thread using `Fl::awake(void* message)`. If using this thread message interface, your `main()` might look 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 loop and process thread messages */
    while (Fl::wait() > 0) {
        if ((next_message = Fl::thread_message()) != NULL) {
            /* process your data, update widgets, etc. */
            ...
        }
    }

    /* terminate any pending worker threads */
    ... stop threads ...

    return 0;
}
```

Your worker threads can send messages to the `main()` thread using `Fl::awake(void* message)`:

```
void *msg;           // "msg" is a pointer to your message
Fl::awake(msg);     // send "msg" to main thread
```

A message can be anything you like. The `main()` thread can retrieve the message by calling `Fl::thread_message()`.

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 mechanisms used to deliver `Fl::awake(void* message)` and `Fl::awake(Fl_Awake_Handler cb, void* userdata)` events to the `main()` thread can interact in unexpected ways on some platforms. Therefore, for reliable operation, it is advised that a program use either `Fl::awake(Fl_Awake_Handler cb, void* userdata)` or `Fl::awake(void* message)`, but that they never be intermixed. Calling `Fl::awake()` with no parameters should be safe in either case.

If you have to choose between using the `Fl::awake(void* message)` and `Fl::awake(Fl_Awake_Handler cb, void* userdata)` mechanisms and don't know which to choose, then try the `Fl::awake(Fl_Awake_Handler cb, void* userdata)` method first as it tends to be more powerful in general.

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?

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.)
- Don't start or cancel timers from a worker thread
- 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()`.

Chapter 14

Unicode and UTF-8 Support

This chapter explains how FLTK handles international text via Unicode and UTF-8.

Unicode support was added to FLTK starting with version 1.3.0 and is still incomplete but mostly functional. This chapter is Work in Progress, reflecting the current state of Unicode support.

14.1 About Unicode, ISO 10646 and UTF-8

The summary of Unicode, ISO 10646 and UTF-8 given below is deliberately brief and provides just enough information for the rest of this chapter.

For further information, please see:

- <https://unicode.org>
- <https://iso.org>
- <https://en.wikipedia.org/wiki/Unicode>
- <https://www.cl.cam.ac.uk/~mgk25/unicode.html>
- <https://tools.ietf.org/html/rfc3629>

The Unicode Standard

The Unicode Standard was originally developed by a consortium of mainly US computer manufacturers and developers of multi-lingual software. It has now become a defacto standard for character encoding and is supported by most of the major computing companies in the world.

Before Unicode, many different systems, on different platforms, had been developed for encoding characters for different languages, but no single encoding could satisfy all languages. Unicode provides access to over 130,000 characters used in all the major languages written today, and is independent of platform and language.

Unicode also provides higher-level concepts needed for text processing and typographic publishing systems, such as algorithms for sorting and comparing text, composite character and text rendering, right-to-left and bi-directional text handling.

Note

There are currently no plans to add this extra functionality to FLTK.

ISO 10646

The International Organisation for Standardization (ISO) had also been trying to develop a single unified character set. Although both ISO and the Unicode Consortium continue to publish their own standards, they have agreed to coordinate their work so that specific versions of the Unicode and ISO 10646 standards are compatible with each other.

The international standard ISO 10646 defines the **Universal Character Set** (UCS) which contains the characters required for almost all known languages. The standard also defines three different implementation levels specifying how these characters can be combined.

Note

There are currently no plans for handling the different implementation levels or the combining characters in FLTK.

In UCS, characters have a unique numerical code and an official name, and are usually shown using 'U+' and the code in hexadecimal, e.g. U+0041 is the "Latin capital letter A". The UCS characters U+0000 to U+007F correspond to US-ASCII, and U+0000 to U+00FF correspond to ISO 8859-1 (Latin1).

ISO 10646 was originally designed to handle a 31-bit character set from U+00000000 to U+7FFFFFFF, but the current idea is that 21 bits will be sufficient for all future needs, giving characters up to U+10FFFF. The complete character set is sub-divided into *planes*. *Plane 0*, also known as the **Basic Multilingual Plane** (BMP), ranges from U+0000 to U+FFFD and consists of the most commonly used characters from previous encoding standards. Other planes contain characters for specialist applications.

Todo FLTK 1.3 and later supports the full Unicode range (21 bits), but there are a few exceptions, for instance binary shortcut values in menus ([FL_Shortcut](#)) can only be used with characters from the BMP (16 bits). This may be extended in a future FLTK version.

The UCS also defines various methods of encoding characters as a sequence of bytes. UCS-2 encodes Unicode characters into two bytes, which is wasteful if you are only dealing with ASCII or Latin1 text, and insufficient if you need characters above U+00FFFF. UCS-4 uses four bytes, which lets it handle higher characters, but this is even more wasteful for ASCII or Latin1.

UTF-8

The Unicode standard defines various UCS Transformation Formats (UTF). UTF-16 and UTF-32 are based on units of two and four bytes. UCS characters requiring more than 16 bits are encoded using "surrogate pairs" in UTF-16.

UTF-8 encodes all Unicode characters into variable length sequences of bytes. Unicode characters in the 7-bit ASCII range map to the same value and are represented as a single byte, making the transformation to Unicode quick and easy.

All UCS characters above U+007F are encoded as a sequence of several bytes. The top bits of the first byte are set to show the length of the byte sequence, and subsequent bytes are always in the range 0x80 to 0xBF. This combination provides some level of synchronisation and error detection.

Unicode range	Byte sequences
U+00000000 – U+0000007F	0xxxxxx
U+00000080 – U+000007FF	110xxxxx 10xxxxxx
U+00000800 – U+0000FFFF	1110xxxx 10xxxxxx 10xxxxxx
U+00010000 – U+001FFFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
U+00200000 – U+03FFFFFF	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
U+04000000 – U+7FFFFFFF	1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

Note

This table contains theoretical values outside the valid Unicode range (U+000000 – U+10FFFF). Such values can only be returned by conversion functions for illegal input values (see [Illegal Unicode and UTF-8 Sequences](#)).

Moving from ASCII encoding to Unicode will allow all new FLTK applications to be easily internationalized and used all over the world. By choosing UTF-8 encoding, FLTK remains largely source-code compatible to previous iterations of the library.

14.2 Unicode in FLTK

Todo Work through the code and this documentation to harmonize the [**OksiD**] and [**fltk2**] functions.

FLTK will be entirely converted to Unicode using UTF-8 encoding. If a different encoding is required by the underlying operating system, FLTK will convert the string as needed.

It is important to note that the initial implementation of Unicode and UTF-8 in FLTK involves three important areas:

- provision of Unicode character tables and some simple related functions;
- conversion of `char*` variables and function parameters from single byte per character representation to UTF-8 variable length sequences;
- modifications to the display font interface to accept general Unicode character or UCS code numbers instead of just ASCII or Latin1 characters.

The current implementation of Unicode / UTF-8 in FLTK will impose the following limitations:

- An implementation note in the [**OksiD**] code says that all functions are LIMITED to 24 bit Unicode values, but also says that only 16 bits are really used under linux and win32. [[Can we verify this?](#)]
- The [**fltk2**] `fl_utf8encode()` and `fl_utf8decode()` functions are designed to handle Unicode characters in the range U+000000 to U+10FFFF inclusive, which covers all UTF-16 characters, as specified in RFC 3629. *Note that the user must first convert UTF-16 surrogate pairs to UCS.*
- FLTK will only handle single characters, so composed characters consisting of a base character and floating accent characters will be treated as multiple characters.
- FLTK will only compare or sort strings on a byte by byte basis and not on a general Unicode character basis.
- FLTK will not handle right-to-left or bi-directional text.

Todo Verify 16/24 bit Unicode limit for different character sets? OksiD's code appears limited to 16-bit whereas the FLTK2 code appears to handle a wider set. What about illegal characters? See comments in `fl_utf8fromwc()` and `fl_utf8toUtf16()`.

14.3 Illegal Unicode and UTF-8 Sequences

Three pre-processor variables are defined in the source code [1] that determine how `fl_utf8decode()` handles illegal UTF-8 sequences:

- if `ERRORS_TO_CP1252` is set to 1 (the default), `fl_utf8decode()` will assume that a byte sequence starting with a byte in the range 0x80 to 0x9f represents a Microsoft CP1252 character, and will return the value of an equivalent UCS character. Otherwise, it will be processed as an illegal byte value as described below.
- if `STRICT RFC3629` is set to 1 (not the default!) then UTF-8 sequences that correspond to illegal UCS values are treated as errors. Illegal UCS values include those above U+10FFFF, or corresponding to UTF-16 surrogate pairs. Illegal byte values are handled as described below.
- if `ERRORS_TO_ISO8859_1` is set to 1 (the default), the illegal byte value is returned unchanged, otherwise 0xFFFFD, the Unicode REPLACEMENT CHARACTER, is returned instead.

[1] Since FLTK 1.3.4 you may set these three pre-processor variables on your compile command line with -D"variable=value" (value: 0 or 1) to avoid editing the source code.

`fl_utf8encode()` is less strict, and only generates the UTF-8 sequence for 0xFFFFD, the Unicode REPLACEMENT CHARACTER, if it is asked to encode a UCS value above U+10FFFF.

Many of the [**fltk2**] functions below use `fl_utf8decode()` and `fl_utf8encode()` in their own implementation, and are therefore somewhat protected from bad UTF-8 sequences.

The [**OksiD**] `fl_utf8len()` function assumes that the byte it is passed is the first byte in a UTF-8 sequence, and returns the length of the sequence. Trailing bytes in a UTF-8 sequence will return -1.

- **WARNING:** `fl_utf8len()` can not distinguish between single bytes representing Microsoft CP1252 characters 0x80-0x9f and those forming part of a valid UTF-8 sequence. You are strongly advised not to use `fl_utf8len()` in your own code unless you know that the byte sequence contains only valid UTF-8 sequences.
- **WARNING:** Some of the [**OksiD**] functions below still use `fl_utf8len()` in their implementations. These may need further validation.

Please see the individual function description for further details about error handling and return values.

14.4 FLTK Unicode and UTF-8 Functions

This section provides a brief overview of the functions. For more details, consult the main text for each function via its link.

`int fl_utf8locale() FLTK2`

`fl_utf8locale()` returns true if the "locale" seems to indicate that UTF-8 encoding is used.

It is highly recommended that you change your system so this does return true!

```
int fl_utf8test(const char *src, unsigned len) FLTK2
```

`fl_utf8test()` examines the first `len` bytes of `src`. It returns 0 if there are any illegal UTF-8 sequences; 1 if `src` contains plain ASCII or if `len` is zero; or 2, 3 or 4 to indicate the range of Unicode characters found.

```
int fl_utf_nb_char(const unsigned char *buf, int len) OksiD
```

Returns the number of UTF-8 characters in the first `len` bytes of `buf`.

```
int fl_unichar_to_utf8_size(Fl_Unichar)
int fl_utf8bytes(unsigned ucs)
```

Returns the number of bytes needed to encode `ucs` in UTF-8.

```
int fl_utf8len(char c) OksiD
```

If `c` is a valid first byte of a UTF-8 encoded character sequence, `fl_utf8len()` will return the number of bytes in that sequence. It returns -1 if `c` is not a valid first byte.

```
unsigned int fl_nonspacing(unsigned int ucs) OksiD
```

Returns true if `ucs` is a non-spacing character.

```
const char* fl_utf8back(const char *p, const char *start, const char *end) FLTK2
const char* fl_utf8fwd(const char *p, const char *start, const char *end) FLTK2
```

If `p` already points to the start of a UTF-8 character sequence, these functions will return `p`. Otherwise `fl_utf8back()` searches backwards from `p` and `fl_utf8fwd()` searches forwards from `p`, within the `start` and `end` limits, looking for the start of a UTF-8 character.

```
unsigned int fl_utf8decode(const char *p, const char *end, int *len) FLTK2
int fl_utf8encode(unsigned ucs, char *buf) FLTK2
```

`fl_utf8decode()` attempts to decode the UTF-8 character that starts at `p` and may not extend past `end`. It returns the Unicode value, and the length of the UTF-8 character sequence is returned via the `len` argument. `fl_utf8encode()` writes the UTF-8 encoding of `ucs` into `buf` and returns the number of bytes in the sequence. See the main documentation for the treatment of illegal Unicode and UTF-8 sequences.

unsigned int `fl_utf8froma(char *dst, unsigned dstlen, const char *src, unsigned srclen)` **FLTK2**
 unsigned int `fl_utf8toa(const char *src, unsigned srclen, char *dst, unsigned dstlen)` **FLTK2**

`fl_utf8froma()` converts a character string containing single bytes per character (i.e. ASCII or ISO-8859-1) into UTF-8. If the `src` string contains only ASCII characters, the return value will be the same as `srclen`.

`fl_utf8toa()` converts a string containing UTF-8 characters into single byte characters. UTF-8 characters that do not correspond to ASCII or ISO-8859-1 characters below 0xFF are replaced with '?'.

Both functions return the number of bytes that would be written, not counting the null terminator. `dstlen` provides a means of limiting the number of bytes written, so setting `dstlen` to zero is a means of measuring how much storage would be needed before doing the real conversion.

char* `fl_utf2mbcs(const char *src)` **OksiD**

converts a UTF-8 string to a local multi-byte character string. **[More info required here!]**

unsigned int `fl_utf8fromwc(char *dst, unsigned dstlen, const wchar_t *src, unsigned srclen)` **FLTK2**
 unsigned int `fl_utf8towc(const char *src, unsigned srclen, wchar_t *dst, unsigned dstlen)` **FLTK2**
 unsigned int `fl_utf8toUtf16(const char *src, unsigned srclen, unsigned short *dst, unsigned dstlen)` **FLTK2**

These routines convert between UTF-8 and `wchar_t` or "wide character" strings. The difficulty lies in the fact that `sizeof(wchar_t)` is 2 on Windows and 4 on Linux and most other systems. Therefore some "wide characters" on Windows may be represented as "surrogate pairs" of more than one `wchar_t`.

`fl_utf8fromwc()` converts from a "wide character" string to UTF-8. Note that `srclen` is the number of `wchar_t` elements in the source string and on Windows this might be larger than the number of characters. `dstlen` specifies the maximum number of **bytes** to copy, including the null terminator.

`fl_utf8towc()` converts a UTF-8 string into a "wide character" string. Note that on Windows, some "wide characters" might result in "surrogate pairs" and therefore the return value might be more than the number of characters. `dstlen` specifies the maximum number of `wchar_t` elements to copy, including a zero terminating element. **[Is this all worded correctly?]**

`fl_utf8toUtf16()` converts a UTF-8 string into a "wide character" string using UTF-16 encoding to handle the "surrogate pairs" on Windows. `dstlen` specifies the maximum number of `wchar_t` elements to copy, including a zero terminating element. **[Is this all worded correctly?]**

These routines all return the number of elements that would be required for a full conversion of the `src` string, including the zero terminator. Therefore setting `dstlen` to zero is a way of measuring how much storage would be needed before doing the real conversion.

```
unsigned int fl_utf8from_mb(char *dst, unsigned dstlen, const char *src, unsigned srclen) FLTK2
unsigned int fl_utf8to_mb(const char *src, unsigned srclen, char *dst, unsigned dstlen) FLTK2
```

These functions convert between UTF-8 and the locale-specific multi-byte encodings used on some systems for filenames, etc. If `fl_utf8locale()` returns true, these functions don't do anything useful. [Is this all worded correctly?]

```
int fl_tolower(unsigned int ucs) OksiD
int fl_toupper(unsigned int ucs) OksiD
int fl_utf_tolower(const unsigned char *str, int len, char *buf) OksiD
int fl_utf_toupper(const unsigned char *str, int len, char *buf) OksiD
```

`fl_tolower()` and `fl_toupper()` convert a single Unicode character from upper to lower case, and vice versa. `fl_utf_tolower()` and `fl_utf_toupper()` convert a string of bytes, some of which may be multi-byte UTF-8 encodings of Unicode characters, from upper to lower case, and vice versa.

Warning: to be safe, `buf` length must be at least `3*len` [for 16-bit Unicode]

```
int fl_utf_strcasecmp(const char *s1, const char *s2) OksiD
int fl_utf_strncasecmp(const char *s1, const char *s2, int n) OksiD
```

`fl_utf_strcasecmp()` is a UTF-8 aware string comparison function that converts the strings to lower case Unicode as part of the comparison. `fl_utf_strncasecmp()` only compares the first `n` characters [bytes?]

14.5 FLTK Unicode Versions of System Calls

- `int fl_access(const char* f, int mode)` **OksiD**
- `int fl_chmod(const char* f, int mode)` **OksiD**
- `int fl_execvp(const char* file, char* const* argv)` **OksiD**
- `FILE* fl_fopen(const char* f, const char* mode)` **OksiD**
- `char* fl_getcwd(char* buf, int maxlen)` **OksiD**
- `char* fl_getenv(const char* name)` **OksiD**
- `char fl_make_path(const char* path)` - returns char ? **OksiD**
- `void fl_make_path_for_file(const char* path)` **OksiD**
- `int fl_mkdir(const char* f, int mode)` **OksiD**
- `int fl_open(const char* f, int o, ...)` **OksiD**
- `int fl_rename(const char* f, const char* t)` **OksiD**
- `int fl_rmdir(const char* f)` **OksiD**
- `int fl_stat(const char* path, struct stat* buffer)` **OksiD**

- int [fl_system\(const char* f\)](#) **OksiD**
- int [fl_unlink\(const char* f\)](#) **OksiD**

TODO:

- more doc on unicode, add links
- write something about filename encoding on OS X...
- explain the fl_utf8_... commands
- explain issues with [Fl_Preferences](#)

Chapter 15

FLTK 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.

15.1 Version Numbers

The FLTK version number is stored in a number of compile-time constants:

- `FL_MAJOR_VERSION` - The major release number, currently 1
- `FL_MINOR_VERSION` - The minor release number, currently 4
- `FL_PATCH_VERSION` - The patch release number, currently 0
- `FL_VERSION` - [Deprecated] A combined floating-point version number for the major, minor, and patch release numbers, currently 1.0400
- `FL_API_VERSION` - A combined integer version number for the major, minor, and patch release numbers, currently 10400 (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, currently 10400 (default)

Note

The ABI version (`FL_ABI_VERSION`) is usually constant throughout one major/minor release version, for instance 10300 if `FL_API_VERSION` is 10304. Hence the ABI is constant if only the patch version is changed. You can change this with `configure` or `CMake` though if you want the latest enhancements (called "ABI features", see `CHANGES`).

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.

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.

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

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; use `FL_KP + n` for number `n`.
- 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.

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 and X11, or FL_META on MacOS X.
- FL_CONTROL - An alias for FL_META on Windows and X11, 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_BUTTONS - Any mouse button is pushed.
- `FL_BUTTON(n)` - Mouse button `n` (where `n > 0`) is pushed.

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.

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.

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.

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.

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

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.

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.

Chapter 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.

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.

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 `glutIdleFunc()` 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").

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.

16.4 class Fl_Glut_Window

16.4.1 Class Hierarchy

```

Fl_Gl_Window
|
+---Fl_Glut_Window

```

16.4.2 Include Files

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

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()`.

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.)

16.4.5 Methods

`FI_Glut_Window::FI_Glut_Window(int x, int y, int w, int h, const char *title = 0)`
`FI_Glut_Window::FI_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 FI_Glut_Window::~FI_Glut_Window()`

Destroys the GLUT window.

`void FI_Glut_Window::make_current()`

Switches all drawing functions to the GLUT window.

Chapter 17

Forms Compatibility

This appendix describes the Forms compatibility included with FLTK.

Warning: The Forms compatibility is deprecated and no longer maintained in FLTK1, and is likely to be removed completely after the next official release.

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>`.

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.

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 "`boxtyp`e" 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_DBCLICK` (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

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	<code>Fl::event_x_root()</code>
MOUSE_Y	<code>Fl::event_y_root()</code>
LEFTSHIFTKEY,RIGHTSHIFTKEY	<code>Fl::event_shift()</code>
CAPSLOCKKEY	<code>Fl::event_capslock()</code>
LEFTCTRLKEY,RIGHTCTRLKEY	<code>Fl::event_ctrl()</code>
LEFTALTKEY,RIGHTALTKEY	<code>Fl::event_alt()</code>
MOUSE1,RIGHTMOUSE	<code>Fl::event_state()</code>
MOUSE2,MIDDLEMOUSE	<code>Fl::event_state()</code>
MOUSE3,LEFTMOUSE	<code>Fl::event_state()</code>

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*");
```


Chapter 18

Operating System Issues

This appendix describes the operating system specific interfaces in FLTK:

- [Accessing the OS Interfaces](#)
- [The UNIX \(X11\) Interface](#)
- [The Windows Interface](#)
- [The Apple OS X Interface](#)

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.

18.2 The UNIX (X11) Interface

The UNIX interface provides access to the X Window System state information and data structures.

18.2.1 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 () .

```
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.

```
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_ask()`, for instance, it will not return until the modal function returns.

18.2.2 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;
extern Window fl_window;
extern GC fl_gc;
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*](#).

18.2.3 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_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);
```

18.2.4 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(fл_display, RootWindow(fл_display, fл_screen),
                                   vis->visual, AllocNone);
    }
    Fл_X::make_xid(this, visual, colormap);
}
```

`Fл_X *Fл_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 Fл_X::make_xid(Fl_Window*, XVisualInfo* = fл_visual, Colormap = fл_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 `fл_window` and `fл_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::i(this)->region`. If `NULL` then you should redraw the entire window. The undocumented function `fл_clip_region(X->Region)` will initialize the FLTK clip stack with a region or `NULL` for no clipping. You must set `region` to `NULL` afterwards as `fл_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::i(this)->region);
    Fl_X::i(this)->region = 0;
    if (damage() != 2) {... draw things into backing store ...}
    ... copy backing store to 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(f1_display, my pixmap);
        my pixmap = 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.

18.2.5 Setting the Icon of a Window

FLTK currently supports setting a window's icon **before** it is shown using the `Fl_Window::icon()` method.

`void Fl_Window::icon(const void *)`

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(f1_display, DefaultRootWindow(f1_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"

f1_open_display();                                // needed if display has not been previously opened

Pixmap p, mask;

XpmCreatePixmapFromData(f1_display, DefaultRootWindow(f1_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.

18.2.6 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`).

18.2.7 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.

18.3 The Windows Interface

The Windows interface provides access to the Windows GDI state information and data structures.

18.3.1 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.

18.3.2 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 `PostThreadMessage()` in order to request, rather than force your application to terminate.

18.3.3 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.

18.3.4 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;
extern HWND fl_window;
extern HDC fl_gc;
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.

18.3.5 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.

18.3.6 Setting the Icon of a Window

FLTK currently supports setting a window's icon *before* it is shown using the [Fl_Window::icon\(\)](#) method.

```
void Fl_Window::icon(const void *)
```

Sets 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.

18.3.7 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.

18.3.8 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, `F1::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.
- `F1_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 `F1::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

18.4 The Apple OS X Interface

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 `F1::event_key()` and the keystroke translation in `F1::event_text()`. For example, typing Option-Y on a Mac US keyboard will set `FL_ALT` in `F1::event_state()`, set `F1::event_key()` to 'y' and return the Yen symbol in `F1::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();
#ifndef __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
    //
#ifndef __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.

18.4.1 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.

18.4.2 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 global variable `fl_gc` (of type `CGContextRef`) is the appropriate Quartz 2D drawing environment. Include `FL/platform.H` to declare the `fl_gc` variable.

18.4.3 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.

18.4.4 OpenGL and 'retina' displays

It is possible to have OpenGL produce graphics at the high pixel resolution allowed by the so-called 'retina' displays present on recent Apple hardware. For this, call

```
Fl::use_high_res_GL(1);
```

before any [Fl_Gl_Window](#) is shown. Also, adapt your [Fl_Gl_Window::draw\(\)](#) and [Fl_Gl_Window::draw_overlay\(\)](#) methods replacing

```
glViewport(0, 0, w(), h());
```

by

```
glViewport(0, 0, pixel_w(), pixel_h());
```

making use of the [Fl_Gl_Window::pixel_w\(\)](#) and [Fl_Gl_Window::pixel_h\(\)](#) methods that return the width and height of the GL scene in pixels: if the [Fl_Gl_Window](#) is mapped on a retina display, these methods return twice as much as reported by [Fl_Widget::w\(\)](#) and [Fl_Widget::h\(\)](#); if it's mapped on a regular display, they return the same values as [w\(\)](#) and [h\(\)](#). These methods dynamically change their values if the window is moved into/out from a retina display. If [Fl::use_high_res_GL\(1\)](#) is not called, all [Fl_Gl_Window](#)'s are drawn at low resolution. These methods are useful on all platforms because [Fl_Gl_Window::w\(\)](#) and [Fl_Gl_Window::h\(\)](#) don't return, on HighDPI displays, the quantities in pixels necessary to OpenGL functions .

The [Fl_Gl_Window::pixels_per_unit\(\)](#) method is 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.

18.4.5 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.

18.4.6 Mac File System Specifics

Resource Forks

FLTK does not access the resource fork of an application. However, a minimal resource fork must be created for OS X applications. Starting with OS X 10.6, resource forks are no longer needed.

Caution (OS X 10.2 and older):

When using UNIX commands to copy or move executables, OS X will NOT copy any resource forks! For copying and moving use CpMac and MvMac respectively. For creating a tar archive, all executables need to be stripped from their Resource Fork before packing, e.g. "DeRez fluid > fluid.r". After unpacking the Resource Fork needs to be reattached, e.g. "Rez fluid.r -o fluid".

It is advisable to use the Finder for moving and copying and Mac archiving tools like Sit for distribution as they will handle the Resource Fork correctly.

Mac File Paths

FLTK uses UTF-8-encoded UNIX-style filenames and paths.

See also

[Mac OS X-specific symbols](#)

Chapter 19

Migrating Code from FLTK 1.3 to 1.4

This appendix describes the differences between the FLTK 1.3.x and FLTK 1.4.x functions and classes.

19.1 Migrating from FLTK 1.0 or 1.1 to 1.4

If you want to migrate your code from FLTK 1.0 or 1.1 to FLTK 1.4, then you should first consult the relevant appendices in FLTK 1.3 documentation online or by downloading the FLTK 1.3 documentation. See <https://www.fltk.org/doc-1.3/index.html> and/or <https://www.fltk.org/software.php>, respectively.

19.2 Minor Changes in Header Files

(to be documented)

19.3 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 preferences files has changed slightly. Please see [Fl_Preferences::Fl_Preferences\(Root, const char*, const char*\)](#) for details.

If you want to retain user preferences you may want to move the preferences file from its old location to the new location as documented in [Fl_Preferences::Fl_Preferences\(Root, const char*, const char*\)](#).

Chapter 20

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 boxtyle 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 boxtyle, position, size, and
label string.
\param[in] t boxtyle
\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.

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)

20.1 Non-ASCII Characters

Doxygen understands many HTML quoting characters like
`"`, `ü`, `ç`, `Ç`, 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.

20.2 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 I - Developer Information
```

at its top.

This section is

```
\section development_structure Document Structure
```

The following section is

```
\section development_links Creating Links
```

20.3 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 FLTK Library 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](https://www.fltk.org/) creates a standard HTML link

20.4 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 "html" tags work<br>
```

```
\par
```

Use a single line ending with
 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
 for complicated paragraph titles.

20.5 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 gives the navigation bar at the bottom of this page:

```
\htmlonly
<hr>





```

Chapter 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>).

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 library" 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

Chapter 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.

22.1 Example Applications: Overview

adjuster	animated	arc	ask	bitmap	blocks
boxtyle	browser	button	buttons	cairo_test	checkers
clock	colrowser	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	unitests	utf8	valuators	windowfocus	

22.1.1 adjuster

`adjuster` shows a nifty little widget for quickly setting values in a great range.

22.1.2 animated

`animated` shows a window with an animated square that shows drawing with transparency (alpha channel).

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.

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... .

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.

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.

22.1.7 boxtyle

`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.

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.

22.1.9 button

The `button` test is a simple demo of push-buttons and callbacks.

22.1.10 buttons

`buttons` shows a sample of FLTK button types.

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 if FLTK was configured with Cairo support.

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.

22.1.13 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.

22.1.14 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*.

22.1.15 color_chooser

The `color_chooser` gives a short demo of FLTK's palette based color chooser and of the RGB based color wheel.

22.1.16 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.

22.1.17 CubeView

`CubeView` shows how to create a UI containing OpenGL with Fluid.

22.1.18 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).

22.1.19 curve

`curve` draws a nice Bezier curve into a custom widget. The `points` option for splines is not supported on all platforms.

22.1.20 demo

This tool allows quick access to all programs in the `test` directory. `demo` is based on the visuals of the IrixGL demo program. The menu tree can be changed by editing `test/demo.menu`.

22.1.21 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 'examples' directory is a clipboard watching application that continuously displays the textual or graphical content of the system clipboard (a.k.a pasteboard on Mac OS X) exercising `Fl::paste()`.

22.1.22 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.

22.1.23 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.

22.1.24 fast_slow

`fast_slow` shows how an application can use the `Fl_Widget::when()` setting to receive different kinds of callbacks.

22.1.25 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.

22.1.26 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.

22.1.27 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.

22.1.28 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.

22.1.29 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.

22.1.30 gl_overlay

`gl_overlay` shows OpenGL overlay plane rendering. If no hardware overlay plane is available, FLTK will simulate it for you.

22.1.31 glpuzzle

The `glpuzzle` test shows how most Glut source code compiles easily under FLTK.

22.1.32 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.

22.1.33 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.

22.1.34 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).

22.1.35 iconize

`iconize` demonstrates the effect of the window functions `hide()`, `iconize()`, and `show()`.

22.1.36 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).

22.1.37 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.

22.1.38 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\(\)](#).

22.1.39 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.

22.1.40 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.

22.1.41 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`.

22.1.42 line_style

Advanced line drawing can be tested with `line_style`. Not all platforms support all line styles.

22.1.43 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.

22.1.44 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.

22.1.45 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.

22.1.46 message

`message` pops up a few of FLTK's standard message boxes.

22.1.47 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.

22.1.48 native-filechooser

The `native-filechooser` program invokes the platform specific file chooser, if available (see [Fl_Native_File_Chooser](#) widget).

22.1.49 navigation

`navigation` demonstrates how the text cursor moves from text field to text field when using the arrow keys, tab, and shift-tab.

22.1.50 offscreen

`offscreen` shows how to draw into an offscreen image and display the offscreen image in the program window.

22.1.51 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.

22.1.52 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.

22.1.53 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.

22.1.54 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.

22.1.55 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.

22.1.56 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.

22.1.57 radio

The `radio` tool was created entirely with *fluid*. It shows some of the available button types and tests radio button behavior.

22.1.58 resizebox

`resizebox` shows some possible ways of FLTK's automatic resize behavior.

22.1.59 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.

22.1.60 resize

The `resize` demo tests size and position functions with the given window manager.

22.1.61 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 rendering and updates.

22.1.62 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.

22.1.63 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.

22.1.64 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.

22.1.65 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.).

22.1.66 table

The `table` demo shows the features of the [Fl_Table](#) widget.

22.1.67 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.

22.1.68 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.

22.1.69 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.

22.1.70 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.

22.1.71 tree

The `tree` demo shows the features of the [FL_Tree](#) widget.

22.1.72 twowin

The `twowin` program tests focus transfer from one window to another window.

22.1.73 unittests

`unittests` exercises all of FLTK's drawing features (e.g., text, lines, circles, images), as well as scrollbars and schemes.

22.1.74 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.

22.1.75 valuators

`valuators` shows all of FLTK's nifty widgets to change numeric values.

22.1.76 windowfocus

`windowfocus` shows a very special case when a new window is shown while the focus stays in the original window.

22.1.77 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`. See the [Fluid Tutorial](#) for more details.

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.

22.2.1 cairo_test

The `cairo_test` demo program shows three shiny buttons drawn with Cairo in an [FI_Cairo_Window](#).



Figure 22.1 Buttons drawn with Cairo

22.2.2 icon

The `icon` program lets you set the program icon from an image (here an [FI_RGB_Image](#)).

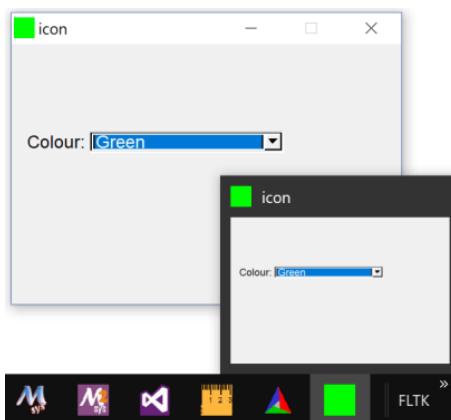


Figure 22.2 Green icon (Windows 10)

22.2.3 unittests

Select "drawing images" in the browser at the left side to see the image drawing example:

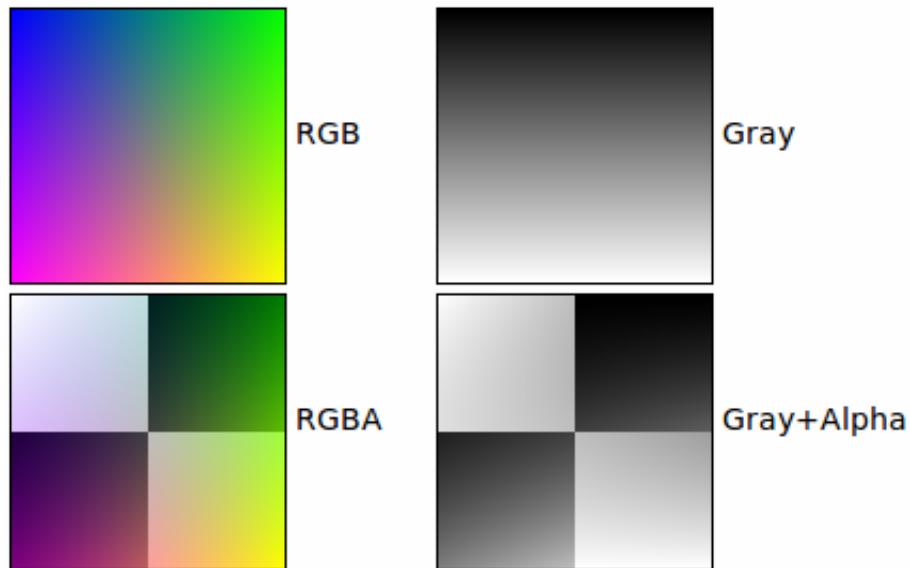


Figure 22.3 Image Drawing

Chapter 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?](#)

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". "Erc0'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.

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.

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 GPL 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 GPL.

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>).

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():
F1::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);
}
```

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 24

Todo List

Page [Adding and Extending Widgets](#)

Clarify [FL_Window::damage\(uchar\)](#) handling - seems confused/wrong? ORing value doesn't match setting behaviour in [FL_Widget.H!](#)

Clarify [FL_Widget::test_shortcut\(\)](#) explanations. [FL_Widget.h](#) says Internal Use only, but subclassing chapter gives details!

Module [Box Types](#)

Description of boxtypes is incomplete. See below for the defined enum [FL_Boxtype](#).

Member [FL_Browser_::scrollbar_width \(\) const](#)

This method should eventually be removed in 1.4+

Member [FL_Browser_::scrollbar_width \(int width\)](#)

This method should eventually be removed in 1.4+

Member [FL_Browser_::sort \(int flags=0\)](#)

Add a flag to ignore case

Class [FL_Button](#)

Refactor the doxygen comments for [FL_Button type\(\)](#) documentation.

Refactor the doxygen comments for [FL_Button when\(\)](#) documentation.

Member [fl_casenumericssort \(struct dirent **A, struct dirent **B\)](#)

Make comparison UTF-8 aware.

Class [FL_Chart](#)

Refactor [FL_Chart::type\(\)](#) information.

Class [FL_Choice](#)

Refactor the doxygen comments for [FL_Choice changed\(\)](#) documentation.

Class [FL_Counter](#)

Refactor the doxygen comments for [FL_Counter type\(\)](#) documentation.

Member [FL_Cursor](#)

enum [FL_Cursor](#) needs maybe an image.

Member [FL_File_Input::errorcolor \(\) const](#)

Remove [FL_File_Input::errorcolor\(\)](#) in FLTK 1.5.0 or higher.

Member [FL_File_Input::errorcolor \(FL_Color c\)](#)

Remove [FL_File_Input::errorcolor\(FL_Color\)](#) in FLTK 1.5.0 or higher.

Member [fl_filename_list \(const char *d, struct dirent ***l, FL_File_Sort_F *s=fl_numericsort\)](#)

should support returning OS error messages

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

Class `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_numericsort (struct dirent **A, struct dirent **B)`

Make comparison UTF-8 aware.

Member `Fl_Preferences::Fl_Preferences (Root root, const char *vendor, const char *application)`

(Matt) Before the release of 1.4.0, I want to make a further attempt to write a preferences file smarter. I plan to use a subgroup of the "runtime" preferences to store data and stay accessible until the application exits. Data would be stored under `./$(vendor)/$(application).prefs` in RAM, but not on disk.

(Matt) I want a way to access the type of the root preferences (SYSTEM, USER, MEMORY), and the state of the file access (OK, FILE_SYSTEM_FAIL, PERMISSION_FAIL, etc.), and probably the `dirty()` flag as well.

(Matt) Also, I need to explain runtime preferences.

(Matt) Lastly, I think I have to put short sample code in the Doxygen docs. The test app ist just not enough.

Member `Fl_Preferences::get (const char *entry, void *value, const void *defaultValue, int defaultSize, int maxSize)`

`maxSize` should receive the number of bytes that were read.

Member `fl_reset_spot (void)`

provide user documentation for `fl_reset_spot` function

Member `Fl_Scroll::bbox (int &, int &, int &, int &)`

The visibility of the scrollbars ought to be checked/calculated outside of the `draw()` method (STR #1895).

Member `fl_set_spot (int font, int size, int X, int Y, int W, int H, Fl_Window *win=0)`

provide user documentation for `fl_set_spot` function

Member `fl_set_status (int X, int Y, int W, int H)`

provide user documentation for `fl_set_status` function

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_Text_Display::display_insert ()`

Unicode?

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 `FI_Text_Display::highlight_data (FI_Text_Buffer *styleBuffer, const Style_Table_Entry *styleTable, int nStyles, char unfinishedStyle, Unfinished_Style_Cb unfinishedHighlightCB, void *cbArg)`
 "extendRangeForStyleMods" does not exist (might be a hangover from the port from nedit). Find the correct function.

Member `FI_Text_Display::maintain_absolute_top_line_number (int state)`

TextDPosToLineAndCol does not exist (nedit port?)

Member `FI_Text_Display::overstrike (const char *text)`

Unicode? Find out exactly what we do here and simplify.

Member `FI_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 `FI_Text_Display::scroll (int topLineNum, int horizOffset)`

Column numbers make little sense here.

Member `FI_Text_Display::scrollbar_width () const`

This method should eventually be removed.

Member `FI_Text_Display::scrollbar_width (int width)`

This method should eventually be removed

Member `FI_Text_Display::shortcut () const`

FIXME : get set methods pointing on shortcut_ have no effects as shortcut_ is unused in this class and derived!

Member `FI_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 `FI_Text_Display::wrap_uses_character (int lineEndPos) const`

TextDEndOfLine and BufEndOfLine functions don't exist (nedit port?)

Member `FI_Text_Display::wrapped_column (int row, int column) 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?)

Unicode?

Member `FI_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 `FI_Tiled_Image::FI_Tiled_Image (FI_Image *i, int W=0, int H=0)`

Fix `FI_Tiled_Image` as background image for widgets and windows and fix the implementation of `FI::scheme(const char *)`.

Member `FI_Tree::handle (int e)`

add `FI_Widget_Tracker` (see `FI_Browser_.cxx::handle()`)

Member `FI_Tree::is_scrollbar (FI_Widget *w)`

should be const

Member `FI_Tree::show_self ()`

should be const

Member `FI_When`

doxygen comments for values are incomplete and maybe wrong or unclear

Member `FI_Widget::argument () const`

[Internal] The user_data value must be implemented using `f1_intptr_t` or similar to avoid 64-bit platform incompatibilities.

Member `FI_Widget::type () const`

Explain "simulate RTTI" (currently only used to decide if a widget is a window, i.e. `type() >= FL_WINDOW ?`). Is `type()` really used in a way that ensures "Forms compatibility" ?

Member [FI_Window::show \(\)](#)

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".

Member [FI_Window::show \(int argc, char **argv\)](#)

explain which system parameters are set up.

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.

Module [Mouse and Keyboard Events](#)

FL_Button and FL_key... constants could be structured better (use an enum or some doxygen grouping ?)

Page [Unicode and UTF-8 Support](#)

FLTK 1.3 and later supports the full Unicode range (21 bits), but there are a few exceptions, for instance binary shortcut values in menus ([FI_Shortcut](#)) can only be used with characters from the BMP (16 bits). This may be extended in a future FLTK version.

Work through the code and this documentation to harmonize the [[OksiD](#)] and [[fltk2](#)] functions.

Verify 16/24 bit Unicode limit for different character sets? OksiD's code appears limited to 16-bit whereas the FLTK2 code appears to handle a wider set. What about illegal characters? See comments in `fl_utf8fromwc()` and `fl_utf8toUtf16()`.

Chapter 25

Deprecated List

Member `Fl::release ()`

Use `Fl::grab(0)` instead.

Member `Fl::set_idle (Fl_Old_Idle_Handler cb)`

This method is obsolete - use the `add_idle()` method instead.

Member `Fl::version ()`

Use `int Fl::api_version()` instead.

Member `fl_ask (const char *fmt,...)`

`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()` with the appropriate verbs instead.

Member `Fl_Browser_::scrollbar_width () const`

Use `scrollbar_size()` instead.

Member `Fl_Browser_::scrollbar_width (int width)`

Use `scrollbar_size()` instead.

Member `fl_clip`

`fl_clip(int, int, int, int)` is deprecated and will be removed from future releases. Please use `fl_push_clip(int x, int y, int w, int h)` instead.

Member `Fl_File_Input::errorcolor () const`

Will be removed in FLTK 1.5.0 or higher.

Member `Fl_File_Input::errorcolor (Fl_Color c)`

Will be removed in FLTK 1.5.0 or higher.

Member `Fl_Group::focus (Fl_Widget *W)`

This is for backwards compatibility only.

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_Surface::highres_image ()`

Use `image()` instead.

Member `Fl_Menu_Item::check ()`

Member `Fl_Menu_Item::checked () const`

Member [FI_Menu_Item::setonly \(\)](#)

This method is dangerous if radio items are first in the menu. Use [FI_Menu_::setonly\(FI_Menu_Item*\)](#) instead.

Member [FI_Menu_Item::uncheck \(\)](#)**Member [FI_Text_Display::scrollbar_width \(\) const](#)**

Use [scrollbar_size\(\)](#) instead.

Member [FI_Text_Display::scrollbar_width \(int width\)](#)

Use [scrollbar_size\(\)](#) instead.

Member [FI_Tree::first_visible \(\)](#)

in 1.3.3 ABI – use [first_visible_item\(\)](#) instead.

Member [FI_Tree::item_clicked \(FI_Tree_Item *val\)](#)

in 1.3.3 ABI – use [callback_item\(\)](#) instead.

Member [FI_Tree::item_clicked \(\)](#)

in 1.3.3 ABI – use [callback_item\(\)](#) instead.

Member [FI_Tree::last_visible \(\)](#)

in 1.3.3 – use [last_visible_item\(\)](#) instead.

Member [FI_Tree_Item::FI_Tree_Item \(const FI_Tree_Prefs &prefs\)](#)

in 1.3.3 ABI – you must use [FI_Tree_Item\(FI_Tree*\)](#) for proper horizontal scrollbar behavior.

Member [FI_Tree_Item::next_displayed \(FI_Tree_Prefs &prefs\)](#)

in 1.3.3 for confusing name, use [next_visible\(\)](#) instead

Member [FI_Tree_Item::prev_displayed \(FI_Tree_Prefs &prefs\)](#)

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 [FI_Widget::color2 \(unsigned a\)](#)

Use [selection_color\(unsigned\)](#) instead.

Member [FI_Widget::color2 \(\) const](#)

Use [selection_color\(\)](#) instead.

Member [FI_Window::free_position \(\)](#)

please use [force_position\(0\)](#) instead

Member [FI_Window::icon \(\) const](#)

in 1.3.3

Member [FI_Window::icon \(const void *ic\)](#)

in 1.3.3

Chapter 26

Module Index

26.1 Modules

Here is a list of all modules:

Callback function typedefs	229
runtime graphics driver configuration	231
runtime printer driver configuration	232
runtime window and event manager configuration	233
runtime system configuration	234
Windows handling functions	235
Events handling functions	238
Selection & Clipboard functions	256
Screen functions	260
Color & Font functions	267
Drawing functions	278
Multithreading support functions	313
Safe widget deletion support functions	315
Cairo Support Functions and Classes	319
Unicode and UTF-8 functions	321
Mac OS X-specific symbols	342
Common Dialogs classes and functions	344
File names and URI utility functions	359
Fl_string	367

Chapter 27

Hierarchical Index

27.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

FI_Preferences::Entry	369
FI	369
FI_Cairo_State	469
FL_CHART_ENTRY	477
FI_End	524
FI_File_Chooser	532
FI_File_Icon	544
FI_GI_Choice	568
FI_Glut_Bitmap_Font	580
FI_Glut_StrokeChar	580
FI_Glut_StrokeFont	580
FI_Glut_StrokeStrip	581
FI_Glut_StrokeVertex	581
FI_Help_Block	598
FI_Help_Dialog	599
FI_Help_Font_Stack	603
FI_Help_Font_Style	604
FI_Help_Link	605
FI_Help_Target	606
FI_Image	621
FI_Bitmap	406
FI_XBM_Image	1242
FI_Pixmap	753
FI_GIF_Image	567
FI_XPM_Image	1244
FI_RGB_Image	816
FI_BMP_Image	410
FI_JPEG_Image	671
FI_PNG_Image	760
FI_PNM_Image	762
FI_SVG_Image	888
FI_Shared_Image	843
FI_Tiled_Image	1033
FI_Image_Reader	632
FI_Label	672

FI_Mac_App_Menu	677
FI_Menu_Item	704
FI_Multi_Label	726
FI_Native_File_Chooser	731
FI_Plugin	757
FI_Device_Plugin	516
FI_Preferences	775
FI_Plugin_Manager	758
FI_Rect	808
FI_Scroll::ScrollInfo::FI_Region_LRTB	811
FI_Scroll::ScrollInfo::FI_Region_XYWH	811
FI_Scroll::ScrollInfo::FI_Scrollbar_Data	839
FI_Surface_Device	880
FI_Display_Device	521
FI_Widget_Surface	1207
FI_Copy_Surface	508
FI_EPS_File_Surface	525
FI_Image_Surface	632
FI_Paged_Device	747
FI_PostScript_File_Device	768
FI_Printer	796
FI_SVG_File_Surface	883
FI_Text_Buffer	939
FI_Text_Selection	1024
FI_Tooltip	1041
FI_Tree_Item	1100
FI_Tree_Item_Array	1128
FI_Tree_Prefs	1132
FI_Widget	1159
FI_Box	411
FI_Button	463
FI_Light_Button	674
FI_Check_Button	487
FI_Radio_Light_Button	807
FI_Round_Button	825
FI_Radio_Round_Button	807
FI_Radio_Button	806
FI_Repeat_Button	812
FI_Return_Button	814
FI_Toggle_Button	1039
FI_Chart	472
FI_Clock_Output	496
FI_Clock	493
FI_Round_Clock	827
FI_FormsBitmap	559
FI_FormsPixmap	560
FI_FormsText	563
FI_Free	564
FI_Group	584
FI_Browser	442
FI_Browser	414
FI_File_Browser	529
FI_Hold_Browser	617
FI_Multi_Browser	725
FI_Select_Browser	842
FI_Check_Browser	478

Fl_Color_Chooser	502
Fl_Help_View	606
Fl_Input_Choice	664
Fl_Pack	745
Fl_Scroll	828
Fl_Spinner	871
Fl_Table	904
Fl_Table_Row	927
Fl_Tabs	930
Fl_Text_Display	962
Fl_Simple_Terminal	853
Fl_Text_Editor	1010
Fl_Tile	1029
Fl_Tree	1047
Fl_Window	1213
Fl_Double_Window	521
Fl_Cairo_Window	470
Fl_Overlay_Window	741
Fl_GI_Window	569
Fl_Glut_Window	581
Fl_Single_Window	865
Fl_Menu_Window	721
Fl_Wizard	1240
Fl_Input_	641
Fl_Input	638
Fl_File_Input	552
Fl_Float_Input	558
Fl_Int_Input	670
Fl_Multiline_Input	729
Fl_Output	739
Fl_Multiline_Output	730
Fl_Secret_Input	840
Fl_Spinner::Fl_Spinner_Input	879
Fl_Menu_	679
Fl_Choice	489
Fl_Menu_Bar	697
Fl_Sys_Menu_Bar	893
Fl_Menu_Button	700
Fl_Positioner	763
Fl_Progress	803
Fl_Timer	1036
Fl_Valuator	1139
Fl_Adjuster	404
Fl_Counter	512
Fl_Simple_Counter	852
Fl_Dial	518
Fl_Fill_Dial	555
Fl_Line_Dial	677
Fl_Roller	823
Fl_Slider	866
Fl_Fill_Slider	557
Fl_Hor_Fill_Slider	618
Fl_Hor_Nice_Slider	619
Fl_Hor_Slider	619
Fl_Nice_Slider	739
Fl_Scrollbar	835
Fl_Value_Slider	1155

FI_Hor_Value_Slider	620
FI_Value_Input	1145
FI_Value_Output	1151
FI_Widget_Tracker	1211
FI_XColor	1243
FI_Text_Editor::Key_Binding	1245
FI_Preferences::Name	1245
FI_Preferences::Node	1247
FI_Paged_Device::page_format	1247
FI_Preferences::RootNode	1248
FI_Scroll::ScrollInfo	1248
FI_Text_Display::Style_Table_Entry	1249

Chapter 28

Class Index

28.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

FI_Preferences::Entry	369
FI The FI is the FLTK global (static) class containing state information and global methods for the current application	369
FI_Adjuster Was stolen from Prisms, and has proven to be very useful for values that need a large dynamic range	404
FI_Bitmap Supports caching and drawing of mono-color (bitmap) images	406
FI_BMP_Image Supports loading, caching, and drawing of Windows Bitmap (BMP) image files	410
FI_Box This widget simply draws its box, and possibly its label	411
FI_Browser Displays a scrolling list of text lines, and manages all the storage for the text	414
FI_Browser_ This is the base class for browsers	442
FI_Button Buttons generate callbacks when they are clicked by the user	463
FI_Cairo_State Contains all the necessary info on the current cairo context	469
FI_Cairo_Window This defines a FLTK window with cairo support	470
FI_Chart FI_Chart displays simple charts	472
FL_CHART_ENTRY For internal use only	477
FI_Check_Browser Displays a scrolling list of text lines that may be selected and/or checked by the user	478
FI_Check_Button A button with a "checkmark" to show its status	487
FI.Choice A button that is used to pop up a menu	489
FI_Clock This widget provides a round analog clock display	493

FI_Clock_Output	This widget can be used to display a program-supplied time	496
FI_Color_Chooser	Standard RGB color chooser	502
FI_Copy_Surface	Supports copying of graphical data to the clipboard	508
FI_Counter	Controls a single floating point value with button (or keyboard) arrows	512
FI_Device_Plugin	This plugin socket allows the integration of new device drivers for special window or screen types	516
FI_Dial	Circular dial to control a single floating point value	518
FI_Display_Device	A display to which the computer can draw	521
FI_Double_Window	The FI_Double_Window provides a double-buffered window	521
FI_End	This is a dummy class that allows you to end a FI_Group in a constructor list of a class:	524
FI_EPS_File_Surface	Encapsulated PostScript drawing surface	525
FI_File_Browser	Displays a list of filenames, optionally with file-specific icons	529
FI_File_Chooser	Displays a standard file selection dialog that supports various selection modes	532
FI_File_Icon	Manages icon images that can be used as labels in other widgets and as icons in the FileBrowser widget	544
FI_File_Input	This widget displays a pathname in a text input field	552
FI_Fill_Dial	Draws a dial with a filled arc	555
FI_Fill_Slider	Widget that draws a filled horizontal slider, useful as a progress or value meter	557
FI_Float_Input	Subclass of FI_Input that only allows the user to type floating point numbers (sign, digits, decimal point, more digits, 'E' or 'e', sign, digits)	558
FI_FormsBitmap	Forms compatibility Bitmap Image Widget	559
FI_FormsPixmap	Forms pixmap drawing routines	560
FI_FormsText	563
FI_Free	Emulation of the Forms "free" widget	564
FI_GIF_Image	Supports loading, caching, and drawing of Compuserve GIF SM images	567
FI_GL_Choice	568
FI_GL_Window	Sets things up so OpenGL works	569
FI_Glut_Bitmap_Font	Fltk glut font/size attributes used in the glutXXX functions	580
FI_Glut_StrokeChar	580
FI_Glut_StrokedFont	580
FI_Glut_StrokedStrip	581
FI_Glut_StrokedVertex	581
FI_Glut_Window	GLUT is emulated using this window class and these static variables (plus several more static variables hidden in glut_compatibility.cxx):	581

FI_Group	FLTK container widget	584
FI_Help_Block	598
FI_Help_Dialog	Displays a standard help dialog window using the FI_Help_View widget	599
FI_Help_Font_Stack	603
FI_Help_Font_Style	FI_Help_View font stack element definition	604
FI_Help_Link	Definition of a link for the html viewer	605
FI_Help_Target	FI_Help_Target structure	606
FI_Help_View	Displays HTML text	606
FI_Hold_Browser	The FI_Hold_Browser is a subclass of FI_Browser which lets the user select a single item, or no items by clicking on the empty space	617
FI_Hor_Fill_Slider	618
FI_Hor_Nice_Slider	619
FI_Hor_Slider	Horizontal Slider class	619
FI_Hor_Value_Slider	620
FI_Image	Base class for image caching, scaling and drawing	621
FI_Image_Reader	632
FI_Image_Surface	Directs all graphics requests to an FI_Image	632
FI_Input	This is the FLTK text input widget	638
FI_Input_	This class provides a low-overhead text input field	641
FI_Input_Choice	A combination of the input widget and a menu button	664
FI_Int_Input	Subclass of FI_Input that only allows the user to type decimal digits (or hex numbers of the form 0xaef)	670
FI_JPEG_Image	Supports loading, caching, and drawing of Joint Photographic Experts Group (JPEG) File Interchange Format (JFIF) images	671
FI_Label	This struct stores all information for a text or mixed graphics label	672
FI_Light_Button	This subclass displays the "on" state by turning on a light, rather than drawing pushed in	674
FI_Line_Dial	677
FI_Mac_App_Menu	677
FI_Menu_	Base class of all widgets that have a menu in FLTK	679
FI_Menu_Bar	This widget provides a standard menubar interface	697
FI_Menu_Button	This is a button that when pushed pops up a menu (or hierarchy of menus) defined by an array of FI_Menu_Item objects	700
FI_Menu_Item	The FI_Menu_Item structure defines a single menu item that is used by the FI_Menu_ class	704
FI_Menu_Window	Window type used for menus	721
FI_Multi_Browser	Subclass of FI_Browser which lets the user select any set of the lines	725

FI_Multi_Label	Allows a mixed text and/or graphics label to be applied to an FI_Menu_Item or FI_Widget	726
FI_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	729
FI_Multiline_Output	This widget is a subclass of FI_Output that displays multiple lines of text	730
FI_Native_File_Chooser	This class lets an FLTK application easily and consistently access the operating system's native file chooser	731
FI_Nice_Slider	739
FI_Output	This widget displays a piece of text	739
FI_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	741
FI_Pack	This widget was designed to add the functionality of compressing and aligning widgets	745
FI_Paged_Device	Represents page-structured drawing surfaces	747
FI_Pixmap	Supports caching and drawing of colormap (pixmap) images, including transparency	753
FI_Plugin	FI_Plugin allows link-time and run-time integration of binary modules	757
FI_Plugin_Manager	FI_Plugin_Manager manages link-time and run-time plugin binaries	758
FI_PNG_Image	Supports loading, caching, and drawing of Portable Network Graphics (PNG) image files	760
FI_PNM_Image	Supports loading, caching, and drawing of Portable Anymap (PNM, PBM, PGM, PPM) image files	762
FI_Positioner	This class is provided for Forms compatibility	763
FI_PostScript_File_Device	To send graphical output to a PostScript file	768
FI_Preferences	FI_Preferences provides methods to store user settings between application starts	775
FI_Printer	OS-independent print support	796
FI_Progress	Displays a progress bar for the user	803
FI_Radio_Button	806
FI_Radio_Light_Button	807
FI_Radio_Round_Button	807
FI_Rect	Rectangle with standard FLTK coordinates (X, Y, W, H)	808
FI_Scroll::ScrollInfo::FI_Region_LRTB	A local struct to manage a region defined by left/right/top/bottom	811
FI_Scroll::ScrollInfo::FI_Region_XYWH	A local struct to manage a region defined by xywh	811
FI_Repeat_Button	The FI_Repeat_Button is a subclass of FI_Button that generates a callback when it is pressed and then repeatedly generates callbacks as long as it is held down	812
FI_Return_Button	The FI_Return_Button is a subclass of FI_Button that generates a callback when it is pressed or when the user presses the Enter key	814
FI_RGB_Image	Supports caching and drawing of full-color images with 1 to 4 channels of color information	816

FI_Roller	"dolly" control commonly used to move 3D objects	823
FI_Round_Button	Buttons generate callbacks when they are clicked by the user	825
FI_Round_Clock	A clock widget of type FL_ROUND_CLOCK	827
FI_Scroll	This container widget lets you maneuver around a set of widgets much larger than your window	828
FI_Scrollbar	Displays a slider with arrow buttons at the ends of the scrollbar	835
FI_Scroll::ScrollInfo::FI_Scrollbar_Data	A local struct to manage a scrollbar's xywh region and tab values	839
FI_Secret_Input	Subclass of FI_Input that displays its input as a string of placeholders	840
FI_Select_Browser	The class is a subclass of FI_Browser which lets the user select a single item, or no items by clicking on the empty space	842
FI_Shared_Image	This class supports caching, loading, and drawing of image files	843
FI_Simple_Counter	This widget creates a counter with only 2 arrow buttons	852
FI_Simple_Terminal	This is a continuous text scroll widget for logging and debugging output, much like a terminal .	853
FI_Single_Window	This is the same as FI_Window	865
FI_Slider	Sliding knob inside a box	866
FI_Spinner	This widget is a combination of a numerical input widget and repeat buttons	871
FI_Spinner::FI_Spinner_Input	879
FI_Surface_Device	A drawing surface that's susceptible to receive graphical output	880
FI_SVG_File_Surface	A drawing surface producing a Scalable Vector Graphics (SVG) file	883
FI_SVG_Image	Supports loading, caching and drawing of scalable vector graphics (SVG) images	888
FI_Sys_Menu_Bar	A class to create and modify menus that appear on macOS in the menu bar at the top of the screen	893
FI_Table	A table of widgets or other content	904
FI_Table_Row	A table with row selection capabilities	927
FI_Tabs	"file card tabs" interface that allows you to put lots and lots of buttons and switches in a panel, as popularized by many toolkits	930
FI_Text_Buffer	This class manages Unicode text displayed in one or more FI_Text_Display widgets	939
FI_Text_Display	Rich text display widget	962
FI_Text_Editor	This is the FLTK text editor widget	1010
FI_Text_Selection	This is an internal class for FI_Text_Buffer to manage text selections	1024
FI_Tile	Lets you resize its children by dragging the border between them	1029
FI_Tiled_Image	This class supports tiling of images over a specified area	1033

FI_Timer	This is provided only to emulate the Forms Timer widget	1036
FI_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	1039
FI_Tooltip	Tooltip support for all FLTK widgets	1041
FI_Tree	Tree widget	1047
FI_Tree_Item	Tree widget item	1100
FI_Tree_Item_Array	Manages an array of FI_Tree_Item pointers	1128
FI_Tree_Prefs	Tree widget's preferences	1132
FI_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	1139
FI_Value_Input	Displays a numeric value	1145
FI_Value_Output	Displays a floating point value	1151
FI_Value_Slider	FI_Slider widget with a box displaying the current value	1155
FI_Widget	FI_Widget is the base class for all widgets in FLTK	1159
FI_Widget_Surface	A surface on which any FLTK widget can be drawn	1207
FI_Widget_Tracker	This class should be used to control safe widget deletion	1211
FI_Window	This widget produces an actual window	1213
FI_Wizard	This widget is based off the FI_Tabs widget, but instead of displaying tabs it only changes "tabs" under program control	1240
FI_XBM_Image	Supports loading, caching, and drawing of X Bitmap (XBM) bitmap files	1242
FI_XColor	1243
FI_XPM_Image	Supports loading, caching, and drawing of X Pixmap (XPM) images, including transparency	1244
FI_Text_Editor::Key_Binding	Simple linked list item associating a key/state to a function	1245
FI_Preferences::Name	'Name' provides a simple method to create numerical or more complex procedural names for entries and groups on the fly	1245
FI_Preferences::Node	1247
FI_Paged_Device::page_format	Width, height and name of a page format	1247
FI_Preferences::RootNode	1248
FI_Scroll::ScrollInfo	Structure to manage scrollbar and widget interior sizes	1248
FI_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	1249

Chapter 29

File Index

29.1 File List

Here is a list of all documented files with brief descriptions:

abi-version.h	??
android.H	??
armscii_8.h	??
ascii.h	??
big5.h	??
big5_emacs.h	??
case.h	??
cgdebug.h	??
config_lib.h	??
cp1133.h	??
cp1251.h	??
cp1255.h	??
cp1256.h	??
cp936ext.h	??
dingbats_.h	??
Enumerations.H	
This file contains type definitions and general enumerations	1251
fastarrow.h	??
filename.H	
File names and URI utility functions	1280
Fl.cxx	
Implementation of the member functions of class Fl	1281
Fl.H	
Fl static class	1283
Fl_Adjuster.H	??
fl_arc.cxx	
Utility functions for drawing arcs and circles	1284
fl_ask.cxx	
Utility functions for common dialogs	1284
fl_ask.H	
API for common dialogs	1286
Fl_Bitmap.H	??
Fl_BMP_Image.H	??
Fl_Box.H	??
fl_boxtype.cxx	
Drawing code for common box types	1288

Fl_Browser.H	??
Fl_Browser_.H	??
Fl_Button.H	??
Fl_Cairo.H	??
Fl_Cairo_Window.H	??
Fl_Chart.H	??
Fl_Check_Browser.H	??
Fl_Check_Button.H	??
Fl_Choice.H	??
Fl_Clock.H	??
fl_cmap.h	??
fl_color.cxx	Color handling	1290
Fl_Color_Chooser.H	Fl_Color_Chooser widget	1291
Fl_compose.cxx	Utility functions to support text input	1291
Fl_Copy_Surface.H	??
Fl_Counter.H	??
fl_curve.cxx	Utility for drawing Bezier curves, adding the points to the current fl_begin/fl_vertex/fl_end path .	1292
Fl_Device.H	Declaration of classes Fl_Surface_Device, Fl_Display_Device, Fl_Device_Plugin	1292
Fl_Dial.H	??
Fl_Double_Window.cxx	Fl_Double_Window implementation	1292
Fl_Double_Window.H	??
fl_draw.H	Utility header to pull drawing functions together	1293
Fl_Export.H	??
Fl_File_Browser.H	??
Fl_File_Chooser.H	??
Fl_File_Icon.H	??
Fl_File_Input.H	??
Fl_Fill_Dial.H	??
Fl_Fill_Slider.H	??
Fl_Float_Input.H	??
Fl_FormsBitmap.H	??
Fl_FormsPixmap.H	??
Fl_Free.H	??
Fl_GIF_Image.H	??
Fl_GI_Choice.H	??
Fl_GI_Window.H	??
Fl_GI_Window_Driver.H	??
Fl_Graphics_Driver.H	??
Fl_Group.H	??
Fl_Help_Dialog.H	??
Fl_Help_View.H	??
Fl_Hold_Browser.H	??
Fl_Hor_Fill_Slider.H	??
Fl_Hor_Nice_Slider.H	??
Fl_Hor_Slider.H	??
Fl_Hor_Value_Slider.H	??
Fl_Image.H	Fl_Image, Fl_RGB_Image classes	1298
Fl_Image_Reader.h	??
Fl_Image_Surface.H	??
Fl_Input.H	??

FI_Input.H	??
FI_Input_Choice.H	??
FI_Int_Input.H	??
FI_JPEG_Image.H	??
FI_Light_Button.H	??
FI_Line_Dial.H	??
FI_Menu.H	??
FI_Menu_.H	??
FI_Menu_Bar.H	??
FI_Menu_Button.H	??
FI_Menu_Item.H	1299
FI_Menu_Window.H	??
fl_message.H	??
FI_Multi_Browser.H	??
FI_Multi_Label.H	??
FI_Multiline_Input.H	??
FI_Multiline_Output.H	??
FI_Native_File_Chooser.H	
FI_Native_File_Chooser widget	1300
FI_Nice_Slider.H	??
FI_Object.H	??
FI_Output.H	??
FI_Overlay_Window.H	??
FI_Pack.H	??
FI_Paged_Device.cxx	
Implementation of class FI_Paged_Device	1301
FI_Paged_Device.H	
Declaration of class FI_Paged_Device	1301
FI_Pixmap.H	??
FI_Plugin.H	??
FI_PNG_Image.H	??
FI_PNM_Image.H	??
FI_Positioner.H	??
FI_PostScript.H	
Declaration of classes FI_PostScript_Graphics_Driver , FI_PostScript_File_Device	1301
FI_Preferences.H	??
FI_Printer.H	
Declaration of class FI_Printer	1302
FI_Progress.H	??
FI_Radio_Button.H	??
FI_Radio_Light_Button.H	??
FI_Radio_Round_Button.H	??
fl_rect.cxx	
Drawing and clipping routines for rectangles	1302
FI_Rect.H	??
FI_Repeat_Button.H	??
FI_Return_Button.H	??
FI_RGB_Image.H	??
FI_Roller.H	??
FI_Round_Button.H	??
FI_Round_Clock.H	??
FI_Screen_Driver.H	??
FI_Scroll.H	??
FI_Scrollbar.H	??
FI_Secret_Input.H	??
FI_Select_Browser.H	??
FI_Shared_Image.H	
FI_Shared_Image class	1303

fl_show_colormap.H	The <code>fl_show_colormap()</code> function hides the implementation classes used to provide the popup window and color selection mechanism	1303
fl_show_input.H	??
FI_Simple_Counter.H	??
FI_Simple_Terminal.H	??
FI_Single_Window.H	??
FI_Slider.H	??
FI_Spinner.H	??
fl_string.h	Public header for FLTK's own platform agnostic string handling	1304
FI_SVG_File_Surface.H	??
FI_SVG_Image.H	??
FI_Sys_Menu_Bar.H	Definition of class <code>FI_Sys_Menu_Bar</code>	1304
FI_Sys_Menu_Bar_Driver.H	??
FI_System_Driver.H	??
FI_Table.H	??
FI_Table_Row.H	??
FI_Tabs.H	??
FI_Text_Buffer.H	??
FI_Text_Display.H	??
FI_Text_Editor.H	??
FI_Tile.H	??
FI_Tiled_Image.H	??
FI_Timer.H	??
FI_Toggle_Button.H	??
FI_Toggle_Light_Button.H	??
FI_Toggle_Round_Button.H	??
FI_Tooltip.H	??
FI_Tree.H	This file contains the definitions of the <code>FI_Tree</code> class	1305
FI_Tree_Item.H	This file contains the definitions for <code>FI_Tree_Item</code>	1306
FI_Tree_Item_Array.H	This file defines a class that manages an array of <code>FI_Tree_Item</code> pointers	1306
FI_Tree_Prefs.H	This file contains the definitions for <code>FI_Tree</code> 's preferences	1307
fl_types.h	This file contains simple "C"-style type definitions	1309
fl_utf8.h	Header for Unicode and UTF-8 character handling	1310
FI_Valuator.H	??
FI_Value_Input.H	??
FI_Value_Output.H	??
FI_Value_Slider.H	??
fl_vertex.cxx	Portable drawing code for drawing arbitrary shapes with simple 2D transformations	1313
FI_Widget.H	<code>FI_Widget</code> , <code>FI_Label</code> classes	1313
FI_Widget_Surface.H	??
FI_Window.H	<code>FI_Window</code> widget	1314
FI_Window_Driver.H	??
FI_Wizard.H	??
FI_XBM_Image.H	??
FI_XColor.H	??
FI_XPM_Image.H	??

flstring.h	??
forms.H	??
freeglut_teapot_data.h	??
gb2312.h	??
georgian_academy.h	??
georgian_ps.h	??
gl.h	This file defines wrapper functions for OpenGL in FLTK	1315
gl2openGL.h	??
gl_draw.H	??
glu.h	??
glut.H	??
iso8859_1.h	??
iso8859_10.h	??
iso8859_11.h	??
iso8859_13.h	??
iso8859_14.h	??
iso8859_15.h	??
iso8859_16.h	??
iso8859_2.h	??
iso8859_3.h	??
iso8859_4.h	??
iso8859_5.h	??
iso8859_6.h	??
iso8859_7.h	??
iso8859_8.h	??
iso8859_9.h	??
iso8859_9e.h	??
jisx0201.h	??
jisx0208.h	??
jisx0212.h	??
koi8_c.h	??
koi8_r.h	??
koi8_u.h	??
ksc5601.h	??
mac.H	Mac OS X-specific symbols	1319
math.h	??
mediumarrow.h	??
mulelao.h	??
names.h	??
numericsort.c	1320
platform.H	??
platform_types.h	Definitions of platform-dependent types	1322
print_panel.h	??
slowarrow.h	??
spacing.h	??
symbol_.h	??
tatar_cyr.h	??
tcvn.h	??
tis620.h	??
ucs2be.h	??
utf8.h	??
utf8_internal.h	??
viscii.h	??
win32.H	??
x.H	??

Ximint.h	??
Xlibint.h	??
Xutf8.h	??

Chapter 30

Module Documentation

30.1 Callback function typedefs

Typedefs defined in <FL/FI.H> for callback or handler functions passed as function parameters.

Typedefs

- `typedef void(* FI_Abort_Handler) (const char *format,...)`
Signature of set_abort functions passed as parameters.
- `typedef int(* FI_Args_Handler) (int argc, char **argv, int &i)`
Signature of args functions passed as parameters.
- `typedef void(* FI_Atclose_Handler) (FI_Window *window, void *data)`
Signature of set_atclose functions passed as parameters.
- `typedef void(* FI_Awake_Handler) (void *data)`
Signature of some wakeup callback functions passed as parameters.
- `typedef void() FI_Box_Draw_F(int x, int y, int w, int h, FI_Color color)`
Signature of some box drawing functions passed as parameters.
- `typedef void(* FI_Clipboard_Notify_Handler) (int source, void *data)`
Signature of add_clipboard_notify functions passed as parameters.
- `typedef int(* FI_Event_Dispatch) (int event, FI_Window *w)`
Signature of event_dispatch functions passed as parameters.
- `typedef int(* FI_Event_Handler) (int event)`
Signature of add_handler functions passed as parameters.
- `typedef void(* FI_FD_Handler) (FL_SOCKET fd, void *data)`
Signature of add_fd functions passed as parameters.
- `typedef void(* FI_Idle_Handler) (void *data)`
Signature of add_idle callback functions passed as parameters.
- `typedef void() FI_Label_Draw_F(const FI_Label *label, int x, int y, int w, int h, FI_Align align)`
Signature of some label drawing functions passed as parameters.
- `typedef void() FI_Label_Measure_F(const FI_Label *label, int &width, int &height)`
Signature of some label measurement functions passed as parameters.
- `typedef void(* FI_Old_Idle_Handler) ()`
Signature of set_idle callback functions passed as parameters.
- `typedef int(* FI_System_Handler) (void *event, void *data)`
Signature of add_system_handler functions passed as parameters.
- `typedef void(* FI_Timeout_Handler) (void *data)`
Signature of some timeout callback functions passed as parameters.

30.1.1 Detailed Description

Typedefs defined in <FL/Fl.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

30.1.2 Typedef Documentation

30.1.2.1 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\(Fl_Event_Dispatch\)](#)

30.2 runtime graphics driver configuration

Variables

- static bool `Fl::cfg_gfx_cairo` = 0
 - Cairo rendering available, available on many platforms.*
- static bool `Fl::cfg_gfx_directx` = 0
 - DirectX rendering available, usually on Windows systems.*
- static bool `Fl::cfg_gfx_gdi` = 0
 - GDI rendering available, usually on Windows systems.*
- static bool `Fl::cfg_gfx_opengl` = 0
 - OpenGL rendering available, available on many platforms.*
- static bool `Fl::cfg_gfx_quartz` = 0
 - Quartz rendering available, usually on OS X systems.*
- static bool `Fl::cfg_gfx_xlib` = 0
 - X11 Xlib rendering available, usually on Linux systems.*

30.2.1 Detailed Description

30.3 runtime printer driver configuration

Variables

- static bool `Fl::cfg_prn_gdi` = 0
GDI rendering available, usually on Windows systems.
- static bool `Fl::cfg_prn_ps` = 0
PostScript rendering available, usually on Linux systems.
- static bool `Fl::cfg_prn_quartz` = 0
Quartz rendering available, usually on OS X systems.

30.3.1 Detailed Description

30.4 runtime window and event manager configuration

Variables

- static bool `Fl::cfg_win_cocoa` = 0
Cocoa window management available, usually on OS X systems.
- static bool `Fl::cfg_win_win32` = 0
Windows window management available, on low level Windows.
- static bool `Fl::cfg_win_x11` = 0
X11 window management available, usually on Linux systems.

30.4.1 Detailed Description

30.5 runtime system configuration

Variables

- static bool `Fl::cfg_sys_posix` = 0
Posix system available, usually on Linux and OS X systems, but also Cygwin.
- static bool `Fl::cfg_sys_win32` = 0
Windows system available, on Windows.

30.5.1 Detailed Description

30.6 Windows handling functions

Windows and standard dialogs handling declared in <FL/FL.H>

Functions

- static void [Fl::default_atclose \(Fl_Window *, void *\)](#)
Default callback for window widgets.
- static [Fl_Window * Fl::first_window \(\)](#)
Returns the first top-level window in the list of shown() windows.
- static void [Fl::first_window \(Fl_Window *\)](#)
Sets the window that is returned by [first_window\(\)](#).
- static [Fl_Window * Fl::grab \(\)](#)
Returns the window that currently receives all events.
- static void [Fl::grab \(Fl_Window *\)](#)
Selects the window to grab.
- static [Fl_Window * Fl::modal \(\)](#)
Returns the top-most [modal\(\)](#) window currently shown.
- static [Fl_Window * Fl::next_window \(const Fl_Window *\)](#)
Returns the next top-level window in the list of shown() windows.
- static void [Fl::set_abort \(Fl_Abort_Handler f\)](#)
For back compatibility, sets the void [Fl::fatal](#) handler callback.
- static void [Fl::set_atclose \(Fl_Atclose_Handler f\)](#)
For back compatibility, sets the [Fl::atclose](#) handler callback.

Variables

- static void(* [Fl::atclose \(\)\(Fl_Window *, void *\)](#))
Back compatibility: default window callback handler.

30.6.1 Detailed Description

Windows and standard dialogs handling declared in <FL/FL.H>

30.6.2 Function Documentation

30.6.2.1 default_atclose()

```
void Fl::default_atclose (
    Fl_Window * window,
    void * v ) [static]
```

Default callback for window widgets.

It hides the window and then calls the default widget callback.

30.6.2.2 first_window() [1/2]

```
F1_Window * Fl::first_window ( ) [static]
```

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.

30.6.2.3 first_window() [2/2]

```
void Fl::first_window (
    F1_Window * window ) [static]
```

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.

30.6.2.4 grab() [1/2]

```
static F1_Window* Fl::grab ( ) [inline], [static]
```

Returns the window that currently receives all events.

Returns

The window that currently receives all events, or NULL if event grabbing is currently OFF.

30.6.2.5 grab() [2/2]

```
void Fl::grab (
    F1_Window * win ) [static]
```

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.

30.6.2.6 modal()

```
static Fl_Window* Fl::modal ( ) [inline], [static]
```

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).

30.6.2.7 next_window()

```
Fl_Window * Fl::next_window (
    const Fl_Window * window ) [static]
```

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	<code>window</code>	must be shown and not NULL
----	---------------------	----------------------------

30.6.2.8 set_atclose()

```
static void Fl::set_atclose (
    Fl_Atclose_Handler f ) [inline], [static]
```

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\(Fl_Callback*\)](#)

30.6.3 Variable Documentation

30.6.3.1 atclose

```
void(* Fl::atclose) (Fl_Window *, void *)=default_atclose [static], [default]
```

Back compatibility: default window callback handler.

See also

[Fl::set_atclose\(\)](#)

30.7 Events handling functions

Fl class events handling API declared in <FL/Fl.H>

Functions

- static void `Fl::add_handler (Fl_Event_Handler h)`
Install a function to parse unrecognized events.
- static void `Fl::add_system_handler (Fl_System_Handler h, void *data)`
Install a function to intercept system events.
- static `Fl_Widget * Fl::belowmouse ()`
Gets the widget that is below the mouse.
- static void `Fl::belowmouse (Fl_Widget *)`
Sets the widget that is below the mouse.
- static int `Fl::compose (int &del)`
Any text editing widget should call this for each FL_KEYBOARD event.
- static void `Fl::compose_reset ()`
If the user moves the cursor, be sure to call `Fl::compose_reset()`.
- static void `Fl::disable_im ()`
Disables the system input methods facilities.
- static void `Fl::enable_im ()`
Enables the system input methods facilities.
- static int `Fl::event ()`
Returns the last event that was processed.
- static int `Fl::event_alt ()`
Returns non-zero if the Alt key is pressed.
- static int `Fl::event_button ()`
Gets which particular mouse button caused the current event.
- static int `Fl::event_button1 ()`
Returns non-zero if mouse button 1 is currently held down.
- static int `Fl::event_button2 ()`
Returns non-zero if button 2 is currently held down.
- static int `Fl::event_button3 ()`
Returns non-zero if button 3 is currently held down.
- static int `Fl::event_buttons ()`
Returns the mouse buttons state bits; if non-zero, then at least one button is pressed now.
- static int `Fl::event_clicks ()`
Returns non zero if we had a double click event.
- static void `Fl::event_clicks (int i)`
Manually sets the number returned by `Fl::event_clicks()`.
- static void * `Fl::event_clipboard ()`
During an FL_PASTE event of non-textual data, returns a pointer to the pasted data.
- static const char * `Fl::event_clipboard_type ()`
Returns the type of the pasted data during an FL_PASTE event.
- static int `Fl::event_command ()`
Returns non-zero if the FL_COMMAND key is pressed, either FL_CTRL or on OSX FL_META.
- static int `Fl::event_ctrl ()`
Returns non-zero if the Control key is pressed.
- static void `Fl::event_dispatch (Fl_Event_Dispatch d)`

- static `Fl_Event_Dispatch Fl::event_dispatch ()`

Set a new event dispatch function.
- static int `Fl::event_dx ()`

Return the current event dispatch function.
- static int `Fl::event_dy ()`

Returns the current horizontal mouse scrolling associated with the FL_MOUSEWHEEL event.
- static int `Fl::event_inside (int, int, int, int)`

Returns whether or not the mouse event is inside the given rectangle.
- static int `Fl::event_inside (const Fl_Widget *)`

Returns whether or not the mouse event is inside a given child widget.
- static 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".
- static void `Fl::event_is_click (int i)`

Clears the value returned by `Fl::event_is_click()`.
- static int `Fl::event_key ()`

Gets which key on the keyboard was last pushed.
- static int `Fl::event_key (int key)`

Returns true if the given `key` was held down (or pressed) during the last event.
- static int `Fl::event_length ()`

Returns the length of the text in `Fl::event_text()`.
- static int `Fl::event_original_key ()`

Returns the keycode of the last key event, regardless of the NumLock state.
- static int `Fl::event_shift ()`

Returns non-zero if the Shift key is pressed.
- static int `Fl::event_state ()`

Returns the keyboard and mouse button states of the last event.
- static int `Fl::event_state (int mask)`

Returns non-zero if any of the passed event state bits are turned on.
- static const char * `Fl::event_text ()`

Returns the text associated with the current event, including FL_PASTE or FL_DND_RELEASE events.
- static int `Fl::event_x ()`

Returns the mouse position of the event relative to the `Fl_Window` it was passed to.
- static int `Fl::event_x_root ()`

Returns the mouse position on the screen of the event.
- static int `Fl::event_y ()`

Returns the mouse position of the event relative to the `Fl_Window` it was passed to.
- static int `Fl::event_y_root ()`

Returns the mouse position on the screen of the event.
- static `Fl_Widget * Fl::focus ()`

Gets the current `Fl::focus()` widget.
- static void `Fl::focus (Fl_Widget *)`

Sets the widget that will receive FL_KEYBOARD events.
- static int `Fl::get_key (int key)`

Returns true if the given `key` is held down now.
- static void `Fl::get_mouse (int &, int &)`

Return where the mouse is on the screen by doing a round-trip query to the server.
- static int `Fl::handle (int, Fl_Window *)`

Handle events from the window system.
- static int `Fl::handle_ (int, Fl_Window *)`

- Handle events from the window system.*
- static [Fl_Widget * Fl::pushed \(\)](#)
Gets the widget that is being pushed.
 - static void [Fl::pushed \(Fl_Widget *\)](#)
Sets the widget that is being pushed.
 - static void [Fl::remove_handler \(Fl_Event_Handler h\)](#)
Removes a previously added event handler.
 - static void [Fl::remove_system_handler \(Fl_System_Handler h\)](#)
Removes a previously added system event handler.
 - static 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

- const char *const [fl_eventnames \[\]](#)
This is an array of event names you can use to convert event numbers into names.
- const char *const [fl_fontnames \[\]](#)
This is an array of font names you can use to convert font numbers into names.

30.7.1 Detailed Description

[Fl](#) class events handling API declared in <[FL/Fl.H](#)>

30.7.2 Function Documentation

30.7.2.1 add_handler()

```
void Fl::add_handler (
    Fl_Event_Handler ha ) [static]
```

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.
- 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\(int, Fl_Window*\)](#)

30.7.2.2 add_system_handler()

```
void Fl::add_system_handler (
    Fl_System_Handler ha,
    void * data ) [static]
```

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`

Parameters

<code>ha</code>	The event handler function to register
<code>data</code>	User data to include on each call

See also

[Fl::remove_system_handler\(Fl_System_Handler\)](#)

30.7.2.3 belowmouse() [1/2]

```
static Fl_Widget* Fl::belowmouse ( ) [inline], [static]
```

Gets the widget that is below the mouse.

See also

[belowmouse\(Fl_Widget*\)](#)

30.7.2.4 belowmouse() [2/2]

```
void Fl::belowmouse (
    Fl_Widget * o ) [static]
```

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\(\)](#)).

30.7.2.5 compose()

```
int Fl::compose (
    int & del ) [static]
```

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.

On the Mac OS platform, text input can involve marked text, that is, temporary text replaced by other text during the input process. This occurs, e.g., 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_marked_text()` when processing `FL_UNFOCUS` events. Optionally, widgets can also call `void Fl::insertion_point_location(int x, int y, int height)` 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 appear just below the insertion point. If widgets don't do that, auxiliary windows appear at the widget's bottom. 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_ACCENT_S_MENU)` ; in their constructor if they want to use the feature introduced with Mac OS 10.7 "Lion" where pressing and holding certain keys on the keyboard opens an accented-character menu window.

Though the current implementation returns immediately, future versions may take quite awhile, as they may pop up a window or do other user-interface things to allow characters to be selected.

30.7.2.6 compose_reset()

```
void Fl::compose_reset () [static]
```

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.

30.7.2.7 disable_im()

```
void Fl::disable_im () [static]
```

Disables the system input methods facilities.

See also

[enable_im\(\)](#)

30.7.2.8 enable_im()

```
void Fl::enable_im ( ) [static]
```

Enables the system input methods facilities.

This is the default.

See also

[disable_im\(\)](#)

30.7.2.9 event()

```
static int Fl::event ( ) [inline], [static]
```

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.

30.7.2.10 event_alt()

```
static int Fl::event_alt ( ) [inline], [static]
```

Returns non-zero if the Alt key is pressed.

30.7.2.11 event_button()

```
static int Fl::event_button ( ) [inline], [static]
```

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

<i>FL_LEFT_MOUSE</i>	
<i>FL_MIDDLE_MOUSE</i>	
<i>FL_RIGHT_MOUSE</i>	

See also

[Fl::event_buttons\(\)](#)

30.7.2.12 event_button1()

```
static int Fl::event_button1 ( ) [inline], [static]
```

Returns non-zero if mouse button 1 is currently held down.

For more details, see [Fl::event_buttons\(\)](#).

30.7.2.13 event_button2()

```
static int Fl::event_button2 ( ) [inline], [static]
```

Returns non-zero if button 2 is currently held down.

For more details, see [Fl::event_buttons\(\)](#).

30.7.2.14 event_button3()

```
static int Fl::event_button3 ( ) [inline], [static]
```

Returns non-zero if button 3 is currently held down.

For more details, see [Fl::event_buttons\(\)](#).

30.7.2.15 event_buttons()

```
static int Fl::event_buttons ( ) [inline], [static]
```

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_BUTTON3] }

30.7.2.16 event_clicks() [1/2]

```
static int Fl::event_clicks ( ) [inline], [static]
```

Returns non zero if we had a double click event.

Return values

<i>Non-zero</i>	if the most recent FL_PUSH or FL_KEYBOARD was a "double click".
<i>N-1</i>	for N clicks. A double click is counted if the same button is pressed again while event_is_click() is true.

30.7.2.17 event_clicks() [2/2]

```
static void Fl::event_clicks (
    int i ) [inline], [static]
```

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\(\)](#)

30.7.2.18 event_clipboard()

```
static void* Fl::event_clipboard () [inline], [static]
```

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](#).

30.7.2.19 event_clipboard_type()

```
static const char* Fl::event_clipboard_type () [inline], [static]
```

Returns the type of the pasted data during an FL_PASTE event.

This type can be [Fl::clipboard_plain_text](#) or [Fl::clipboard_image](#).

30.7.2.20 event_command()

```
static int Fl::event_command () [inline], [static]
```

Returns non-zero if the FL_COMMAND key is pressed, either FL_CTRL or on OSX FL_META.

30.7.2.21 event_ctrl()

```
static int Fl::event_ctrl ( ) [inline], [static]
```

Returns non-zero if the Control key is pressed.

30.7.2.22 event_dispatch()

```
void Fl::event_dispatch (
    Fl_Event_Dispatch d ) [static]
```

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

<code>d</code>	new dispatch function, or NULL
----------------	--------------------------------

See also

[Fl::add_handler\(Fl_Event_Handler\)](#)
[Fl::handle\(int, Fl_Window*\)](#)
[Fl::handle_\(int, Fl_Window*\)](#)

30.7.2.23 event_dx()

```
static int Fl::event_dx ( ) [inline], [static]
```

Returns the current horizontal mouse scrolling associated with the `FL_MOUSEWHEEL` event.

Right is positive.

30.7.2.24 event_dy()

```
static int Fl::event_dy ( ) [inline], [static]
```

Returns the current vertical mouse scrolling associated with the FL_MOUSEWHEEL event.

Down is positive.

30.7.2.25 event_inside() [1/2]

```
int Fl::event_inside (
    int xx,
    int yy,
    int ww,
    int hh ) [static]
```

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

30.7.2.26 event_inside() [2/2]

```
int Fl::event_inside (
    const Fl_Widget * o ) [static]
```

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	<i>o</i>	child widget to be tested
----	----------	---------------------------

Returns

non-zero, if mouse event is inside the widget

30.7.2.27 event_is_click() [1/2]

```
static int Fl::event_is_click ( ) [inline], [static]
```

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.

30.7.2.28 event_is_click() [2/2]

```
static void Fl::event_is_click (
    int i ) [inline], [static]
```

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.

30.7.2.29 event_key() [1/2]

```
static int Fl::event_key ( ) [inline], [static]
```

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\(int&\)](#).

30.7.2.30 event_key() [2/2]

```
int Fl::event_key (
    int key ) [static]
```

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. 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.

30.7.2.31 event_length()

```
static int Fl::event_length ( ) [inline], [static]
```

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.

30.7.2.32 event_original_key()

```
static int Fl::event_original_key ( ) [inline], [static]
```

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.

30.7.2.33 event_shift()

```
static int Fl::event_shift ( ) [inline], [static]
```

Returns non-zero if the Shift key is pressed.

30.7.2.34 event_state() [1/2]

```
static int Fl::event_state ( ) [inline], [static]
```

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.

The legal event state bits are:

- FL_SHIFT
- FL_CAPS_LOCK
- FL_CTRL
- FL_ALT
- FL_NUM_LOCK
- FL_META
- FL_SCROLL_LOCK
- FL_BUTTON1
- FL_BUTTON2
- FL_BUTTON3

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. In addition there is a bug in the way X works so that the shift state is not correctly reported until the first event *after* the shift key is pressed or released.

30.7.2.35 event_state() [2/2]

```
static int Fl::event_state (
    int mask ) [inline], [static]
```

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\(\)](#).

30.7.2.36 event_text()

```
static const char* Fl::event_text ( ) [inline], [static]
```

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.

30.7.2.37 event_x_root()

```
static int Fl::event_x_root ( ) [inline], [static]
```

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\(\)](#).

30.7.2.38 event_y_root()

```
static int Fl::event_y_root ( ) [inline], [static]
```

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\(\)](#).

30.7.2.39 focus() [1/2]

```
static Fl_Widget* Fl::focus ( ) [inline], [static]
```

Gets the current [Fl::focus\(\)](#) widget.

See also

[Fl::focus\(Fl_Widget*\)](#)

30.7.2.40 focus() [2/2]

```
void Fl::focus (
    Fl_Widget * o ) [static]
```

Sets the widget that will receive FL_KEYBOARD events.

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\(\)](#)).

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\(\)](#)

30.7.2.41 get_key()

```
int Fl::get_key (
    int key ) [static]
```

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\(int\)](#)

30.7.2.42 get_mouse()

```
void Fl::get_mouse (
    int & x,
    int & y ) [static]
```

Return where the mouse is on the screen by doing 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.

30.7.2.43 handle()

```
int Fl::handle (
    int e,
    Fl_Window * window ) [static]
```

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

<i>e</i>	the event type (<code>Fl::event_number()</code> is not yet set)
<i>window</i>	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\(Fl_Event_Dispatch\)](#)

30.7.2.44 handle_()

```
int Fl::handle_
    (int e,
     Fl_Window * window) [static]
```

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_\(int, Fl_Window*\)](#)

Parameters

<i>e</i>	the event type (<code>Fl::event_number()</code> is not yet set)
<i>window</i>	the window that caused this event

Returns

0 if the event was not handled

See also

[Fl::event_dispatch\(Fl_Event_Dispatch\)](#)

30.7.2.45 pushed() [1/2]

```
static Fl_Widget* Fl::pushed () [inline], [static]
```

Gets the widget that is being pushed.

See also

[void pushed\(Fl_Widget*\)](#)

30.7.2.46 pushed() [2/2]

```
void Fl::pushed (
    Fl_Widget * o ) [static]
```

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\(\)](#)).

30.7.2.47 remove_handler()

```
void Fl::remove_handler (
    Fl_Event_Handler ha ) [static]
```

Removes a previously added event handler.

See also

[Fl::handle\(int, Fl_Window*\)](#)

30.7.2.48 remove_system_handler()

```
void Fl::remove_system_handler (
    Fl_System_Handler ha ) [static]
```

Removes a previously added system event handler.

Parameters

<i>ha</i>	The event handler function to remove
-----------	--------------------------------------

See also

[Fl::add_system_handler\(Fl_System_Handler\)](#)

30.7.2.49 test_shortcut()

```
int Fl::test_shortcut (
    Fl_Shortcut shortcut ) [static]
```

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.

30.7.3 Variable Documentation

30.7.3.1 fl_eventnames

```
const char* const fl_eventnames[]
```

This is an array of event names you can use to convert event 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::handle(int e) {
    printf("Event was %s (%d)\n", fl_eventnames[e], e);
    // ..resulting output might be e.g. "Event was FL_PUSH (1)...
    [...]
}
```

30.7.3.2 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)...
    [...]
}
```

30.8 Selection & Clipboard functions

FLTK global copy/cut/paste functions declared in <[FL/FL.H](#)>

Functions

- static 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.
- static int [Fl::clipboard_contains](#) (const char *type)
Returns non 0 if the clipboard contains data matching type.
- static void [Fl::copy](#) (const char *stuff, int len, int destination=0, const char *type=[Fl::clipboard_plain_text](#))
Copies the data pointed to by stuff to the selection buffer (destination is 0), the clipboard (destination is 1), or both (destination is 2).
- static int [Fl::dnd](#) ()
Initiate a Drag And Drop operation.
- static 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.
- static void [Fl::paste](#) ([Fl_Widget](#) &receiver)
Backward compatibility only.
- static 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.
- static void [Fl::selection](#) ([Fl_Widget](#) &owner, const char *, int len)
Changes the current selection.
- static [Fl_Widget](#) * [Fl::selection_owner](#) ()
back-compatibility only: Gets the widget owning the current selection
- static 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.

Variables

- static char const *const [Fl::clipboard_image](#) = "image"
Denotes image data.
- static char const *const [Fl::clipboard_plain_text](#) = "text/plain"
Denotes plain textual data.

30.8.1 Detailed Description

FLTK global copy/cut/paste functions declared in <[FL/FL.H](#)>

30.8.2 Function Documentation

30.8.2.1 add_clipboard_notify()

```
void Fl::add_clipboard_notify (
    Fl_Clipboard_Notify_Handler h,
    void * data = 0 ) [static]
```

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.

30.8.2.2 clipboard_contains()

```
int Fl::clipboard_contains (
    const char * type ) [static]
```

Returns non 0 if the clipboard contains data matching `type`.

`type` can be `Fl::clipboard_plain_text` or `Fl::clipboard_image`.

30.8.2.3 copy()

```
void Fl::copy (
    const char * stuff,
    int len,
    int destination = 0,
    const char * type = Fl::clipboard_plain_text ) [static]
```

Copies the data pointed to by `stuff` to the selection buffer (`destination` is 0), the clipboard (`destination` is 1), or both (`destination` is 2).

Copying to both is only relevant on X11, on other platforms it maps to the clipboard (1). `len` is the number of relevant bytes in `stuff`. `type` is always `Fl::clipboard_plain_text`. The selection buffer is used for middle-mouse pastes and for drag-and-drop selections. The clipboard is used for traditional copy/cut/paste operations.

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 in the future to copy other kinds of data.

30.8.2.4 dnd()

```
int Fl::dnd ( ) [static]
```

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)

30.8.2.5 paste() [1/2]

```
void Fl::paste (
    Fl_Widget & receiver,
    int source,
    const char * type = Fl::clipboard_plain_text ) [static]
```

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. If 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.

30.8.2.6 `paste()` [2/2]

```
void Fl::paste (
    Fl_Widget & receiver )  [static]
```

Backward compatibility only.

This calls `Fl::paste(receiver, 0);`

See also

[Fl::paste\(Fl_Widget &receiver, int clipboard, const char* type\)](#)

30.8.2.7 `selection()`

```
void Fl::selection (
    Fl_Widget & owner,
    const char * text,
    int len )  [static]
```

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.

30.8.2.8 `selection_owner()` [1/2]

```
static Fl_Widget* Fl::selection_owner ( )  [inline], [static]
```

back-compatibility only: Gets the widget owning the current selection

See also

[Fl_Widget* selection_owner\(Fl_Widget*\)](#)

30.8.2.9 `selection_owner()` [2/2]

```
void Fl::selection_owner (
    Fl_Widget * owner )  [static]
```

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.

30.9 Screen functions

Fl global screen functions declared in <FL/Fl.H>.

Functions

- static int `Fl::h ()`
Returns the height in pixels of the main screen work area.
- static void `Fl::keyboard_screen_scaling (int value)`
Controls the possibility to scale all windows by ctrl/+/-/0/ or cmd/+/-/0/.
- static int `Fl::screen_count ()`
Gets the number of available screens.
- static void `Fl::screen_dpi (float &h, float &v, int n=0)`
Gets the screen resolution in dots-per-inch for the given screen.
- static int `Fl::screen_num (int x, int y)`
Gets the screen number of a screen that contains the specified screen position x, y.
- static int `Fl::screen_num (int x, int y, int w, int h)`
Gets the screen number for the screen which intersects the most with the rectangle defined by x, y, w, h.
- static float `Fl::screen_scale (int n)`
Current value of the GUI scaling factor for screen number n.
- static void `Fl::screen_scale (int n, float factor)`
Set the value of the GUI scaling factor for screen number n.
- static int `Fl::screen_scaling_supported ()`
See if scaling factors are supported by this platform.
- static 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.
- static 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.
- static 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.
- static void `Fl::screen_xywh (int &X, int &Y, int &W, int &H)`
Gets the bounding box of a screen that contains the mouse pointer.
- static 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.
- static void `Fl::screen_xywh (int &X, int &Y, int &W, int &H, int n)`
Gets the screen bounding rect for the given screen.
- static 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.
- static int `Fl::w ()`
Returns the width in pixels of the main screen work area.
- static int `Fl::x ()`
Returns the leftmost x coordinate of the main screen work area.
- static int `Fl::y ()`
Returns the topmost y coordinate of the main screen work area.

30.9.1 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 use as initial scaling factor. The scaling factor value can be further changed at runtime by typing `ctrl-+/-0/` (`cmd-+/-0/` under MacOS). FLTK sends the `FL_ZOOM_EVENT` when the factor value is changed, to which a callback can be associated with `Fl::add_handler()`. By default, FLTK also displays the new scaling factor value in a yellow, transient window. This can be changed with option `Fl::OPTION_SHOW_SCALING`. The scaling factor value is programmatically get and set with the `Fl::screen_scale()` functions.

30.9.2 Function Documentation

30.9.2.1 h()

```
int Fl::h ( ) [static]
```

Returns the height in pixels of the main screen work area.

30.9.2.2 keyboard_screen_scaling()

```
void Fl::keyboard_screen_scaling (
    int value ) [static]
```

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

<code>value</code>	0 to stop recognition of <code>ctrl-+/-0/</code> (or <code>cmd-+/-0/</code> under macOS) keys as window scaling.
--------------------	--

30.9.2.3 screen_dpi()

```
void Fl::screen_dpi (
```

```
float & h,
float & v,
int n = 0 ) [static]
```

Gets the screen resolution in dots-per-inch for the given screen.

Parameters

out	<i>h,v</i>	horizontal and vertical resolution
in	<i>n</i>	the screen number (0 to Fl::screen_count() - 1)

See also

[void screen_xywh\(int &x, int &y, int &w, int &h, int mx, int my\)](#)

30.9.2.4 screen_num() [1/2]

```
int Fl::screen_num (
    int x,
    int y ) [static]
```

Gets the screen number of a screen that contains the specified screen position *x, y*.

Parameters

in	<i>x,y</i>	the absolute screen position
----	------------	------------------------------

30.9.2.5 screen_num() [2/2]

```
int Fl::screen_num (
    int x,
    int y,
    int w,
    int h ) [static]
```

Gets the screen number for the screen which intersects the most with the rectangle defined by *x, y, w, h*.

Parameters

in	<i>x,y,w,h</i>	the rectangle to search for intersection with
----	----------------	---

30.9.2.6 screen_scale()

```
void Fl::screen_scale (
    int n,
    float factor ) [static]
```

Set the value of the GUI scaling factor for screen number *n*.

When this function is called before the first window is show()'n it sets the application's initial scaling factor value. Otherwise, it sets the scale factor value of all windows mapped to screen number *n*

30.9.2.7 screen_scaling_supported()

```
int Fl::screen_scaling_supported () [static]
```

See if scaling factors are supported by this platform.

Returns

0 if scaling factors are not supported by this platform, 1 if a single scaling factor value is shared by all screens,
2 if each screen can have its own scaling factor value.

See also

[Fl::screen_scale\(int\)](#)

30.9.2.8 screen_work_area() [1/3]

```
void Fl::screen_work_area (
    int & X,
    int & Y,
    int & W,
    int & H,
    int mx,
    int my ) [static]
```

Gets the bounding box of the work area of a screen that contains the specified screen position *mx, my*.

Parameters

out	<i>X,Y,W,H</i>	the work area bounding box
in	<i>mx,my</i>	the absolute screen position

30.9.2.9 screen_work_area() [2/3]

```
void Fl::screen_work_area (
```

```
int & X,
int & Y,
int & W,
int & H,
int n ) [static]
```

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\(int &x, int &y, int &w, int &h, int mx, int my\)](#)

30.9.2.10 screen_work_area() [3/3]

```
void Fl::screen_work_area (
    int & X,
    int & Y,
    int & W,
    int & H ) [static]
```

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\(int &x, int &y, int &w, int &h, int mx, int my\)](#)

30.9.2.11 screen_xywh() [1/4]

```
void Fl::screen_xywh (
    int & X,
    int & Y,
    int & W,
    int & H ) [static]
```

Gets the bounding box of a screen that contains the mouse pointer.

Parameters

<code>out</code>	<code>X,Y,W,H</code>	the corresponding screen bounding box
------------------	----------------------	---------------------------------------

See also

`void screen_xywh(int &x, int &y, int &w, int &h, int mx, int my)`

30.9.2.12 screen_xywh() [2/4]

```
void Fl::screen_xywh (
    int & X,
    int & Y,
    int & W,
    int & H,
    int mx,
    int my )  [static]
```

Gets the bounding box of a screen that contains the specified screen position `mx, my`.

Parameters

<code>out</code>	<code>X,Y,W,H</code>	the corresponding screen bounding box
<code>in</code>	<code>mx,my</code>	the absolute screen position

30.9.2.13 screen_xywh() [3/4]

```
void Fl::screen_xywh (
    int & X,
    int & Y,
    int & W,
    int & H,
    int n )  [static]
```

Gets the screen bounding rect for the given screen.

Under Windows, Mac OS X, and the Gnome desktop, screen #0 contains the menubar/taskbar

Parameters

<code>out</code>	<code>X,Y,W,H</code>	the corresponding screen bounding box
<code>in</code>	<code>n</code>	the screen number (0 to <code>Fl::screen_count()</code> - 1)

See also

[void screen_xywh\(int &x, int &y, int &w, int &h, int mx, int my\)](#)

30.9.2.14 screen_xywh() [4/4]

```
void Fl::screen_xywh (
    int & X,
    int & Y,
    int & W,
    int & H,
    int mx,
    int my,
    int mw,
    int mh )  [static]
```

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\(int &X, int &Y, int &W, int &H, int n\)](#)

30.9.2.15 w()

```
int Fl::w ( )  [static]
```

Returns the width in pixels of the main screen work area.

30.9.2.16 x()

```
int Fl::x ( )  [static]
```

Returns the leftmost x coordinate of the main screen work area.

30.9.2.17 y()

```
int Fl::y ( )  [static]
```

Returns the topmost y coordinate of the main screen work area.

30.10 Color & Font functions

fl global color, font functions.

Functions

- void `fl_color (Fl_Color c)`
Sets 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)`
Sets the color for all subsequent drawing operations.
- `Fl_Color fl_color ()`
Returns the last `fl_color()` that was set.
- `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)`
Returns a color that contrasts with the background color.
- int `fl_descent ()`
Returns 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_EXPORT void `fl_font (Fl_Font face, Fl_Fontsize fsize)`
Sets the current font, which is then used in various drawing routines.
- `Fl_Font fl_font ()`
Returns the `face` set by the most recent call to `fl_font()`.
- int `fl_height ()`
Returns the recommended minimum line spacing for the current font.
- FL_EXPORT 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.
- FL_EXPORT const char * `fl_latin1_to_local (const char *t, int n=-1)`
Converts text from Windows/X11 latin1 character set to local encoding.
- FL_EXPORT const char * `fl_local_to_latin1 (const char *t, int n=-1)`
Converts text from local encoding to Windowx/X11 latin1 character set.
- FL_EXPORT const char * `fl_local_to_mac_roman (const char *t, int n=-1)`
Converts text from local encoding to Mac Roman character set.
- FL_EXPORT const char * `fl_mac_roman_to_local (const char *t, int n=-1)`
Converts text from Mac Roman character set to local encoding.
- FL_EXPORT `Fl_Color fl_show_colormap (Fl_Color oldcol)`
Pops up a window to let the user pick a colormap entry.
- `Fl_Fontsize fl_size ()`
Returns the `size` set by the most recent call to `fl_font()`.
- FL_EXPORT void `fl_text_extents (const char *, int &dx, int &dy, int &w, int &h)`
Determines 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)`
Determines the minimum pixel dimensions of a sequence of `n` characters using the current `fl_font()`.
- FL_EXPORT double `fl_width (const char *txt)`
Returns the typographical width of a nul-terminated string using the current font face and size.
- double `fl_width (const char *txt, int n)`

- **double fl_width (unsigned int c)**
Returns the typographical width of a sequence of n characters using the current font face and size.
- **static void Fl::free_color (Fl_Color i, int overlay=0)**
Frees the specified color from the colormap, if applicable.
- **static unsigned Fl::get_color (Fl_Color i)**
Returns the RGB value(s) for the given FLTK color index.
- **static void Fl::get_color (Fl_Color i, uchar &red, uchar &green, uchar &blue)**
Returns the RGB value(s) for the given FLTK color index.
- **static const char * Fl::get_font (Fl_Font)**
Gets the string for this face.
- **static const char * Fl::get_font_name (Fl_Font, int *attributes=0)**
Get a human-readable string describing the family of this face.
- **static int Fl::get_font_sizes (Fl_Font, int *&sizep)**
Return an array of sizes in sizep.
- **static void Fl::set_color (Fl_Color, uchar, uchar, uchar)**
Sets an entry in the fl_color index table.
- **static void Fl::set_color (Fl_Color i, unsigned c)**
Sets an entry in the fl_color index table.
- **static void Fl::set_font (Fl_Font, const char *)**
Changes a face.
- **static void Fl::set_font (Fl_Font, Fl_Font)**
Copies one face to another.
- **static Fl_Font Fl::set_fonts (const char *=0)**
FLTK will open the display, and add every fonts on the server to the face table.

30.10.1 Detailed Description

fl global color, font functions.

These functions are declared in <FL/Fl.H> or <FL/fl_draw.H>.

30.10.2 Function Documentation

30.10.2.1 fl_color() [1/3]

```
void fl_color (
    Fl_Color c ) [inline]
```

Sets the color for all subsequent drawing operations.

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. If no valid graphical context (fl_gc) is available, the foreground is not set for the current window.

Parameters

in	c	color
----	---	-------

30.10.2.2 fl_color() [2/3]

```
void fl_color (
    uchar r,
    uchar g,
    uchar b ) [inline]
```

Sets 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
----	-------	------------------

30.10.2.3 fl_color() [3/3]

```
F1_Color fl_color ( ) [inline]
```

Returns the last [fl_color\(\)](#) that was set.

This can be used for state save/restore.

30.10.2.4 fl_color_average()

```
F1_Color fl_color_average (
    F1_Color color1,
    F1_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	<i>color1,color2</i>	boundary colors
in	<i>weight</i>	weighting factor

30.10.2.5 fl_contrast()

```
Fl_Color fl_contrast (
    Fl_Color fg,
    Fl_Color bg )
```

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.

Parameters

in	<i>fg,bg</i>	foreground and background colors
----	--------------	----------------------------------

Returns

contrasting color

30.10.2.6 fl_font() [1/2]

```
FL_EXPORT 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.

30.10.2.7 fl_font() [2/2]

```
Fl_Font fl_font ( ) [inline]
```

Returns the `face` set by the most recent call to [fl_font\(\)](#).

This can be used to save/restore the font.

30.10.2.8 `fl_height()` [1/2]

```
int fl_height ( ) [inline]
```

Returns the recommended minimum line spacing for the current font.

You can also use the value of `size` passed to [fl_font\(\)](#)

30.10.2.9 `fl_height()` [2/2]

```
FL_EXPORT 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 [FI_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.

30.10.2.10 `fl_latin1_to_local()`

```
FL_EXPORT const char* fl_latin1_to_local (
    const char * t,
    int n = -1 )
```

Converts text from Windows/X11 latin1 character set to local encoding.

Parameters

in	<code>t</code>	character string (latin1 encoding)
in	<code>n</code>	optional number of characters to convert (default is all)

Returns

pointer to internal buffer containing converted characters

30.10.2.11 fl_local_to_latin1()

```
FL_EXPORT const char* fl_local_to_latin1 (
    const char * t,
    int n = -1 )
```

Converts text from local encoding to Windowx/X11 latin1 character set.

Parameters

in	<i>t</i>	character string (local encoding)
in	<i>n</i>	optional number of characters to convert (default is all)

Returns

pointer to internal buffer containing converted characters

30.10.2.12 fl_local_to_mac_roman()

```
FL_EXPORT const char* fl_local_to_mac_roman (
    const char * t,
    int n = -1 )
```

Converts text from local encoding to Mac Roman character set.

Parameters

in	<i>t</i>	character string (local encoding)
in	<i>n</i>	optional number of characters to convert (default is all)

Returns

pointer to internal buffer containing converted characters

30.10.2.13 fl_mac_roman_to_local()

```
FL_EXPORT const char* fl_mac_roman_to_local (
    const char * t,
    int n = -1 )
```

Converts text from Mac Roman character set to local encoding.

Parameters

in	<i>t</i>	character string (Mac Roman encoding)
in	<i>n</i>	optional number of characters to convert (default is all)

Returns

pointer to internal buffer containing converted characters

30.10.2.14 fl_show_colormap()

```
FL_EXPORT Fl_Color fl_show_colormap (
    Fl_Color oldcol )
```

Pops up a window to let the user pick a colormap entry.

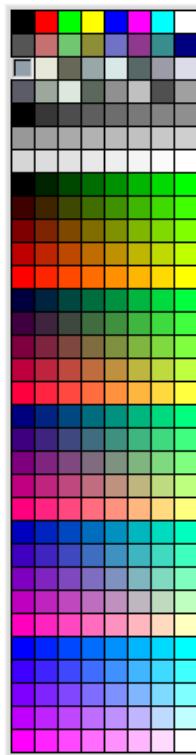


Figure 30.1 fl_show_colormap

Parameters

in	<i>oldcol</i>	color to be highlighted when grid is shown.
----	---------------	---

Return values

<i>Fl_Color</i>	value of the chosen colormap entry.
-----------------	-------------------------------------

See also

[Fl_Color_Chooser](#)

30.10.2.15 fl_size()

```
FL_Fontsize fl_size ( ) [inline]
```

Returns the size set by the most recent call to [fl_font\(\)](#).

This can be used to save/restore the font.

30.10.2.16 fl_text_extents() [1/2]

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

Determines 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.

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+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);
:
```

30.10.2.17 fl_text_extents() [2/2]

```
void fl_text_extents (
    const char * t,
    int n,
    int & dx,
    int & dy,
    int & w,
    int & h ) [inline]
```

Determines the minimum pixel dimensions of a sequence of n characters using the current [fl_font\(\)](#).

See also

[fl_text_extents\(const char*, int& dx, int& dy, int& w, int& h\)](#)

30.10.2.18 fl_width() [1/3]

```
FL_EXPORT double fl_width (
    const char * txt )
```

Returns the typographical width of a nul-terminated string using the current font face and size.

30.10.2.19 fl_width() [2/3]

```
double fl_width (
    const char * txt,
    int n ) [inline]
```

Returns the typographical width of a sequence of *n* characters using the current font face and size.

30.10.2.20 fl_width() [3/3]

```
double fl_width (
    unsigned int c ) [inline]
```

Returns the typographical width of a single character using the current font face and size.

Note

if a valid fl_gc is NOT found then it uses the first window gc, or the screen gc if no fltk window is available when called.

30.10.2.21 free_color()

```
void Fl::free_color (
    Fl_Color i,
    int overlay = 0 ) [static]
```

Frees the specified color from the colormap, if applicable.

If overlay is non-zero then the color is freed from the overlay colormap.

30.10.2.22 get_color() [1/2]

```
unsigned Fl::get_color (
    Fl_Color i ) [static]
```

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.

30.10.2.23 `get_color()` [2/2]

```
void Fl::get_color (
    Fl_Color i,
    uchar & red,
    uchar & green,
    uchar & blue ) [static]
```

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\(Fl_Color c\)](#)

30.10.2.24 `get_font()`

```
const char * Fl::get_font (
    Fl_Font fnum ) [static]
```

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.

30.10.2.25 `get_font_name()`

```
const char * Fl::get_font_name (
    Fl_Font fnum,
    int * attributes = 0 ) [static]
```

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.

30.10.2.26 `get_font_sizes()`

```
int Fl::get_font_sizes (
    Fl_Font fnum,
    int *& sizep ) [static]
```

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.

30.10.2.27 set_color() [1/2]

```
void Fl::set_color (
    Fl_Color i,
    uchar red,
    uchar green,
    uchar blue )  [static]
```

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.

30.10.2.28 set_color() [2/2]

```
void Fl::set_color (
    Fl_Color i,
    unsigned c )  [static]
```

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.

30.10.2.29 set_font() [1/2]

```
void Fl::set_font (
    Fl_Font fnum,
    const char * name )  [static]
```

Changes a face.

The string pointer is simply stored, the string is not copied, so the string must be in static memory.

30.10.2.30 set_font() [2/2]

```
void Fl::set_font (
    Fl_Font fnum,
    Fl_Font from )  [static]
```

Copies one face to another.

30.10.2.31 set_fonts()

```
Fl_Font Fl::set_fonts (
    const char * xstarname = 0 )  [static]
```

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 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 Windows NULL selects fonts with ISO8859-1 encoding and non-NUL selects all fonts.

The return value is how many faces are in the table after this is done.

30.11 Drawing functions

FLTK global graphics and GUI drawing functions.

Macros

- `#define fl_clip fl_push_clip`

Intersects the current clip region with a rectangle and pushes this new region onto the stack (deprecated).

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 }`

Functions

- `FL_EXPORT int fl_add_symbol (const char *name, void(*drawit)(FL_Color), int scalable)`

Adds a symbol to the system.
- `void fl_arc (int x, int y, int w, int h, double a1, double a2)`

Draw ellipse sections using integer coordinates.
- `void fl_arc (double x, double y, double r, double start, double end)`

Adds a series of points to the current path on the arc of a circle.
- `void fl_begin_complex_polygon ()`

Starts drawing a complex filled polygon.
- `void fl_begin_line ()`

Starts drawing a list of lines.
- `void fl_begin_loop ()`

Starts drawing a closed sequence of lines.
- `FL_EXPORT void fl_begin_offscreen (FL_Offscreen ctx)`

Send all subsequent drawing commands to this offscreen buffer.
- `void fl_begin_points ()`

Starts drawing a list of points.
- `void fl_begin_polygon ()`

Starts drawing a convex filled polygon.
- `char fl_can_do_alpha_blending ()`

Checks whether platform supports true alpha blending for RGBA images.
- `FL_EXPORT FL_RGB_Image * fl_capture_window_part (FL_Window *win, int x, int y, int w, int h)`

Captures the content of a rectangular zone of a mapped window.
- `FL_EXPORT 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)`

Intersects a rectangle with the current clip region and returns the bounding box of the result.
- `void fl_clip_region (FL_Region r)`

Replaces the top of the clipping stack with a clipping region of any shape.
- `FL_Region fl_clip_region ()`

- **void fl_copy_offscreen (int x, int y, int w, int h, FL_Offscreen pixmap, int srcx, int srcy)**

Returns the current clipping region.
- **FL_EXPORT FL_Offscreen fl_create_offscreen (int w, int h)**

Copy a rectangular area of the given offscreen buffer into the current drawing destination.
- **FL_EXPORT void fl_cursor (FL_Cursor)**

Creation of an offscreen graphics buffer.
- **FL_EXPORT void fl_curve (double X0, double Y0, double X1, double Y1, double X2, double Y2, double X3, double Y3)**

Adds a series of points on a Bezier curve to the path.
- **FL_EXPORT void fl_delete_offscreen (FL_Offscreen ctx)**

Deletion of an offscreen graphics buffer.
- **FL_EXPORT void fl_draw (const char *str, int x, int y)**

Draws a nul-terminated UTF-8 string starting at the given x, y location.
- **FL_EXPORT void fl_draw (int angle, const char *str, int x, int y)**

Draws a nul-terminated UTF-8 string starting at the given x, y location and rotating angle degrees counter-clockwise.
- **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 (int angle, const char *str, int n, int x, int y)**

Draws at the given x, y location a UTF-8 string of length n bytes rotating angle degrees counter-clockwise.
- **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)**

Fancy string drawing function which is used to draw all the labels.
- **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)**

The same as `fl_draw(const char,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.*
- **FL_EXPORT 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_image (const uchar *buf, int X, int Y, int W, int H, int D=3, int L=0)**

Draws 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)**

Draws 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)**

Draws 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)**

Draws a gray-scale image using a callback function to generate image data.
- **FL_EXPORT int fl_draw_pixmap (const 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 (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.
- **FL_EXPORT 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 ()**

Ends complex filled polygon, and draws.
- **void fl_end_line ()**

Ends list of lines, and draws.
- **void fl_end_loop ()**

Ends closed sequence of lines, and draws.
- **FL_EXPORT void fl_end_offscreen ()**

- Quit sending drawing commands to the current offscreen buffer.*
- void [fl_end_points \(\)](#)
Ends list of points, and draws.
 - void [fl_end_polygon \(\)](#)
Ends convex filled polygon, and draws.
 - 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)
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_EXPORT void [fl_frame](#) (const char *s, int x, int y, int w, int h)
Draws a series of line segments around the given box.
 - FL_EXPORT 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 \(\)](#)
Call fl_gap() to separate loops of the path.
 - void [fl_line](#) (int x, int y, int x1, int y1)
Draws a line from (x,y) to (x1,y1)
 - void [fl_line](#) (int x, int y, int x1, int y1, int x2, int y2)
Draws 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)
Sets how to draw lines (the "pen").
 - void [fl_loop](#) (int x, int y, int x1, int y1, int x2, int y2)
Outlines 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)
Outlines a 4-sided polygon with lines.
 - FL_EXPORT 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.
 - FL_EXPORT int [fl_measure_pixmap](#) (char *const *data, int &w, int &h)
Get the dimensions of a pixmap.
 - FL_EXPORT 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)
Concatenates 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?
 - FL_EXPORT unsigned int [fl_old_shortcut](#) (const char *s)
Emulation of XForms named shortcuts.
 - FL_EXPORT void [fl_overlay_clear \(\)](#)
Erase a selection rectangle without drawing a new one.
 - FL_EXPORT void [fl_overlay_rect](#) (int x, int y, int w, int h)
Draws a selection rectangle, erasing a previous one by XOR'ing it first.
 - 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)
Draws a single pixel at the given coordinates.
 - void [fl_polygon](#) (int x, int y, int x1, int y1, int x2, int y2)
Fills a 3-sided polygon.
 - void [fl_polygon](#) (int x, int y, int x1, int y1, int x2, int y2, int x3, int y3)
Fills a 4-sided polygon.
 - void [fl_pop_clip \(\)](#)

- `void fl_pop_matrix ()`
Restores the previous clip region.
- `void fl_push_clip (int x, int y, int w, int h)`
Restores the current transformation matrix from the stack.
- `void fl_push_matrix ()`
Intersects the current clip region with a rectangle and pushes this new region onto the stack.
- `void fl_push_no_clip ()`
Saves the current transformation matrix on the stack.
- `void fl_rect (int x, int y, int w, int h)`
Pushes an empty clip region onto the stack so nothing will be clipped.
- `FL_EXPORT 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 (int x, int y, int w, int h)`
Draws a 1-pixel border inside the given bounding box.
- `void fl_rect (int x, int y, int w, int h, FL_Color c)`
Draws with passed color a 1-pixel border inside the given bounding box.
- `void fl_rectf (int x, int y, int w, int h)`
Colors 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)`
Colors with passed color a rectangle that exactly fills the given bounding box.
- `FL_EXPORT void fl_rectf (int x, int y, int w, int h, uchar r, uchar g, uchar b)`
Colors a rectangle with "exactly" the passed r, g, b color.
- `FL_EXPORT void fl_rescale_offscreen (FL_Offscreen &ctx)`
Adapts an offscreen buffer to a changed value of the scale factor.
- `FL_EXPORT void fl_reset_spot (void)`
- `void fl_restore_clip ()`
Undoes any clobbering of clip done by your program.
- `void fl_rotate (double d)`
Concatenates rotation transformation onto the current one.
- `void fl_rtl_draw (const char *str, int n, int x, int y)`
Draws a UTF-8 string of length n bytes right to left starting at the given x, y location.
- `void fl_scale (double x, double y)`
Concatenates scaling transformation onto the current one.
- `void fl_scale (double x)`
Concatenates scaling transformation onto the current one.
- `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)`
Scroll a rectangle and draw the newly exposed portions.
- `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_set_status (int X, int Y, int W, int H)`
- `FL_EXPORT const char * fl_shortcut_label (unsigned int shortcut)`
Get a human-readable string from a shortcut value.
- `FL_EXPORT 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)`
Transforms distance using current transformation matrix.
- `double fl_transform_dy (double x, double y)`
Transforms distance using current transformation matrix.
- `double fl_transform_x (double x, double y)`
Transforms coordinate using the current transformation matrix.
- `double fl_transform_y (double x, double y)`
Transforms coordinate using the current transformation matrix.

- void [fl_transformed_vertex](#) (double xf, double yf)
Adds coordinate pair to the vertex list without further transformations.
- void [fl_translate](#) (double x, double y)
Concatenates translation transformation onto the current one.
- void [fl_vertex](#) (double x, double y)
Adds a single vertex to the current path.
- void [fl_xyline](#) (int x, int y, int x1)
Draws a horizontal line from (x,y) to (x1,y)
- void [fl_xyline](#) (int x, int y, int x1, int y2)
Draws 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)
Draws 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)
Draws a vertical line from (x,y) to (x,y1)
- void [fl_yxline](#) (int x, int y, int y1, int x2)
Draws 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)
Draws 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)

30.11.1 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.

30.11.2 Macro Definition Documentation

30.11.2.1 fl_clip

```
#define fl_clip fl_push_clip
```

Intersects the current clip region with a rectangle and pushes this new region onto the stack (deprecated).

Parameters

in	x,y,w,h	position and size
----	---------	-------------------

Deprecated `fl_clip(int, int, int, int)` is deprecated and will be removed from future releases. Please use `fl_push_clip(int x, int y, int w, int h)` instead.

30.11.3 Enumeration Type Documentation

30.11.3.1 anonymous enum

```
anonymous enum
```

Enumerator

FL_SOLID	line style: _____
FL_DASH	line style: - - - - -
FL_DOT	line style:
FL_DASHDOT	line style: _ . _ . _ .
FL_DASHDOTDOT	line style: _ . _ . _ . .
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

30.11.4 Function Documentation

30.11.4.1 fl_add_symbol()

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

Adds a symbol to the system.

Parameters

in	<i>name</i>	name of symbol (without the "@")
in	<i>drawit</i>	function to draw symbol
in	<i>scalable</i>	set to 1 if drawit uses scalable vector drawing

Returns

1 on success, 0 on failure

30.11.4.2 fl_arc() [1/2]

```
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_pie()` draws a series of lines to approximate the arc. Notice that the integer version of `fl_pie()` has a different number of arguments than the double version `fl_pie(double x, double y, double r, double start, double end)`

Parameters

in	<code>x,y,w,h</code>	bounding box of complete circle
in	<code>a1,a2</code>	start and end angles of arc measured in degrees counter-clockwise from 3 o'clock. <code>a2</code> must be greater than or equal to <code>a1</code> .

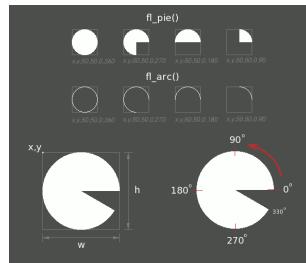


Figure 30.2 `fl_pie()` and `fl_arc()`

30.11.4.3 `fl_arc()` [2/2]

```
void fl_arc (
    double x,
    double y,
    double r,
    double start,
    double end ) [inline]
```

Adds 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()`.

Parameters

in	<i>x,y,r</i>	center and radius of circular arc
in	<i>start,end</i>	angles of start and end of arc measured in degrees counter-clockwise from 3 o'clock. If <i>end</i> is less than <i>start</i> then it draws the arc in a clockwise direction.

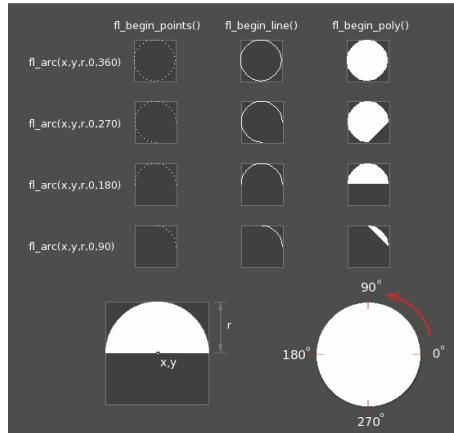


Figure 30.3 fl_arc(x,y,r,a1,a2)

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();
```

30.11.4.4 fl_begin_complex_polygon()

```
void fl_begin_complex_polygon ( ) [inline]
```

Starts 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.

30.11.4.5 fl_begin_offscreen()

```
void fl_begin_offscreen (
    Fl_Offscreen ctx )
```

Send all subsequent drawing commands to this offscreen buffer.

Parameters

<i>ctx</i>	the offscreen buffer.
------------	-----------------------

Note

The *ctx* argument must have been created by [fl_create_offscreen\(\)](#).

30.11.4.6 fl_begin_points()

```
void fl_begin_points ( ) [inline]
```

Starts drawing a list of points.

Points are added to the list with [fl_vertex\(\)](#)

30.11.4.7 fl_can_do_alpha_blending()

```
char fl_can_do_alpha_blending ( ) [inline]
```

Checks 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

30.11.4.8 fl_capture_window_part()

```
FL_EXPORT Fl_RGB_Image* fl_capture_window_part (
    Fl_Window * win,
    int x,
    int y,
    int w,
    int h )
```

Captures the content of a rectangular zone of a mapped window.

Parameters

<code>win</code>	a mapped Fl_Window (derived types including Fl_Gl_Window are also possible)
<code>x,y,w,h</code>	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

30.11.4.9 [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.

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\(\)](#)

Parameters

<code>in</code>	<code>x,y,r</code>	center and radius of circle
-----------------	--------------------	-----------------------------

30.11.4.10 [fl_clip_box\(\)](#)

```
int fl_clip_box (
    int x,
    int y,
    int w,
    int h,
    int & X,
    int & Y,
    int & W,
    int & H ) [inline]
```

Intersects a rectangle with the current clip region and returns 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	<code>x,y,w,h</code>	position and size of rectangle
out	<code>X,Y,W,H</code>	position and size of resulting bounding box.

Returns

Non-zero if the resulting rectangle is different to the original.

See also

[fl_not_clipped\(\)](#)

30.11.4.11 fl_clip_region() [1/2]

```
void fl_clip_region (
    F1_Region r ) [inline]
```

Replaces the top of the clipping stack with a clipping region of any shape.

`F1_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	<i>r</i>	clipping region
----	----------	-----------------

30.11.4.12 fl_clip_region() [2/2]

```
F1_Region fl_clip_region ( ) [inline]
```

Returns 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.

30.11.4.13 fl_copy_offscreen()

```
void fl_copy_offscreen (
    int x,
    int y,
    int w,
    int h,
    F1_Offscreen pixmap,
    int srcx,
    int srcy ) [inline]
```

Copy a rectangular area of the given offscreen buffer into the current drawing destination.

Parameters

<i>x,y</i>	position where to draw the copied rectangle
<i>w,h</i>	size of the copied rectangle
<i>pixmap</i>	offscreen buffer containing the rectangle to copy
<i>srcx,srcy</i>	origin in offscreen buffer of rectangle to copy

30.11.4.14 fl_create_offscreen()

```
F1_Offscreen fl_create_offscreen (
    int w,
    int h )
```

Creation of an offscreen graphics buffer.

Parameters

<code>w,h</code>	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
<code>Fl_Offscreen off = fl_create_offscreen(w, h)</code>	<code>Fl_Image_Surface *surface = new Fl_Image_Surface(w, h, 1)</code>
<code>fl_begin_offscreen(off)</code>	<code>Fl_Surface_Device::push_current(surface)</code>
<code>fl_end_offscreen()</code>	<code>Fl_Surface_Device::pop_current()</code>
<code>fl_copy_offscreen(x,y,w,h, off, sx,sy)</code>	<code>fl_copy_offscreen(x,y,w,h, surface->offscreen(), sx,sy)</code>
<code>fl_rescale_offscreen(off)</code>	<code>surface->rescale()</code>
<code>fl_delete_offscreen(off)</code>	<code>delete surface</code>

30.11.4.15 fl_cursor()

```
FL_EXPORT 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.

30.11.4.16 fl_curve()

```
void fl_curve (
    double x0,
    double y0,
    double x1,
    double y1,
    double x2,
    double y2,
    double x3,
    double y3 ) [inline]
```

Adds a series of points on a Bezier curve to the path.

The curve ends (and two of the points) are at X0,Y0 and X3,Y3.

Parameters

in	<i>X0,Y0</i>	curve start point
in	<i>X1,Y1</i>	curve control point
in	<i>X2,Y2</i>	curve control point
in	<i>X3,Y3</i>	curve end point

30.11.4.17 fl_delete_offscreen()

```
void fl_delete_offscreen (
    F1_Offscreen ctx )
```

Deletion of an offscreen graphics buffer.

Parameters

<i>ctx</i>	the buffer to be deleted.
------------	---------------------------

Note

The *ctx* argument must have been created by [fl_create_offscreen\(\)](#).

30.11.4.18 fl_draw() [1/4]

```
FL_EXPORT void fl_draw (
    const char * str,
    int x,
    int y )
```

Draws 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.

30.11.4.19 fl_draw() [2/4]

```
FL_EXPORT void fl_draw (
    int angle,
    const char * str,
    int x,
    int y )
```

Draws 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 Xft, Win32 and MacOS fltk subsets.

30.11.4.20 fl_draw() [3/4]

```
void fl_draw (
    int angle,
    const char * str,
    int n,
    int x,
    int y ) [inline]
```

Draws 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.

30.11.4.21 fl_draw() [4/4]

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

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'

30.11.4.22 fl_draw_box()

```
FL_EXPORT 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	<i>t</i>	box type
in	<i>x,y,w,h</i>	position and size
in	<i>c</i>	color

30.11.4.23 [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]
```

Draws an 8-bit per color RGB or luminance image.

Parameters

in	<i>buf</i>	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	<i>X,Y</i>	position where to put top-left corner of image
in	<i>W,H</i>	size of the image
in	<i>D</i>	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	<i>L</i>	delta to add to the pointer between lines (if 0 is passed it uses <i>W</i> * <i>D</i>), and may be larger than <i>W</i> * <i>D</i> 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.

30.11.4.24 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]
```

Draws 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 easily.

Parameters

in	<i>cb</i>	callback function to generate scan line data
in	<i>data</i>	user data passed to callback function
in	<i>X,Y</i>	screen position of top left pixel
in	<i>W,H</i>	image width and height
in	<i>D</i>	data size 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.

30.11.4.25 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]
```

Draws a gray-scale (1 channel) image.

See also

[fl_draw_image\(const uchar* buf, int X,int Y,int W,int H, int D, int L\)](#)

30.11.4.26 `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]
```

Draws a gray-scale image using a callback function to generate image data.

See also

[fl_draw_image\(Fl_Draw_Image_Cb cb, void* data, int X,int Y,int W,int H, int D\)](#)

30.11.4.27 `fl_draw_pixmap()` [1/2]

```
FL_EXPORT int fl_draw_pixmap (
    const 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.

The image is dithered on 8-bit displays so you won't lose color space for programs displaying both images and pixmaps.

Parameters

in	<i>data</i>	pointer to XPM image data
in	<i>x,y</i>	position of top-left corner
in	<i>bg</i>	background color

Returns

0 if there was any error decoding the XPM data.

30.11.4.28 `fl_draw_pixmap()` [2/2]

```
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\(const char* const* data, int x, int y, Fl_Color bg\)](#)

30.11.4.29 fl_draw_symbol()

```
FL_EXPORT 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	<i>label</i>	name of symbol
in	<i>x,y</i>	position of symbol
in	<i>w,h</i>	size of symbol
in	<i>col</i>	color of symbox

Returns

1 on success, 0 on failure

30.11.4.30 fl_expand_text()

```
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 )
```

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](#).

30.11.4.31 fl_frame()

```
FL_EXPORT 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	<i>s</i>	sets of 4 grayscale values in top, left, bottom, right order
in	<i>x,y,w,h</i>	position and size

30.11.4.32 fl_frame2()

```
FL_EXPORT 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	<i>s</i>	sets of 4 grayscale values in bottom, right, top, left order
in	<i>x,y,w,h</i>	position and size

30.11.4.33 fl_gap()

```
void fl_gap ( ) [inline]
```

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 vertex, or several times in a row.

30.11.4.34 fl_line_style()

```
void fl_line_style (
    int style,
    int width = 0,
    char * dashes = 0 ) [inline]
```

Sets 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)`.

Parameters

in	<code>style</code>	A bitmask which is a bitwise-OR of a line style, 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	<code>width</code>	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	<code>dashes</code>	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.

Note

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 Windows 95, 98 or Me, since those operating systems do not support complex line styles.

30.11.4.35 fl_measure()

```
FL_EXPORT 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	<i>str</i>	nul-terminated string
out	<i>w,h</i>	width and height of string in current font
in	<i>draw_symbols</i>	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
```

30.11.4.36 fl_measure_pixmap() [1/2]

```
FL_EXPORT int fl_measure_pixmap (
    const 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	<i>data</i>	pointer to XPM image data.
out	<i>w,h</i>	width and height of image

Returns

non-zero if the dimensions were parsed OK
0 if there were any problems

30.11.4.37 fl_measure_pixmap() [2/2]

```
FL_EXPORT int fl_measure_pixmap (
    const char *const * cdata,
    int & w,
    int & h )
```

Get the dimensions of a pixmap.

See also

[fl_measure_pixmap\(char* const* data, int &w, int &h\)](#)

30.11.4.38 fl_mult_matrix()

```
void fl_mult_matrix (
    double a,
    double b,
    double c,
    double d,
    double x,
    double y ) [inline]
```

Concatenates another transformation onto the current one.

Parameters

in	<i>a,b,c,d,x,y</i>	transformation matrix elements such that $X' = aX + cY + x$ and $Y' = bX + dY + y$
----	--------------------	--

30.11.4.39 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	<i>x,y,w,h</i>	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\(\)](#)

30.11.4.40 fl_old_shortcut()

```
FL_EXPORT 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.

30.11.4.41 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 $a1$.

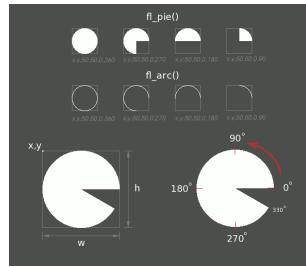


Figure 30.4 [fl_pie\(\)](#) and [fl_arc\(\)](#)

30.11.4.42 fl_polygon() [1/2]

```
void fl_polygon (
    int x,
    int y,
    int x1,
    int y1,
    int x2,
    int y2 ) [inline]
```

Fills a 3-sided polygon.

The polygon must be convex.

30.11.4.43 fl_polygon() [2/2]

```
void fl_polygon (
    int x,
    int y,
    int x1,
    int y1,
    int x2,
    int y2,
    int x3,
    int y3 )  [inline]
```

Fills a 4-sided polygon.

The polygon must be convex.

30.11.4.44 fl_pop_clip()

```
void fl_pop_clip ( )  [inline]
```

Restores 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.

30.11.4.45 fl_push_clip()

```
void fl_push_clip (
    int x,
    int y,
    int w,
    int h )  [inline]
```

Intersects the current clip region with a rectangle and pushes this new region onto the stack.

Parameters

in	x,y,w,h	position and size
----	---------	-------------------

30.11.4.46 fl_push_matrix()

```
void fl_push_matrix ( )  [inline]
```

Saves the current transformation matrix on the stack.

The maximum depth of the stack is 32.

30.11.4.47 fl_read_image()

```
FL_EXPORT 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	<i>p</i>	pixel buffer, or NULL to allocate one
in	<i>X,Y</i>	position of top-left of image to read
in	<i>w,h</i>	width and height of image to read
in	<i>alpha</i>	alpha value for image (0 for none)

Returns

pointer to pixel buffer, or NULL 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_part\(\)](#)

30.11.4.48 fl_rect()

```
void fl_rect (
    int x,
    int y,
    int w,
    int h ) [inline]
```

Draws a 1-pixel 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.

30.11.4.49 fl_rectf()

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

Colors 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.

30.11.4.50 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

30.11.4.51 fl_reset_spot()

```
FL_EXPORT void fl_reset_spot (
    void )
```

Todo provide user documentation for fl_reset_spot function

30.11.4.52 fl_rotate()

```
void fl_rotate (
    double d ) [inline]
```

Concatenates rotation transformation onto the current one.

Parameters

in	<i>d</i>	- rotation angle, counter-clockwise in degrees (not radians)
----	----------	--

30.11.4.53 fl_scale() [1/2]

```
void fl_scale (
    double x,
    double y ) [inline]
```

Concatenates scaling transformation onto the current one.

Parameters

in	<i>x,y</i>	scale factors in x-direction and y-direction
----	------------	--

30.11.4.54 fl_scale() [2/2]

```
void fl_scale (
    double x ) [inline]
```

Concatenates scaling transformation onto the current one.

Parameters

in	<i>x</i>	scale factor in both x-direction and y-direction
----	----------	--

30.11.4.55 fl_scroll()

```
FL_EXPORT 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	<i>X,Y</i>	position of top-left of rectangle
in	<i>W,H</i>	size of rectangle
in	<i>dx,dy</i>	pixel offsets for shifting rectangle
in	<i>draw_area</i>	callback function to draw rectangular areas
in	<i>data</i>	pointer to user data for callback The contents of the rectangular area is first shifted by <i>dx</i> and <i>dy</i> pixels. The <i>draw_area</i> callback is then called for every newly exposed rectangular area.

30.11.4.56 fl_set_spot()

```
FL_EXPORT void fl_set_spot (
    int font,
    int size,
    int X,
    int Y,
    int W,
    int H,
    Fl_Window * win = 0 )
```

Todo provide user documentation for fl_set_spot function

30.11.4.57 fl_set_status()

```
FL_EXPORT void fl_set_status (
    int X,
    int Y,
    int W,
    int H )
```

Todo provide user documentation for fl_set_status function

30.11.4.58 `fl_shortcut_label()` [1/2]

```
FL_EXPORT 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

<code>in</code>	<code>shortcut</code>	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

30.11.4.59 fl_shortcut_label() [2/2]

```
FL_EXPORT const char* fl_shortcut_label (
    unsigned int shortcut,
    const char ** eom )
```

Get a human-readable string from a shortcut value.

Parameters

in	<i>shortcut</i>	the integer value containing the ASCII character or extended keystroke plus modifiers
in	<i>eom</i>	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\(unsigned int shortcut\)](#)

30.11.4.60 fl_transform_dx()

```
double fl_transform_dx (
    double x,
    double y ) [inline]
```

Transforms distance using current transformation matrix.

Parameters

in	<i>x,y</i>	coordinate
----	------------	------------

30.11.4.61 fl_transform_dy()

```
double fl_transform_dy (
    double x,
    double y ) [inline]
```

Transforms distance using current transformation matrix.

Parameters

in	x,y	coordinate
----	-----	------------

30.11.4.62 fl_transform_x()

```
double fl_transform_x (
    double x,
    double y ) [inline]
```

Transforms coordinate using the current transformation matrix.

Parameters

in	x,y	coordinate
----	-----	------------

30.11.4.63 fl_transform_y()

```
double fl_transform_y (
    double x,
    double y ) [inline]
```

Transforms coordinate using the current transformation matrix.

Parameters

in	x,y	coordinate
----	-----	------------

30.11.4.64 fl_transformed_vertex()

```
void fl_transformed_vertex (
    double xf,
    double yf ) [inline]
```

Adds coordinate pair to the vertex list without further transformations.

Parameters

in	xf,yf	transformed coordinate
----	-------	------------------------

30.11.4.65 fl_translate()

```
void fl_translate (
    double x,
    double y ) [inline]
```

Concatenates translation transformation onto the current one.

Parameters

in	x,y	translation factor in x-direction and y-direction
----	-----	---

30.11.4.66 fl_vertex()

```
void fl_vertex (
    double x,
    double y ) [inline]
```

Adds a single vertex to the current path.

Parameters

in	x,y	coordinate
----	-----	------------

30.12 Multithreading support functions

fl multithreading support functions declared in <FL/Fl.H>

Functions

- static void [Fl::awake](#) (void *message=0)
Sends a message pointer to the main thread, causing any pending [Fl::wait\(\)](#) call to terminate so that the main thread can retrieve the message and any pending redraws can be processed.
- static int [Fl::awake](#) ([Fl_Awake_Handler](#) cb, void *message=0)
See [void awake\(void message=0\)](#).*
- static int [Fl::lock](#) ()
The [lock\(\)](#) method blocks the current thread until it can safely access FLTK widgets and data.
- static void * [Fl::thread_message](#) ()
The [thread_message\(\)](#) method returns the last message that was sent from a child by the [awake\(\)](#) method.
- static void [Fl::unlock](#) ()
The [unlock\(\)](#) method releases the lock that was set using the [lock\(\)](#) method.

30.12.1 Detailed Description

fl multithreading support functions declared in <FL/Fl.H>

30.12.2 Function Documentation

30.12.2.1 [awake\(\)](#) [1/2]

```
void Fl::awake (
    void * msg = 0 )  [static]
```

Sends a message pointer to the main thread, causing any pending [Fl::wait\(\)](#) call to terminate so that the main thread can retrieve the message and any pending redraws can be processed.

Multiple calls to [Fl::awake\(\)](#) will queue multiple pointers for the main thread to process, up to a system-defined (typically several thousand) depth. The default message handler saves the last message which can be accessed using the [Fl::thread_message\(\)](#) function.

In the context of a threaded application, a call to [Fl::awake\(\)](#) with no argument will trigger event loop handling in the main thread. Since it is not possible to call [Fl::flush\(\)](#) from a subsidiary thread, [Fl::awake\(\)](#) is the best (and only, really) substitute.

See also: [Multithreading](#)

30.12.2.2 `awake()` [2/2]

```
int Fl::awake (
    Fl_Awake_Handler func,
    void * data = 0 ) [static]
```

See `void awake(void* message=0)`.

Let the main thread know an update is pending and have it call a specific function.

Registers a function that will be called by the main thread during the next message handling cycle. Returns 0 if the callback function was registered, and -1 if registration failed. Over a thousand awake callbacks can be registered simultaneously.

See also

[Fl::awake\(void* message=0\)](#)

30.12.2.3 `lock()`

```
int Fl::lock ( ) [static]
```

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 `lock()` to initialize the threading support in FLTK. `lock()` will return non-zero if threading is not available on the platform.

Child threads must call `unlock()` when they are done accessing FLTK.

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](#)

30.12.2.4 `thread_message()`

```
void * Fl::thread_message ( ) [static]
```

The `thread_message()` method returns the last message that was sent from a child by the `awake()` method.

See also: [Multithreading](#)

30.12.2.5 `unlock()`

```
void Fl::unlock ( ) [static]
```

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](#)

30.13 Safe widget deletion support functions

These functions, declared in <[FL/Fl.H](#)>, support deletion of widgets inside callbacks.

Functions

- static void [Fl::clear_widget_pointer\(Fl_Widget const *w\)](#)
Clears a widget pointer in the watch list.
- static void [Fl::delete_widget\(Fl_Widget *w\)](#)
Schedules a widget for deletion at the next call to the event loop.
- static void [Fl::do_widget_deletion\(\)](#)
Deletes widgets previously scheduled for deletion.
- static void [Fl::release_widget_pointer\(Fl_Widget *&w\)](#)
Releases a widget pointer from the watch list.
- static void [Fl::watch_widget_pointer\(Fl_Widget *&w\)](#)
Adds a widget pointer to the widget watch list.

30.13.1 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
2. 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
3. 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](#).

30.13.2 Function Documentation

30.13.2.1 clear_widget_pointer()

```
void Fl::clear_widget_pointer (
    Fl_Widget const * w ) [static]
```

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](#)

30.13.2.2 delete_widget()

```
void Fl::delete_widget (
    Fl_Widget * wi ) [static]
```

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_ask\(\)](#), 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\(\)](#)

30.13.2.3 do_widget_deletion()

```
void Fl::do_widget_deletion ( ) [static]
```

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\(Fl_Widget *wi\)](#)

30.13.2.4 release_widget_pointer()

```
void Fl::release_widget_pointer (
    Fl_Widget *& w ) [static]
```

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\(\)](#)

30.13.2.5 watch_widget_pointer()

```
void Fl::watch_widget_pointer (
    Fl_Widget *& w ) [static]
```

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](#)

30.14 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 a FLTK window with cairo support.

Functions

- static 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.
- static bool [Fl::cairo_autolink_context](#) ()
Gets the current autolink mode for cairo support.
- static cairo_t * [Fl::cairo_cc](#) ()
Gets the current cairo context linked with a fltk window.
- static void [Fl::cairo_cc](#) (cairo_t *c, bool own=false)
Sets the current cairo context to c.
- static cairo_t * [Fl::cairo_make_current](#) ([Fl_Window](#) *w)
Provides a corresponding cairo context for window wi.

30.14.1 Detailed Description

30.14.2 Function Documentation

30.14.2.1 [cairo_autolink_context\(\)](#) [1/2]

```
static void Fl::cairo_autolink_context (
    bool alink) [inline], [static]
```

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 but 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 when configure has the –enable-cairo option

30.14.2.2 [cairo_autolink_context\(\)](#) [2/2]

```
static bool Fl::cairo_autolink_context () [inline], [static]
```

Gets the current autolink mode for cairo support.

Return values

<i>false</i>	if no cairo context autolink is made for each window.
<i>true</i>	if any fltk window is attached a cairo context when it is current.

See also

void [cairo_autolink_context\(bool alink\)](#)

Note

Only available when configure has the –enable-cairo option

30.14.2.3 [cairo_cc\(\)](#) [1/2]

```
static cairo_t* Fl::cairo_cc ( ) [inline], [static]
```

Gets the current cairo context linked with a fltk window.

30.14.2.4 [cairo_cc\(\)](#) [2/2]

```
static void Fl::cairo_cc (
    cairo_t * c,
    bool own = false ) [inline], [static]
```

Sets the current cairo context to *c*.

Set *own* to true if you want fltk to handle this cc deletion.

Note

Only available when configure has the –enable-cairo option

30.14.2.5 [cairo_make_current\(\)](#)

```
cairo_t * Fl::cairo_make_current (
    Fl_Window * wi ) [static]
```

Provides a corresponding 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 configure has the –enable-cairo option

Returns

the valid cairo_t* cairo context associated to this window.

30.15 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

- `FL_EXPORT int fl_access (const char *f, int mode)`
Cross-platform function to test a files access() with a UTF-8 encoded name or value.
- `FL_EXPORT int fl_chdir (const char *path)`
Cross-platform function to change the current working directory, given as a UTF-8 encoded string.
- `FL_EXPORT int fl_chmod (const char *f, int mode)`
Cross-platform function to set a files mode() with a UTF-8 encoded name or value.
- `FL_EXPORT int fl_execvp (const char *file, char *const *argv)`
- `FL_EXPORT FILE * fl_fopen (const char *f, const char *mode)`
Cross-platform function to open files with a UTF-8 encoded name.
- `FL_EXPORT char * fl_getcwd (char *buf, int len)`
Cross-platform function to get the current working directory as a UTF-8 encoded value.
- `FL_EXPORT char * fl_getenv (const char *v)`
Cross-platform function to get environment variables with a UTF-8 encoded name or value.
- `FL_EXPORT char fl_make_path (const char *path)`
Cross-platform function to recursively create a path in the file system.
- `FL_EXPORT void fl_make_path_for_file (const char *path)`
Cross-platform function to create a path for the file in the file system.
- `FL_EXPORT int fl_mkdir (const char *f, int mode)`
Cross-platform function to create a directory with a UTF-8 encoded name.
- `FL_EXPORT unsigned int fl_nonspacing (unsigned int ucs)`
Returns true if the Unicode character ucs is non-spacing.
- `FL_EXPORT int fl_open (const char *fname, int oflags,...)`
Cross-platform function to open files with a UTF-8 encoded name.
- `FL_EXPORT int fl_open_ext (const char *fname, int binary, int oflags,...)`
Cross-platform function to open files with a UTF-8 encoded name.
- `FL_EXPORT int fl_putenv (const char *var)`
Cross-platform function to write environment variables with a UTF-8 encoded name or value.
- `FL_EXPORT int fl_rename (const char *f, const char *n)`
Cross-platform function to rename a filesystem object using UTF-8 encoded names.
- `FL_EXPORT int fl_rmdir (const char *f)`
Cross-platform function to remove a directory with a UTF-8 encoded name.
- `FL_EXPORT int fl_stat (const char *f, struct stat *b)`
Cross-platform function to stat() a file using a UTF-8 encoded name or value.

- FL_EXPORT int `fl_system` (const char *cmd)
Cross-platform function to run a system command with a UTF-8 encoded string.
- FL_EXPORT int `fl_tolower` (unsigned int ucs)
Returns the Unicode lower case value of ucs.
- FL_EXPORT int `fl_toupper` (unsigned int ucs)
Returns the Unicode upper case value of ucs.
- FL_EXPORT 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.
- FL_EXPORT int `fl_unlink` (const char *fname)
Cross-platform function to unlink() (that is, delete) a file using a UTF-8 encoded filename.
- FL_EXPORT char * `fl_utf2mbcs` (const char *s)
Converts UTF-8 string s to a local multi-byte character string.
- FL_EXPORT 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.
- FL_EXPORT int `fl_utf8bytes` (unsigned ucs)
Return the number of bytes needed to encode the given UCS4 character in UTF-8.
- FL_EXPORT unsigned `fl_utf8decode` (const char *p, const char *end, int *len)
Decode a single UTF-8 encoded character starting at p.
- FL_EXPORT int `fl_utf8encode` (unsigned ucs, char *buf)
Write the UTF-8 encoding of ucs into buf and return the number of bytes written.
- FL_EXPORT 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.
- FL_EXPORT 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.
- FL_EXPORT unsigned `fl_utf8fromw` (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.
- FL_EXPORT 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.
- FL_EXPORT int `fl_utf8len` (char c)
Returns the byte length of the UTF-8 sequence with first byte c, or -1 if c is not valid.
- FL_EXPORT 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.
- FL_EXPORT int `fl_utf8locale` ()
Return true if the "locale" seems to indicate that UTF-8 encoding is used.
- FL_EXPORT 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.
- FL_EXPORT 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).
- FL_EXPORT unsigned `fl_utf8toa` (const char *src, unsigned srclen, char *dst, unsigned dstlen)
Convert a UTF-8 sequence into an array of 1-byte characters.
- FL_EXPORT 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.
- FL_EXPORT unsigned `fl_utf8towc` (const char *src, unsigned srclen, wchar_t *dst, unsigned dstlen)
Converts a UTF-8 string into a wide character string.
- FL_EXPORT int `fl_utf_nb_char` (const unsigned char *buf, int len)
Returns the number of Unicode chars in the UTF-8 string.
- FL_EXPORT int `fl_utf_strcasecmp` (const char *s1, const char *s2)
UTF-8 aware strcasecmp - converts to Unicode and tests.
- FL_EXPORT int `fl_utf_strncasecmp` (const char *s1, const char *s2, int n)
UTF-8 aware strncasecmp - converts to lower case Unicode and tests.

- FL_EXPORT int [fl_utf_tolower](#) (const unsigned char *str, int len, char *buf)
Converts the string str to its lower case equivalent into buf.
- FL_EXPORT int [fl_utf_toupper](#) (const unsigned char *str, int len, char *buf)
Converts the string str to its upper case equivalent into buf.
- FL_EXPORT int [fl_wcwidth](#) (const char *src)
extended wrapper around fl_wcwidth_(unsigned int ucs) function.
- FL_EXPORT int [fl_wcwidth_](#) (unsigned int ucs)
Wrapper to adapt Markus Kuhn's implementation of wcwidth() for FLTK.

30.15.1 Detailed Description

fl global Unicode and UTF-8 handling functions declared in <[FL/fl_utf8.h](#)>

30.15.2 Macro Definition Documentation

30.15.2.1 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.

30.15.2.2 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 0xffffd. 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.

30.15.2.3 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.

30.15.3 Function Documentation

30.15.3.1 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.

Parameters

in	<i>f</i>	the UTF-8 encoded filename
in	<i>mode</i>	the mode to test

Returns

the return value of _waccess() on Windows or access() on other platforms.

30.15.3.2 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

<code>in</code>	<code>path</code>	the target directory for chdir (may be NULL)
-----------------	-------------------	--

Returns

0 if successful, -1 on error (errno may be set)

30.15.3.3 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

<code>in</code>	<code>f</code>	the UTF-8 encoded filename
<code>in</code>	<code>mode</code>	the mode to set

Returns

the return value of _wchmod() on Windows or chmod() on other platforms.

30.15.3.4 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.

Parameters

<code>f</code>	the UTF-8 encoded filename
<code>mode</code>	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\(\)](#).

30.15.3.5 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	<i>buf</i>	the buffer to populate (may be NULL)
in	<i>len</i>	the length of the buffer

Returns

the CWD encoded as UTF-8

30.15.3.6 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.

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.

30.15.3.7 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

30.15.3.8 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.

30.15.3.9 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	<i>f</i>	the UTF-8 encoded filename
in	<i>mode</i>	the mode of the directory

Returns

the return value of `_wmkdir()` on Windows or `mkdir()` on other platforms.

30.15.3.10 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

30.15.3.11 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.

Parameters

in	<i>fname</i>	the UTF-8 encoded filename
in	<i>oflags</i>	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\(\)](#).

30.15.3.12 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	<i>fname</i>	the UTF-8 encoded filename
in	<i>binary</i>	if non-zero, the file is to be accessed in binary (a.k.a. untranslated) mode.
in	<i>oflags</i> ,...	these arguments are as in the standard open() function. Setting <i>oflags</i> to zero opens the file for reading.

Returns

a file descriptor upon successful completion, or -1 in case of error.

30.15.3.13 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	<code>var</code>	the UTF-8 encoded environment variable 'name=value'
----	------------------	---

Returns

0 on success, non-zero in case of error.

30.15.3.14 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	<code>f</code>	the UTF-8 encoded filename to change
in	<code>n</code>	the new UTF-8 encoded filename to set

Returns

the return value of `_wrename()` on Windows or `rename()` on other platforms.

30.15.3.15 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.

30.15.3.16 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
	b	the stat struct to populate

Returns

the return value of `_wstat()` on Windows or `stat()` on other platforms.

30.15.3.17 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.

Parameters

<code>in</code>	<code>cmd</code>	the UTF-8 encoded command string
-----------------	------------------	----------------------------------

Returns

the return value of `_wsystem()` on Windows or `system()` on other platforms.

30.15.3.18 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.

`ucs` is the value to convert.

`dst` points at an array to write, and `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.

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.

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.

30.15.3.19 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

<code>fname</code>	the filename to unlink
--------------------	------------------------

Returns

the return value of `_wunlink()` on Windows or `unlink()` on other platforms.

30.15.3.20 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.

`start` is the start of the string and is used to limit the backwards search for the start of a UTF-8 character.

`end` is the end of the string and is assumed to be a break between characters. It is assumed to be greater than `p`.

If you wish to decrement a UTF-8 pointer, pass `p-1` to this.

30.15.3.21 fl_utf8bytes()

```
int fl_utf8bytes (
    unsigned ucs )
```

Return the number of bytes needed to encode the given UCS4 character in UTF-8.

Returns number of bytes that `utf8encode()` will use to encode the character `ucs`.

Parameters

in	ucs	UCS4 encoded character
----	-----	------------------------

Returns

number of bytes required

30.15.3.22 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 {
    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.

30.15.3.23 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 0xFFFFD (REPLACEMENT CHARACTER).

RFC 3629 also says many other values for *ucs* are illegal (in the range 0xd800 to 0xdfff, or ending with 0xffffe or 0xfffff). 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.

30.15.3.24 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.

30.15.3.25 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.

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 bytes in `src` to convert.

If the return value equals `srclen` then this indicates that no conversion is necessary, as only ASCII characters are in the string.

30.15.3.26 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 0xFFFFD (REPLACEMENT CHARACTER). Characters in the range 0xd800 to 0xdfff, or ending with 0xffffe or 0xfffff 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.

30.15.3.27 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.

start is the start of the string and is used to limit the backwards search for the start of a UTF-8 character.

end is the end of the string and is assumed to be a break between characters. It is assumed to be greater than *p*.

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.

30.15.3.28 fl_utf8len()

```
int fl_utf8len (
    char c )
```

Returns the byte length of the UTF-8 sequence with first byte *c*, or -1 if *c* is not valid.

This function is helpful for finding faulty UTF-8 sequences.

See also

[fl_utf8len1](#)

30.15.3.29 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 sequences, albeit ignoring invalid codes.

See also

[fl_utf8len](#)

30.15.3.30 fl_utf8locale()

```
int fl_utf8locale ( )
```

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.

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.

30.15.3.31 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.

30.15.3.32 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.

30.15.3.33 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).

`src` points at the UTF-8 sequence, and `srclen` is the number of bytes to convert.

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.

30.15.3.34 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.

`src` points at the UTF-8, and `srclen` is the number of bytes to convert.

`dst` points at an array to write, and `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.

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.

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).

30.15.3.35 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.

`src` points at the UTF-8, and `srclen` is the number of bytes to convert.

`dst` points at an array to write, and `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`.

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.

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 that Windows includes Cygwin, i.e. compiled with Cygwin's POSIX layer (`cygwin1.dll`, `--enable-cygwin`), either native (GDI) or X11.

30.15.3.36 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

30.15.3.37 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.

Parameters

<i>s1,s2</i>	the UTF-8 strings to compare
<i>n</i>	the maximum number of UTF-8 characters to compare

Returns

result of comparison

Return values

<i>0</i>	if the strings are equal
<i>>0</i>	if <i>s1</i> is greater than <i>s2</i>
<i><0</i>	if <i>s1</i> is less than <i>s2</i>

30.15.3.38 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*.

Warning: to be safe *buf* length must be at least $3 * \text{len}$ [for 16-bit Unicode]

30.15.3.39 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*.

Warning: to be safe *buf* length must be at least $3 * \text{len}$ [for 16-bit Unicode]

30.15.3.40 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.

30.15.3.41 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.

30.16 Mac OS X-specific symbols

Mac OS X-specific symbols declared in <[FL/platform.H](#)>

Classes

- class [Fl_Mac_App_Menu](#)

Functions

- 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.
- 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).

30.16.1 Detailed Description

Mac OS X-specific symbols declared in <[FL/platform.H](#)>

See also

[The Apple OS X Interface](#)

30.16.2 Function Documentation

30.16.2.1 [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\(Fl_Callback *cb, void *user_data\)](#).

30.16.2.2 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.

30.16.3 Variable Documentation

30.16.3.1 fl_mac_os_version

```
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).

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.

30.17 Common Dialogs classes and functions

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.
- int [fl_ask](#) (const char *fmt,...)
Shows a dialog displaying the `fmt` message, this dialog features 2 yes/no buttons.
- void [fl_beep](#) (int type)
Emits a system beep message.
- 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_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 *)
Set the "OK" button label.
- const char * [fl_input](#) (const char *fmt, const char *defstr,...)
Shows an input dialog displaying the `fmt` message.
- void [fl_message](#) (const char *fmt,...)
Shows an information message dialog box.
- void [fl_message_hotspot](#) (int enable)
Sets whether or not to move the common 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 common 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_position](#) (const int x, const int y, const int center)
Sets the preferred position for the common 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 common 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 common 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.*

Variables

- static void(* `Fl::error`)(const char *,...) = Fl_System_Driver::error
 - FLTK calls `Fl::error()` to output a normal error message.*
- static 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*
- 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*
- static void(* `Fl::warning`)(const char *,...) = Fl_System_Driver::warning
 - FLTK calls `Fl::warning()` to output a warning message.*

30.17.1 Detailed Description

30.17.2 Function Documentation

30.17.2.1 `fl_alert()`

```
void fl_alert (
    const char * fmt,
    ... )
```

Shows an alert message dialog box.

Note

Common dialog boxes are application modal. No more than one common dialog box can be open at any time. Requests for additional dialog boxes are ignored.

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

Parameters

in	<i>fmt</i>	can be used as an sprintf-like format and variables for the message text
----	------------	--

30.17.2.2 fl_ask()

```
int fl_ask (
    const char * fmt,
    ... )
```

Shows a dialog displaying the *fmt* message, this dialog features 2 yes/no buttons.

Note

Common dialog boxes are application modal. No more than one common dialog box can be open at any time.
Requests for additional dialog boxes are ignored.

#include <FL/fl_ask.H>

Parameters

in	<i>fmt</i>	can be used as an sprintf-like format and variables for the message text
----	------------	--

Return values

0	if the no button is selected or another dialog box is still open
1	if yes is selected

Deprecated [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\(\)](#) with the appropriate verbs instead.

30.17.2.3 fl_beep()

```
void fl_beep (
    int type )
```

Emits a system beep message.

Parameters

in	<i>type</i>	The beep type from the Fl_Beep enumeration.
----	-------------	---

Note

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

30.17.2.4 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.

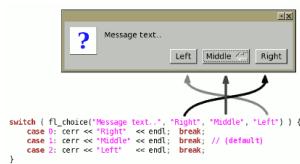


Figure 30.5 fl_choice() button ordering

Note

Common dialog boxes are application modal. No more than one common dialog box can be open at any time. Requests for additional dialog boxes are ignored.

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

Three choices with printf() style formatting:

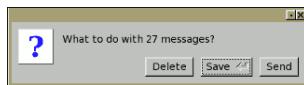


Figure 30.6 fl_choice() three choices with printf formatting

```
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:

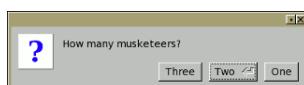


Figure 30.7 fl_choice() three choices

```
switch ( fl_choice("How many bedrooms?", "Zero", "One", "Two") ) {
    case 0: ... // "Zero"
    case 1: ... // "One" (default)
    case 2: ... // "Two"
}
```

Two choice example:

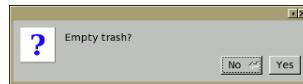


Figure 30.8 fl_choice() two choices

```
switch ( fl_choice("Empty trash?", "Yes", "No", 0) ) {
    case 0: ... // Yes
    case 1: ... // No (default)
}
```

One choice example:

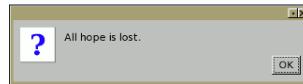


Figure 30.9 fl_choice() one choice

```
fl_choice("All hope is lost.", "OK", 0, 0); // "OK" default
```

Parameters

in	<i>fmt</i>	can be used as an sprintf-like format and variables for the message text
in	<i>b0</i>	text label for right button 0
in	<i>b1</i>	text label for middle button 1 (can be 0)
in	<i>b2</i>	text label for left button 2 (can be 0)

Return values

0	if the button with <i>b0</i> text is pushed or another dialog box is still open
1	if the button with <i>b1</i> text is pushed
2	if the button with <i>b2</i> text is pushed

30.17.2.5 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>
```

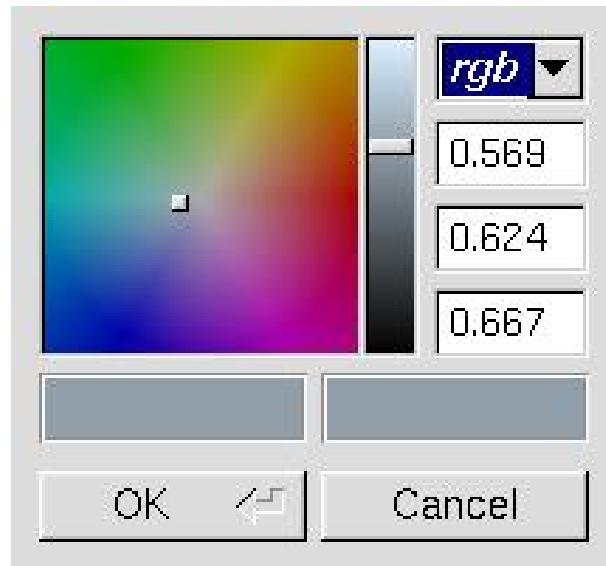


Figure 30.10 fl_color_chooser

Parameters

in	<i>name</i>	Title label for the window
in, out	<i>r,g,b</i>	Color components in the range 0.0 to 1.0.
in	<i>cmode</i>	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

30.17.2.6 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>
```

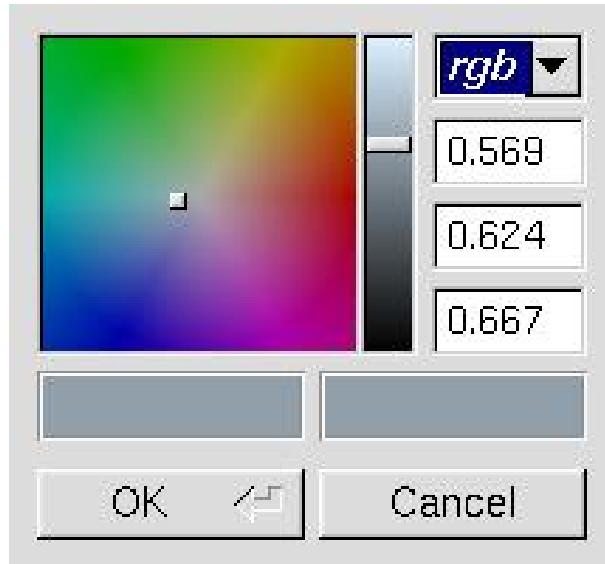


Figure 30.11 fl_color_chooser

Parameters

in	<i>name</i>	Title label for the window
in,out	<i>r,g,b</i>	Color components in the range 0 to 255.
in	<i>cmode</i>	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

30.17.2.7 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	<i>message</i>	title bar text
in	<i>fname</i>	initial/default directory name
in	<i>relative</i>	0 for absolute path return, relative otherwise

Returns

the directory path string chosen by the user or NULL if user cancels

30.17.2.8 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>
```

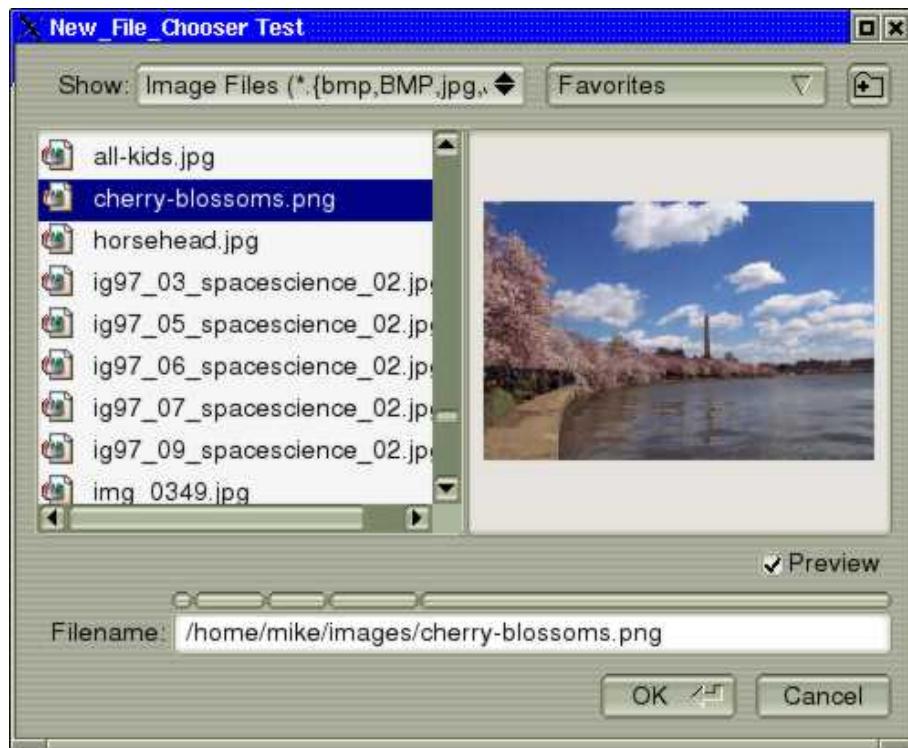


Figure 30.12 FL_File_Chooser

Parameters

in	<i>message</i>	text in title bar
in	<i>pat</i>	filename pattern filter
in	<i>fname</i>	initial/default filename selection
in	<i>relative</i>	0 for absolute path name, relative path name otherwise

Returns

the user selected filename, in absolute or relative format or NULL if user cancels

30.17.2.9 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>
```

30.17.2.10 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>
```

30.17.2.11 fl_input()

```
const char* fl_input (
    const char * fmt,
    const char * defstr,
    ... )
```

Shows an input dialog displaying the *fmt* message.

Note

Common dialog boxes are application modal. No more than one common dialog box can be open at any time.
Requests for additional dialog boxes are ignored.

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

Parameters

in	<i>fmt</i>	can be used as an sprintf-like format and variables for the message text
in	<i>defstr</i>	defines the default returned string if no text is entered

Returns

the user string input if OK was pushed, NULL if Cancel was pushed or another dialog box was still open

30.17.2.12 fl_message()

```
void fl_message (
    const char * fmt,
    ...
)
```

Shows an information message dialog box.

Note

Common dialog boxes are application modal. No more than one common dialog box can be open at any time.
Requests for additional dialog boxes are ignored.

#include <FL/fl_ask.H>

Parameters

in	<i>fmt</i>	can be used as an sprintf-like format and variables for the message text
----	------------	--

30.17.2.13 fl_message_hotspot() [1/2]

```
void fl_message_hotspot (
    int enable )
```

Sets whether or not to move the common 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.

Note

#include <FL/fl_ask.H>

Parameters

in	<i>enable</i>	non-zero enables hotspot behavior, 0 disables hotspot
----	---------------	---

30.17.2.14 fl_message_hotspot() [2/2]

```
int fl_message_hotspot (
    void )
```

Gets whether or not to move the common 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.

Note

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

Returns

0 if disable, non-zero otherwise

See also

[fl_message_hotspot\(int\)](#)

30.17.2.15 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\(\)](#)

Note

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

30.17.2.16 fl_message_position() [1/3]

```
void fl_message_position (
    const int x,
    const int y,
    const int center )
```

Sets the preferred position for the common message box used in many common dialogs like [fl_message\(\)](#), [fl_alert\(\)](#), [fl_ask\(\)](#), [fl_choice\(\)](#), [fl_input\(\)](#), [fl_password\(\)](#).

Resets after every call to any of the common dialogs.

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\(int\)](#).

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).

Note

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

Parameters

in	<i>x</i>	Preferred X position
in	<i>y</i>	Preferred Y position
in	<i>center</i>	1 = centered, 0 = absolute

See also

int [fl_message_position\(int *x, int *y\)](#)

30.17.2.17 fl_message_position() [2/3]

```
void fl_message_position (
    Fl_Widget * widget )
```

Sets the preferred position for the common message box used in many common dialogs like [fl_message\(\)](#), [fl_alert\(\)](#), [fl_ask\(\)](#), [fl_choice\(\)](#), [fl_input\(\)](#), [fl_password\(\)](#).

The common 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.

Note

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

Parameters

in	<i>widget</i>	Widget or window to position the message box over.
----	---------------	--

See also

int [fl_message_position\(int x, int y, int center\)](#)

30.17.2.18 fl_message_position() [3/3]

```
int fl_message_position (
    int * x,
    int * y )
```

Gets the preferred position for the common message box used in many common dialogs like [fl_message\(\)](#), [fl_alert\(\)](#), [fl_ask\(\)](#), [fl_choice\(\)](#), [fl_input\(\)](#), [fl_password\(\)](#).

Note

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

Parameters

<code>out</code>	<code>x</code>	Preferred X position, returns -1 if not set
<code>out</code>	<code>y</code>	Preferred Y position, returns -1 if not set

Returns

whether position is currently set or not

Return values

<code>0</code>	position is not set (may be hotspot or not)
<code>1</code>	position is set (window position)
<code>2</code>	position is set (message box centered)

See also

[fl_message_position\(int, int\)](#)
[fl_message_hotspot\(int\)](#)
int [fl_message_hotspot\(\)](#)

30.17.2.19 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.

Note

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

Parameters

<code>in</code>	<code>title</code>	window label, string copied internally
-----------------	--------------------	--

30.17.2.20 [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.

Note

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

Parameters

in	<i>title</i>	default window label, string copied internally
----	--------------	--

30.17.2.21 [fl_password\(\)](#)

```
const char* fl_password (
    const char * fmt,
    const char * defstr,
    ... )
```

Shows an input dialog displaying the *fmt* message.

Like [fl_input\(\)](#) except the input text is not shown, '*' characters are displayed instead.

Note

Common dialog boxes are application modal. No more than one common dialog box can be open at any time.
Requests for additional dialog boxes are ignored.

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

Parameters

in	<i>fmt</i>	can be used as an sprintf-like format and variables for the message text
in	<i>defstr</i>	defines the default returned string if no text is entered

Returns

the user string input if OK was pushed, NULL if Cancel was pushed or another dialog box was still open

30.17.3 Variable Documentation

30.17.3.1 error

```
void(* Fl::error)(const char *format,...) = Fl_System_Driver::error [static]
```

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>
```

30.17.3.2 fatal

```
void(* Fl::fatal)(const char *format,...) = Fl_System_Driver::fatal [static]
```

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>
```

30.17.3.3 warning

```
void(* Fl::warning)(const char *format,...) = Fl_System_Driver::warning [static]
```

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>
```

30.18 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() `FI_File_Sort_F`(struct dirent **, struct dirent **)
File sorting function.

Functions

- FL_EXPORT void `fl_decode_uri` (char *uri)
Decodes a URL-encoded string.
- FL_EXPORT int `fl_filename_absolute` (char *to, int tolen, const char *from)
Makes a filename absolute from a relative filename.
- FL_EXPORT int `fl_filename_expand` (char *to, int tolen, const char *from)
Expands a filename containing shell variables and tilde (~).
- FL_EXPORT const char * `fl_filename_ext` (const char *buf)
Gets the extension of a filename.
- FL_EXPORT void `fl_filename_free_list` (struct dirent ***l, int n)
Free the list of filenames that is generated by `fl_filename_list()`.
- FL_EXPORT int `fl_filename_isdir` (const char *name)
Determines if a file exists and is a directory from its filename.
- FL_EXPORT int `fl_filename_list` (const char *d, struct dirent ***l, `FI_File_Sort_F` *s=`fl_numericsort`)
Portable and const-correct wrapper for the scandir() function.
- FL_EXPORT int `fl_filename_match` (const char *name, const char *pattern)
Checks if a string s matches a pattern p.
- FL_EXPORT const char * `fl_filename_name` (const char *filename)
Gets the file name from a path.
- FL_EXPORT int `fl_filename_relative` (char *to, int tolen, const char *from)
Makes a filename relative to the current working directory.
- FL_EXPORT char * `fl_filename_setext` (char *to, int tolen, const char *ext)
Replaces the extension in buf of max.
- FL_EXPORT int `fl_open_uri` (const char *uri, char *msg, int msglen)
Opens the specified Uniform Resource Identifier (URI).

30.18.1 Detailed Description

File names and URI functions defined in <FL/filename.H>

30.18.2 Typedef Documentation

30.18.2.1 Fl_File_Sort_F

```
typedef int() Fl_File_Sort_F(struct dirent **, struct dirent **)
```

File sorting function.

See also

[fl_filename_list\(\)](#)

30.18.3 Function Documentation

30.18.3.1 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.

30.18.3.2 fl_filename_absolute()

```
FL_EXPORT int fl_filename_absolute (
    char * to,
    int tolen,
    const char * from )
```

Makes a filename absolute from a relative filename.

```
#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

<i>out</i>	<i>to</i>	resulting absolute filename
<i>in</i>	<i>tolen</i>	size of the absolute filename buffer
<i>in</i>	<i>from</i>	relative filename

Returns

0 if no change, non zero otherwise

30.18.3.3 fl_filename_expand()

```
FL_EXPORT 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"             // does NOT handle ${VARNAME}
"${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

<i>out</i>	<i>to</i>	resulting expanded filename
<i>in</i>	<i>tolen</i>	size of the expanded filename buffer
<i>in</i>	<i>from</i>	filename containing shell variables

Returns

0 if no change, non zero otherwise

30.18.3.4 fl_filename_ext()

```
FL_EXPORT 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: NULL
```

Parameters

in	<i>buf</i>	the filename to be parsed
----	------------	---------------------------

Returns

a pointer to the extension (including '.') if any or NULL otherwise

30.18.3.5 fl_filename_free_list()

```
FL_EXPORT 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	<i>list</i>	table containing the resulting directory listing
in	<i>n</i>	number of entries in the list

30.18.3.6 fl_filename_isdir()

```
FL_EXPORT 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	<i>n</i>	the filename to parse
----	----------	-----------------------

Returns

non zero if file exists and is a directory, zero otherwise

30.18.3.7 fl_filename_list()

```
FL_EXPORT int fl_filename_list (
    const char * d,
    dirent *** list,
    F1_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	<i>d</i>	the name of the directory to list. It does not matter if it has a trailing slash.
out	<i>list</i>	table containing the resulting directory listing
in	<i>sort</i>	sorting functor: <ul style="list-style-type: none"> • fl_alpha sort: The files are sorted in ascending alphabetical order; upper and lowercase letters are compared according to their ASCII ordering uppercase before lowercase. • fl_casealpha sort: The files are sorted in ascending alphabetical order; upper and lowercase letters are compared equally case is not significant. • fl_casenumeric sort: 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_numeric sort: 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

30.18.3.8 fl_filename_match()

```
FL_EXPORT 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 must be matched exactly.

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

30.18.3.9 fl_filename_name()

```
FL_EXPORT 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.

30.18.3.10 fl_filename_relative()

```
FL_EXPORT int fl_filename_relative (
    char * to,
    int tolen,
    const char * from )
```

Makes a filename relative to the current working directory.

```
#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), "./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

<i>out</i>	<i>to</i>	resulting relative filename
<i>in</i>	<i>tolen</i>	size of the relative filename buffer
<i>in</i>	<i>from</i>	absolute filename

Returns

0 if no change, non zero otherwise

30.18.3.11 fl_filename_setext()

```
FL_EXPORT 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.

30.18.3.12 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

<i>uri</i>	The URI to open
<i>msg</i>	Optional buffer which contains the command or error message
<i>msglen</i>	Length of optional buffer

Returns

1 on success, 0 on failure

30.19 Fl_string

Functions

- FL_EXPORT char * [fl_strdup](#) (const char *s)
Cross platform interface to POSIX function strdup().

30.19.1 Detailed Description

30.19.2 Function Documentation

30.19.2.1 fl_strdup()

```
FL_EXPORT 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()

Chapter 31

Class Documentation

31.1 FI_Preferences::Entry Struct Reference

Public Attributes

- char * **name**
- char * **value**

The documentation for this struct was generated from the following file:

- FI_Preferences.H

31.2 FI Class Reference

The **FI** is the FLTK global (static) class containing state information and global methods for the current application.

```
#include <F1.H>
```

Public Types

- enum **FI_Option** {
OPTION_ARROW_FOCUS = 0, OPTION_VISIBLE_FOCUS, OPTION_DND_TEXT, OPTION_SHOW_TOOLTIPS,
OPTION_FNFC_USES_GTK, OPTION_PRINTER_USES_GTK, OPTION_SHOW_SCALING, OPTION_LAYOUT }

Enumerator for global FLTK options.

Static Public Member Functions

- static int `abi_check` (const int val=`FL_ABI_VERSION`)

Returns whether the runtime library ABI version is correct.
- static int `abi_version` ()

Returns the compiled-in value of the `FL_ABI_VERSION` constant.
- static int `add_awake_handler_` (`Fl_Awake_Handler`, void *)

Adds an awake handler for use in `awake()`.
- static void `add_check` (`Fl_Timeout_Handler`, void *=0)

FLTK will call this callback just before it flushes the display and waits for events.
- static 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.
- static void `add_fd` (int fd, int when, `Fl_FD_Handler cb`, void *=0)

Adds file descriptor fd to listen to.
- static void `add_fd` (int fd, `Fl_FD_Handler cb`, void *=0)

Adds file descriptor fd to listen to.
- static void `add_handler` (`Fl_Event_Handler h`)

Install a function to parse unrecognized events.
- static 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).
- static void `add_system_handler` (`Fl_System_Handler h`, void *data)

Install a function to intercept system events.
- static void `add_timeout` (double t, `Fl_Timeout_Handler`, void *=0)

Adds a one-shot timeout callback.
- static int `api_version` ()

Returns the compiled-in value of the `FL_API_VERSION` constant.
- static int `arg` (int argc, char **argv, int &i)

Parse a single switch from argv, starting at word i.
- static int `args` (int argc, char **argv, int &i, `Fl_Args_Handler cb`=0)

Parse command line switches using the cb argument handler.
- static void `args` (int argc, char **argv)

Parse all command line switches matching standard FLTK options only.
- static void `awake` (void *message=0)

Sends a message pointer to the main thread, causing any pending `Fl::wait()` call to terminate so that the main thread can retrieve the message and any pending redraws can be processed.
- static int `awake` (`Fl_Awake_Handler cb`, void *message=0)

See void `awake(void message=0)`.*
- static 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.
- static void `background2` (uchar, uchar, uchar)

Changes the alternative background color.
- static `Fl_Widget *` `belowmouse` ()

Gets the widget that is below the mouse.
- static void `belowmouse` (`Fl_Widget *`)

Sets the widget that is below the mouse.
- static int `box_border_radius_max` ()

Get the maximum border radius of all "rounded" boxtypes in pixels.
- static void `box_border_radius_max` (int R)

Set the maximum border radius of all "rounded" boxtypes in pixels.

- static [Fl_Color box_color \(Fl_Color\)](#)
Gets the drawing color to be used for the background of a box.
- static int [box_dh \(Fl_Boxtype\)](#)
Returns the height offset for the given boxtyle.
- static int [box_dw \(Fl_Boxtype\)](#)
Returns the width offset for the given boxtyle.
- static int [box_dx \(Fl_Boxtype\)](#)
Returns the X offset for the given boxtyle.
- static int [box_dy \(Fl_Boxtype\)](#)
Returns the Y offset for the given boxtyle.
- static int [box_shadow_width \(\)](#)
Get the box shadow width of all "shadow" boxtypes in pixels.
- static void [box_shadow_width \(int W\)](#)
Set the box shadow width of all "shadow" boxtypes in pixels.
- static 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.
- static bool [cairo_autolink_context \(\)](#)
Gets the current autolink mode for cairo support.
- static [cairo_t * cairo_cc \(\)](#)
Gets the current cairo context linked with a fltk window.
- static void [cairo_cc \(cairo_t *c, bool own=false\)](#)
Sets the current cairo context to c.
- static [cairo_t * cairo_make_current \(Fl_Window *w\)](#)
Provides a corresponding cairo context for window wi.
- static int [check \(\)](#)
Same as Fl::wait(0).
- static void [clear_widget_pointer \(Fl_Widget const *w\)](#)
Clears a widget pointer in the watch list.
- static int [clipboard_contains \(const char *type\)](#)
Returns non 0 if the clipboard contains data matching type.
- static int [compose \(int &del\)](#)
Any text editing widget should call this for each FL_KEYBOARD event.
- static void [compose_reset \(\)](#)
If the user moves the cursor, be sure to call [Fl::compose_reset\(\)](#).
- static void [copy \(const char *stuff, int len, int destination=0, const char *type=Fl::clipboard_plain_text\)](#)
Copies the data pointed to by stuff to the selection buffer (destination is 0), the clipboard (destination is 1), or both (destination is 2).
- static void [damage \(int d\)](#)
If true then [flush\(\)](#) will do something.
- static int [damage \(\)](#)
If true then [flush\(\)](#) will do something.
- static void [default_atclose \(Fl_Window *, void *\)](#)
Default callback for window widgets.
- static void [delete_widget \(Fl_Widget *w\)](#)
Schedules a widget for deletion at the next call to the event loop.
- static void [disable_im \(\)](#)
Disables the system input methods facilities.
- static void [display \(const char *\)](#)
Sets the X display to use for all windows.
- static int [dnd \(\)](#)

- Initiate a Drag And Drop operation.*
- static void [dnd_text_ops](#) (int v)

Sets whether drag and drop text operations are supported.
 - static int [dnd_text_ops](#) ()

Gets whether drag and drop text operations are supported.
 - static void [do_widget_deletion](#) ()

Deletes widgets previously scheduled for deletion.
 - static int [draw_box_active](#) ()

Determines if the currently drawn box is active or inactive.
 - static void [draw_GL_text_with_textures](#) (int val)

sets whether OpenGL uses textures to draw all text.
 - static int [draw_GL_text_with_textures](#) ()

returns whether whether OpenGL uses textures to draw all text.
 - static void [enable_im](#) ()

Enables the system input methods facilities.
 - static int [event](#) ()

Returns the last event that was processed.
 - static int [event_alt](#) ()

Returns non-zero if the Alt key is pressed.
 - static int [event_button](#) ()

Gets which particular mouse button caused the current event.
 - static int [event_button1](#) ()

Returns non-zero if mouse button 1 is currently held down.
 - static int [event_button2](#) ()

Returns non-zero if button 2 is currently held down.
 - static int [event_button3](#) ()

Returns non-zero if button 3 is currently held down.
 - static int [event_buttons](#) ()

Returns the mouse buttons state bits; if non-zero, then at least one button is pressed now.
 - static int [event_clicks](#) ()

Returns non zero if we had a double click event.
 - static void [event_clicks](#) (int i)

Manually sets the number returned by [Fl::event_clicks\(\)](#).
 - static void * [event_clipboard](#) ()

During an FL_PASTE event of non-textual data, returns a pointer to the pasted data.
 - static const char * [event_clipboard_type](#) ()

Returns the type of the pasted data during an FL_PASTE event.
 - static int [event_command](#) ()

Returns non-zero if the FL_COMMAND key is pressed, either FL_CTRL or on OSX FL_META.
 - static int [event_ctrl](#) ()

Returns non-zero if the Control key is pressed.
 - static void [event_dispatch](#) (Fl_Event_Dispatch d)

Set a new event dispatch function.
 - static Fl_Event_Dispatch [event_dispatch](#) ()

Return the current event dispatch function.
 - static int [event_dx](#) ()

Returns the current horizontal mouse scrolling associated with the FL_MOUSEWHEEL event.
 - static int [event_dy](#) ()

Returns the current vertical mouse scrolling associated with the FL_MOUSEWHEEL event.
 - static int [event_inside](#) (int, int, int, int)

Returns whether or not the mouse event is inside the given rectangle.

- static int `event_inside` (const `FL_Widget` *)

Returns whether or not the mouse event is inside a given child widget.
- static 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".
- static void `event_is_click` (int i)

Clears the value returned by `Fl::event_is_click()`.
- static int `event_key` ()

Gets which key on the keyboard was last pushed.
- static int `event_key` (int key)

Returns true if the given `key` was held down (or pressed) during the last event.
- static int `event_length` ()

Returns the length of the text in `Fl::event_text()`.
- static int `event_original_key` ()

Returns the keycode of the last key event, regardless of the NumLock state.
- static int `event_shift` ()

Returns non-zero if the Shift key is pressed.
- static int `event_state` ()

Returns the keyboard and mouse button states of the last event.
- static int `event_state` (int mask)

Returns non-zero if any of the passed event state bits are turned on.
- static const char * `event_text` ()

Returns the text associated with the current event, including `FL_PASTE` or `FL_DND_RELEASE` events.
- static int `event_x` ()

Returns the mouse position of the event relative to the `Fl_Window` it was passed to.
- static int `event_x_root` ()

Returns the mouse position on the screen of the event.
- static int `event_y` ()

Returns the mouse position of the event relative to the `Fl_Window` it was passed to.
- static int `event_y_root` ()

Returns the mouse position on the screen of the event.
- static `Fl_Window` * `first_window` ()

Returns the first top-level window in the list of shown() windows.
- static void `first_window` (`Fl_Window` *)

Sets the window that is returned by `first_window()`.
- static void `flush` ()

Causes all the windows that need it to be redrawn and graphics forced out through the pipes.
- static `Fl_Widget` * `focus` ()

Gets the current `Fl::focus()` widget.
- static void `focus` (`Fl_Widget` *)

Sets the widget that will receive `FL_KEYBOARD` events.
- static void `foreground` (uchar, uchar, uchar)

Changes `fl_color(FL_FOREGROUND_COLOR)`.
- static void `free_color` (`Fl_Color` i, int overlay=0)

Frees the specified color from the colormap, if applicable.
- static int `get_awake_handler` (`Fl_Awake_Handler` &, void *&)

Gets the last stored awake handler for use in `awake()`.
- static `Fl_Box_Draw_F` * `get_boxtyle` (`Fl_Boxtype`)

Gets the current box drawing function for the specified box type.
- static unsigned `get_color` (`Fl_Color` i)

Returns the RGB value(s) for the given FLTK color index.

- static void `get_color (Fl_Color i, uchar &red, uchar &green, uchar &blue)`
Returns the RGB value(s) for the given FLTK color index.
- static const char * `get_font (Fl_Font)`
Gets the string for this face.
- static const char * `get_font_name (Fl_Font, int *attributes=0)`
Get a human-readable string describing the family of this face.
- static int `get_font_sizes (Fl_Font, int *&sizesp)`
Return an array of sizes in sizesp.
- static int `get_key (int key)`
Returns true if the given key is held down now.
- static void `get_mouse (int &, int &)`
Return where the mouse is on the screen by doing a round-trip query to the server.
- static void `get_system_colors ()`
Read the user preference colors from the system and use them to call `Fl::foreground()`, `Fl::background()`, and `Fl::background2()`.
- static int `gl_visual (int, int *alist=0)`
This does the same thing as `Fl::visual(int)` but also requires OpenGL drawing to work.
- static `Fl_Window * grab ()`
Returns the window that currently receives all events.
- static void `grab (Fl_Window *)`
Selects the window to grab.
- static void `grab (Fl_Window &win)`
See `grab(Fl_Window)`*
- static int `h ()`
Returns the height in pixels of the main screen work area.
- static int `handle (int, Fl_Window *)`
Handle events from the window system.
- static int `handle_ (int, Fl_Window *)`
Handle events from the window system.
- static int `has_check (Fl_Timeout_Handler, void **=0)`
Returns 1 if the check exists and has not been called yet, 0 otherwise.
- static int `has_idle (Fl_Idle_Handler cb, void *data=0)`
Returns true if the specified idle callback is currently installed.
- static int `has_timeout (Fl_Timeout_Handler, void **=0)`
Returns true if the timeout exists and has not been called yet.
- static void `insertion_point_location (int x, int y, int height)`
Sets window coordinates and height of insertion point.
- static int `is_scheme (const char *name)`
Returns whether the current scheme is the given name.
- static void `keyboard_screen_scaling (int value)`
Controls the possibility to scale all windows by ctrl/+/-/0/ or cmd/+/-/0/.
- static int `lock ()`
The `lock()` method blocks the current thread until it can safely access FLTK widgets and data.
- static int `menu_linespacing ()`
Gets the default line spacing used by menus.
- static void `menu_linespacing (int H)`
Sets the default line spacing used by menus.
- static `Fl_Window * modal ()`
Returns the top-most `modal()` window currently shown.
- static `Fl_Window * next_window (const Fl_Window *)`
Returns the next top-level window in the list of `shown()` windows.

- static bool `option (Fl_Option opt)`
FLTK library options management.
- static void `option (Fl_Option opt, bool val)`
Override an option while the application is running.
- static void `own_colormap ()`
Makes FLTK use its own colormap.
- static 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.
- static void `paste (Fl_Widget &receiver)`
Backward compatibility only.
- static int `program_should_quit ()`
Returns non-zero when a request for program termination was received and accepted.
- static void `program_should_quit (int should_i)`
Indicate to the FLTK library whether a program termination request was received and accepted.
- static `Fl_Widget * pushed ()`
Gets the widget that is being pushed.
- static void `pushed (Fl_Widget *)`
Sets the widget that is being pushed.
- static `Fl_Widget * readqueue ()`
Reads the default callback queue and returns the first widget.
- static 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.
- static void `redraw ()`
Redraws all widgets.
- static void `release ()`
Releases the current grabbed window, equals `grab(0)`.
- static void `release_widget_pointer (Fl_Widget *&w)`
Releases a widget pointer from the watch list.
- static int `reload_scheme ()`
Called by scheme according to scheme name.
- static void `remove_check (Fl_Timeout_Handler, void *=0)`
Removes a check callback.
- static 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.
- static void `remove_fd (int, int when)`
Removes a file descriptor handler.
- static void `remove_fd (int)`
Removes a file descriptor handler.
- static void `remove_handler (Fl_Event_Handler h)`
Removes a previously added event handler.
- static void `remove_idle (Fl_Idle_Handler cb, void *data=0)`
Removes the specified idle callback, if it is installed.
- static void `remove_system_handler (Fl_System_Handler h)`
Removes a previously added system event handler.
- static void `remove_timeout (Fl_Timeout_Handler, void *=0)`
Removes a timeout callback.
- static void `repeat_timeout (double t, Fl_Timeout_Handler, void *=0)`
Repeats a timeout callback from the expiration of the previous timeout, allowing for more accurate timing.
- static void `reset_marked_text ()`
Resets marked text.

- static int **run** ()

Calls `Fl::wait()` repeatedly as long as any windows are displayed.
- static void **run_checks** ()
- static int **scheme** (const char *name)

Sets the current widget scheme.
- static const char * **scheme** ()

*See void `scheme(const char *name)`*
- static int **screen_count** ()

Gets the number of available screens.
- static void **screen_dpi** (float &h, float &v, int n=0)

Gets the screen resolution in dots-per-inch for the given screen.
- static Fl_Screen_Driver * **screen_driver** ()

Returns a pointer to the unique `Fl_Screen_Driver` object of the platform.
- static int **screen_num** (int x, int y)

Gets the screen number of a screen that contains the specified screen position x, y.
- static int **screen_num** (int x, int y, int w, int h)

Gets the screen number for the screen which intersects the most with the rectangle defined by x, y, w, h.
- static float **screen_scale** (int n)

Current value of the GUI scaling factor for screen number n.
- static void **screen_scale** (int n, float factor)

Set the value of the GUI scaling factor for screen number n.
- static int **screen_scaling_supported** ()

See if scaling factors are supported by this platform.
- static 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.
- static 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.
- static 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.
- static void **screen_xywh** (int &X, int &Y, int &W, int &H)

Gets the bounding box of a screen that contains the mouse pointer.
- static 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.
- static void **screen_xywh** (int &X, int &Y, int &W, int &H, int n)

Gets the screen bounding rect for the given screen.
- static 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.
- static int **scrollbar_size** ()

Gets the default scrollbar size used by `Fl_Browser_`, `Fl_Help_View`, `Fl_Scroll`, and `Fl_Text_Display` widgets.
- static 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.
- static void **selection** (Fl_Widget &owner, const char *, int len)

Changes the current selection.
- static Fl_Widget * **selection_owner** ()

back-compatibility only: Gets the widget owning the current selection
- static 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.
- static void **set_abort** (Fl_Abort_Handler f)

For back compatibility, sets the void `Fl::fatal` handler callback.
- static void **set_atclose** (Fl_Atclose_Handler f)

- For back compatibility, sets the `Fl::atclose` handler callback.*
- static void `set_box_color (Fl_Color)`

Sets the drawing color for the box that is currently drawn.
 - static void `set_boxtype (Fl_Boxtype, Fl_Box_Draw_F *, uchar, uchar, uchar, uchar)`

Sets the function to call to draw a specific boxtype.
 - static void `set_boxtype (Fl_Boxtype, Fl_Boxtype from)`

Copies the from boxtype.
 - static void `set_color (Fl_Color, uchar, uchar, uchar)`

Sets an entry in the fl_color index table.
 - static void `set_color (Fl_Color i, unsigned c)`

Sets an entry in the fl_color index table.
 - static void `set_font (Fl_Font, const char *)`

Changes a face.
 - static void `set_font (Fl_Font, Fl_Font)`

Copies one face to another.
 - static `Fl_Font set_fonts (const char *=0)`

FLTK will open the display, and add every fonts on the server to the face table.
 - static void `set_idle (Fl_Old_Idle_Handler cb)`

Sets an idle callback.
 - static 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.
 - static void `set_labeltype (Fl_Labeltype, Fl_Labeltype from)`

Sets the functions to call to draw and measure a specific labeltype.
 - static `Fl_System_Driver * system_driver ()`

Returns a pointer to the unique `Fl_System_Driver` object of the platform.
 - static 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`).
 - static void * `thread_message ()`

The `thread_message()` method returns the last message that was sent from a child by the `awake()` method.
 - static void `unlock ()`

The `unlock()` method releases the lock that was set using the `lock()` method.
 - static void `use_high_res_GL (int val)`

sets whether GL windows should be drawn at high resolution on Apple computers with retina displays
 - static int `use_high_res_GL ()`

returns whether GL windows should be drawn at high resolution on Apple computers with retina displays.
 - static double `version ()`

Returns the compiled-in value of the `FL_VERSION` constant.
 - static void `visible_focus (int v)`

Gets or sets the visible keyboard focus on buttons and other non-text widgets.
 - static int `visible_focus ()`

Gets or sets the visible keyboard focus on buttons and other non-text widgets.
 - static int `visual (int)`

Selects a visual so that your graphics are drawn correctly.
 - static int `w ()`

Returns the width in pixels of the main screen work area.
 - static int `wait ()`

Waits until "something happens" and then returns.
 - static double `wait (double time)`

Waits a maximum of `time_to_wait` seconds or until "something happens".
 - static void `watch_widget_pointer (Fl_Widget *&w)`

- Adds a widget pointer to the widget watch list.
- static int **x** ()

Returns the leftmost x coordinate of the main screen work area.
- static int **y** ()

Returns the topmost y coordinate of the main screen work area.

Static Public Attributes

- static void(* **atclose**)(FI_Window *, void *)

Back compatibility: default window callback handler.
- static bool **cfg_gfx_cairo** = 0

Cairo rendering available, available on many platforms.
- static bool **cfg_gfx_directx** = 0

DirectX rendering available, usually on Windows systems.
- static bool **cfg_gfx_gdi** = 0

GDI rendering available, usually on Windows systems.
- static bool **cfg_gfx_opengl** = 0

OpenGL rendering available, available on many platforms.
- static bool **cfg_gfx_quartz** = 0

Quartz rendering available, usually on OS X systems.
- static bool **cfg_gfx_xlib** = 0

X11 Xlib rendering available, usually on Linux systems.
- static bool **cfg_prn_gdi** = 0

GDI rendering available, usually on Windows systems.
- static bool **cfg_prn_ps** = 0

PostScript rendering available, usually on Linux systems.
- static bool **cfg_prn_quartz** = 0

Quartz rendering available, usually on OS X systems.
- static bool **cfg_sys_posix** = 0

Posix system available, usually on Linux and OS X systems, but also Cygwin.
- static bool **cfg_sys_win32** = 0

Windows system available, on Windows.
- static bool **cfg_win_cocoa** = 0

Cocoa window management available, usually on OS X systems.
- static bool **cfg_win_win32** = 0

Windows window management available, on low level Windows.
- static bool **cfg_win_x11** = 0

X11 window management available, usually on Linux systems.
- static char const *const **clipboard_image** = "image"

Denotes image data.
- static char const *const **clipboard_plain_text** = "text/plain"

Denotes plain textual data.
- static void(* **error**)(const char *,...) = FI_System_Driver::error

FLTK calls [FI::error\(\)](#) to output a normal error message.
- static void(* **fatal**)(const char *,...) = FI_System_Driver::fatal

FLTK calls [FI::fatal\(\)](#) to output a fatal error message.
- static const char *const **help** = helpmsg+13

Usage string displayed if [FI::args\(\)](#) detects an invalid argument.
- static void(* **idle**)()

The currently executing idle callback function: DO NOT USE THIS DIRECTLY!
- static void(* **warning**)(const char *,...) = FI_System_Driver::warning

FLTK calls [FI::warning\(\)](#) to output a warning message.

31.2.1 Detailed Description

The `Fl` is the FLTK global (static) class containing state information and global methods for the current application.

31.2.2 Member Enumeration Documentation

31.2.2.1 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), <code>Fl_Native_File_Chooser</code> 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_PRINTER_USES_GTK	When switched on (default), <code>Fl_Printer</code> 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 a transient yellow window the zoom factor value. When switched off, no such window gets displayed.
OPTION_LAST	For internal use only.

31.2.3 Member Function Documentation

31.2.3.1 abi_check()

```
static int Fl::abi_check (
    const int val = FL_ABI_VERSION ) [inline], [static]
```

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\(\)](#)

31.2.3.2 abi_version()

```
int Fl::abi_version () [static]
```

Returns the compiled-in value of the `FL_ABI_VERSION` constant.

This is useful for checking the version of a shared library.

31.2.3.3 add_awake_handler_()

```
int Fl::add_awake_handler_ (
    Fl_Awake_Handler func,
    void * data ) [static]
```

Adds an awake handler for use in [awake\(\)](#).

31.2.3.4 add_check()

```
void Fl::add_check (
    Fl_Timeout_Handler cb,
    void * argp = 0 ) [static]
```

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();
}
```

31.2.3.5 add_fd() [1/2]

```
void Fl::add_fd (
    int fd,
    int when,
    Fl_FD_Handler cb,
    void * d = 0 ) [static]
```

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.

31.2.3.6 add_fd() [2/2]

```
void Fl::add_fd (
    int fd,
    Fl_FD_Handler cb,
    void * d = 0 ) [static]
```

Adds file descriptor fd to listen to.

See [Fl::add_fd\(int fd, int when, Fl_FD_Handler cb, void* = 0\)](#) for details

31.2.3.7 add_idle()

```
void Fl::add_idle (
    Fl_Idle_Handler cb,
    void * data = 0 ) [static]
```

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. 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.

31.2.3.8 add_timeout()

```
void Fl::add_timeout (
    double t,
    Fl_Timeout_Handler cb,
    void * argp = 0 ) [static]
```

Adds a one-shot timeout callback.

The function will be called by [Fl::wait\(\)](#) at t seconds after this function is called. The optional void* argument is passed to the callback.

You can have multiple timeout callbacks. To remove a timeout callback use [Fl::remove_timeout\(\)](#).

If you need more accurate, repeated timeouts, use [Fl::repeat_timeout\(\)](#) to reschedule the subsequent timeouts.

The following code will print "TICK" each second on stdout with a fair degree of accuracy:

```
#include <stdio.h>
#include <FL/Fl.H>
#include <FL/Fl_Window.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();
}
```

31.2.3.9 api_version()

```
int Fl::api_version ( ) [static]
```

Returns the compiled-in value of the FL_API_VERSION constant.

This is useful for checking the version of a shared library.

31.2.3.10 arg()

```
int Fl::arg (
    int argc,
    char ** argv,
    int & i ) [static]
```

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\(\)](#).
- -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\(\)](#).

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.

31.2.3.11 args() [1/2]

```
int Fl::args (
    int argc,
    char ** argv,
    int & i,
    Fl_Args_Handler cb = 0 ) [static]
```

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.

31.2.3.12 args() [2/2]

```
void Fl::args (
    int argc,
    char ** argv ) [static]
```

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.

31.2.3.13 background()

```
void Fl::background (
    uchar r,
    uchar g,
    uchar b ) [static]
```

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.

31.2.3.14 background2()

```
void Fl::background2 (
    uchar r,
    uchar g,
    uchar b ) [static]
```

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_BACKGROUN2_COLOR.

31.2.3.15 box_border_radius_max() [1/2]

```
static int Fl::box_border_radius_max ( ) [inline], [static]
```

Get the maximum border radius of all "rounded" boxtypes in pixels.

Since

1.4.0

31.2.3.16 box_border_radius_max() [2/2]

```
static void Fl::box_border_radius_max (
    int R ) [inline], [static]
```

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

31.2.3.17 `box_color()`

```
Fl_Color Fl::box_color (
    Fl_Color c ) [static]
```

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 `inactive_r()` or `c` otherwise.

31.2.3.18 `box_dh()`

```
int Fl::box_dh (
    Fl_Boxtype t ) [static]
```

Returns the height offset for the given boxtyle.

See also

[box_dy\(\)](#).

31.2.3.19 `box_dw()`

```
int Fl::box_dw (
    Fl_Boxtype t ) [static]
```

Returns the width offset for the given boxtyle.

See also

[box_dy\(\)](#).

31.2.3.20 `box_dx()`

```
int Fl::box_dx (
    Fl_Boxtype t ) [static]
```

Returns the X offset for the given boxtyle.

See also

[box_dy\(\)](#)

31.2.3.21 box_dy()

```
int Fl::box_dy (   
    Fl_Boxtype t ) [static]
```

Returns the Y offset for the given boxtyle.

These functions return the offset values necessary for a given boxtyle, useful for computing the area inside a box's borders, to prevent overrawing the borders.

For instance, in the case of a boxtyle 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().

31.2.3.22 box_shadow_width() [1/2]

```
static int Fl::box_shadow_width ( ) [inline], [static]
```

Get the box shadow width of all "shadow" boxtypes in pixels.

Since

1.4.0

31.2.3.23 box_shadow_width() [2/2]

```
static void Fl::box_shadow_width (   
    int W ) [inline], [static]
```

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

31.2.3.24 check()

```
int Fl::check ( ) [static]
```

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).

31.2.3.25 damage()

```
static int Fl::damage ( ) [inline], [static]
```

If true then [flush\(\)](#) will do something.

31.2.3.26 display()

```
void Fl::display (
    const char * d ) [static]
```

Sets the X display to use for all windows.

Actually this just sets the environment variable \$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.

31.2.3.27 dnd_text_ops() [1/2]

```
static void Fl::dnd_text_ops (
    int v ) [inline], [static]
```

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.

31.2.3.28 dnd_text_ops() [2/2]

```
static int Fl::dnd_text_ops ( ) [inline], [static]
```

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.

31.2.3.29 draw_box_active()

```
int Fl::draw_box_active ( ) [static]
```

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\(Fl_Color c\)](#)

31.2.3.30 draw_GL_text_with_textures() [1/2]

```
static void Fl::draw_GL_text_with_textures (
    int val ) [inline], [static]
```

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

<code>val</code>	use 0 to prevent FLTK from drawing GL text with textures
------------------	--

See also

[gl_texture_pile_height\(int max\)](#)

Version

1.4.0

31.2.3.31 draw_GL_text_with_textures() [2/2]

```
static int Fl::draw_GL_text_with_textures ( ) [inline], [static]
```

returns whether OpenGL uses textures to draw all text.

Default is yes.

See also

[draw_GL_text_with_textures\(int val\)](#)

Version

1.4.0

31.2.3.32 flush()

```
void Fl::flush ( ) [static]
```

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.

31.2.3.33 foreground()

```
void Fl::foreground (
    uchar r,
    uchar g,
    uchar b ) [static]
```

Changes fl_color([FL_FOREGROUND_COLOR](#)).

31.2.3.34 get_awake_handler_()

```
int Fl::get_awake_handler_ (
    Fl_Awake_Handler & func,
    void *& data ) [static]
```

Gets the last stored awake handler for use in [awake\(\)](#).

31.2.3.35 get_boxtype()

```
Fl_Box_Draw_F * Fl::get_boxtype (
    Fl_Boxtype t ) [static]
```

Gets the current box drawing function for the specified box type.

31.2.3.36 get_system_colors()

```
void Fl::get_system_colors ( ) [static]
```

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.

31.2.3.37 gl_visual()

```
static int Fl::gl_visual (
    int ,
    int * alist = 0 ) [static]
```

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.

31.2.3.38 insertion_point_location()

```
void Fl::insertion_point_location (
    int x,
    int y,
    int height ) [static]
```

Sets window coordinates and height of insertion point.

See also

[Fl::compose\(int& del\)](#) for a detailed description.

31.2.3.39 is_scheme()

```
static int Fl::is_scheme (
    const char * name ) [inline], [static]
```

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	<i>name</i>	lowercase string of requested scheme name.
----	-------------	--

Returns

1 if the given scheme is active, 0 otherwise.

See also

[Fl::scheme\(const char *name\)](#)

31.2.3.40 menu_linespacing() [1/2]

```
int Fl::menu_linespacing ( ) [static]
```

Gets the default line spacing used by menus.

Returns

The default line spacing, in pixels.

31.2.3.41 menu_linespacing() [2/2]

```
void Fl::menu_linespacing (
    int H) [static]
```

Sets the default line spacing used by menus.

Default is 4.

Parameters

in	<i>H</i>	The new default line spacing between menu items, in pixels.
----	----------	---

31.2.3.42 option() [1/2]

```
bool Fl::option (
    Fl_Option opt) [static]
```

FLTK library options management.

This function needs to be documented in more detail. It can be used for more optional settings, such as using a native file chooser instead of the FLTK one wherever possible, disabling tooltips, disabling visible focus, disabling FLTK file chooser preview, etc. .

There should be a command line option interface.

There should be an application that manages options system wide, per user, and per application.

Example:

```
if ( Fl::option(Fl::OPTION_ARROW_FOCUS) )  
{ ..on.. }  
else  
{ ..off.. }
```

Note

As of FLTK 1.3.0, options can be managed within fluid, using the menu *Edit/Global FLTK Settings*.

Parameters

<i>opt</i>	which option
------------	--------------

Returns

true or false

See also

enum [Fl::Fl_Option](#)
[Fl::option\(Fl_Option, bool\)](#)

Since

FLTK 1.3.0

31.2.3.43 option() [2/2]

```
void Fl::option (   
    Fl_Option opt,  
    bool val )  [static]
```

Override an option while the application is running.

This function does not change any system or user settings.

Example:

```
Fl::option(Fl::OPTION_ARROW_FOCUS, true);      // on  
Fl::option(Fl::OPTION_ARROW_FOCUS, false);      // off
```

Parameters

<i>opt</i>	which option
<i>val</i>	set to true or false

See also

enum [Fl::Fl_Option](#)
 bool [Fl::option\(Fl_Option\)](#)

31.2.3.44 own_colormap()

```
void Fl::own_colormap ( ) [static]
```

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.

31.2.3.45 program_should_quit() [1/2]

```
static int Fl::program_should_quit ( ) [inline], [static]
```

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

31.2.3.46 program_should_quit() [2/2]

```
static void Fl::program_should_quit (
    int should_i ) [inline], [static]
```

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

31.2.3.47 readqueue()

```
Fl_Widget * Fl::readqueue ( ) [static]
```

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\(\)](#)

31.2.3.48 ready()

```
int Fl::ready ( ) [static]
```

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;
}
```

31.2.3.49 release()

```
static void Fl::release ( ) [inline], [static]
```

Releases the current grabbed window, equals `grab(0)`.

Deprecated Use `Fl::grab(0)` instead.

See also

- [grab\(Fl_Window*\)](#)

31.2.3.50 reload_scheme()

```
int Fl::reload_scheme ( ) [static]
```

Called by scheme according to scheme name.

Loads or reloads the current scheme selection. See void [scheme\(const char *name\)](#)

31.2.3.51 remove_check()

```
void Fl::remove_check (
    Fl_Timeout_Handler cb,
    void * argp = 0 ) [static]
```

Removes a check callback.

It is harmless to remove a check callback that no longer exists.

31.2.3.52 remove_fd() [1/2]

```
void Fl::remove_fd (
    int fd,
    int when ) [static]
```

Removes a file descriptor handler.

31.2.3.53 remove_fd() [2/2]

```
void Fl::remove_fd (
    int fd ) [static]
```

Removes a file descriptor handler.

31.2.3.54 remove_timeout()

```
void Fl::remove_timeout (
    Fl_Timeout_Handler cb,
    void * argp = 0 ) [static]
```

Removes a timeout callback.

It is harmless to remove a timeout callback that no longer exists.

Note

This version removes all matching timeouts, not just the first one. This may change in the future.

31.2.3.55 repeat_timeout()

```
void Fl::repeat_timeout (
    double t,
    Fl_Timeout_Handler cb,
    void * argp = 0 ) [static]
```

Repeats a timeout callback from the expiration of the previous timeout, allowing for more accurate timing.

You may only call this method inside a timeout callback of the same timer or at least a closely related timer, otherwise the timing accuracy can't be improved and the behavior is undefined.

The following code will print "TICK" each second on stdout with a fair degree of accuracy:

```
void callback(void*) {
    puts("TICK");
    Fl::repeat_timeout(1.0, callback);
}

int main() {
    Fl::add_timeout(1.0, callback);
    return Fl::run();
}
```

31.2.3.56 reset_marked_text()

```
void Fl::reset_marked_text ( ) [static]
```

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.

31.2.3.57 run()

```
int Fl::run ( ) [static]
```

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);
```

31.2.3.58 scheme()

```
int Fl::scheme (
    const char * s ) [static]
```

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 Mac OS X
- "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).

Uppercase scheme names are equivalent, but the stored scheme name will always be lowercase and [Fl::scheme\(\)](#) will return this lowercase name.

If the resulting scheme name is not defined, the default scheme will be used and [Fl::scheme\(\)](#) will return NULL.

See also

[Fl::is_scheme\(\)](#)

31.2.3.59 scrollbar_size() [1/2]

```
int Fl::scrollbar_size ( ) [static]
```

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.

31.2.3.60 scrollbar_size() [2/2]

```
void Fl::scrollbar_size (
    int W ) [static]
```

Sets the default scrollbar size that is used by the [Fl_Browser_](#), [Fl_Help_View](#), [Fl_Scroll](#), and [Fl_Text_Display](#) widgets.

Parameters

in	<i>W</i>	The new default size for widget scrollbars, in pixels.
----	----------	--

31.2.3.61 set_box_color()

```
void Fl::set_box_color (
    Fl_Color c ) [static]
```

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 [inactive_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\(Fl_Color\)](#)

31.2.3.62 set_boxtyppe() [1/2]

```
void Fl::set_boxtyppe (
    Fl_Boxtype t,
    Fl_Box_Draw_F * f,
    uchar a,
    uchar b,
    uchar c,
    uchar d ) [static]
```

Sets the function to call to draw a specific boxtyppe.

31.2.3.63 set_boxtyppe() [2/2]

```
void Fl::set_boxtyppe (
    Fl_Boxtype to,
    Fl_Boxtype from ) [static]
```

Copies the from boxtyppe.

31.2.3.64 set_idle()

```
static void Fl::set_idle (
    Fl_Old_Idle_Handler cb ) [inline], [static]
```

Sets an idle callback.

Deprecated This method is obsolete - use the [add_idle\(\)](#) method instead.

31.2.3.65 set_labeltype() [1/2]

```
void Fl::set_labeltype (
    Fl_Labeltype t,
    Fl_Label_Draw_F * f,
    Fl_Label_Measure_F * m ) [static]
```

Sets the functions to call to draw and measure a specific labeltype.

31.2.3.66 set_labeltype() [2/2]

```
static void Fl::set_labeltype (
    Fl_Labeltype ,
    Fl_Labeltype from ) [static]
```

Sets the functions to call to draw and measure a specific labeltype.

31.2.3.67 use_high_res_GL() [1/2]

```
static void Fl::use_high_res_GL (
    int val ) [inline], [static]
```

sets whether GL windows should be drawn at high resolution on Apple computers with retina displays

Version

1.3.4

31.2.3.68 use_high_res_GL() [2/2]

```
static int Fl::use_high_res_GL ( ) [inline], [static]
```

returns whether GL windows should be drawn at high resolution on Apple computers with retina displays.

Default is no.

Version

1.3.4

31.2.3.69 version()

```
double Fl::version ( ) [static]
```

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.

31.2.3.70 visible_focus() [1/2]

```
static void Fl::visible_focus (
    int v ) [inline], [static]
```

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.

31.2.3.71 visible_focus() [2/2]

```
static int Fl::visible_focus ( ) [inline], [static]
```

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.

31.2.3.72 visual()

```
int Fl::visual (
    int flags ) [static]
```

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.
- `Fl::visual(FL_DOUBLE|FL_INDEX)`
Hardware double buffering. Call this if you are going to use [Fl_Double_Window](#).
- `Fl::visual(FL_DOUBLE|FL_RGB)`
- `Fl::visual(FL_DOUBLE|FL_RGB8)`
Hardware double buffering and full color.

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).

31.2.3.73 wait() [1/2]

```
int Fl::wait ( ) [static]
```

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.

31.2.3.74 wait() [2/2]

```
double Fl::wait (
    double time_to_wait )  [static]
```

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).

31.2.4 Member Data Documentation

31.2.4.1 help

```
const char *const Fl::help = helppmsg+13  [static]
```

Usage string displayed if [Fl::args\(\)](#) detects an invalid argument.

This may be changed to point to customized text at run-time.

31.2.4.2 idle

```
void(* Fl::idle) ()  [static]
```

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\(\)](#)

The documentation for this class was generated from the following files:

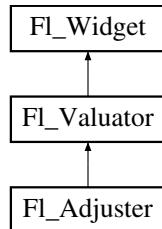
- [Fl.H](#)
- [Fl.cxx](#)
- [Fl_abort.cxx](#)
- [Fl_add_idle.cxx](#)
- [Fl_arg.cxx](#)
- [fl_boxtype.cxx](#)
- [fl_color.cxx](#)
- [Fl_compose.cxx](#)
- [Fl_display.cxx](#)
- [Fl_get_system_colors.cxx](#)
- [Fl_grab.cxx](#)
- [fl_labeltype.cxx](#)
- [Fl_lock.cxx](#)
- [Fl_own_colormap.cxx](#)
- [fl_set_font.cxx](#)
- [fl_shortcut.cxx](#)
- [Fl_visual.cxx](#)
- [Fl_Widget.cxx](#)
- [Fl_Window.cxx](#)
- [screen_xywh.cxx](#)
- [Fl_Cairo.cxx](#)

31.3 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.
- void [soft](#) (int s)
If "soft" is turned on, the user is allowed to drag the value outside the range.
- int [soft](#) () const
If "soft" is turned on, the user is allowed to drag the value outside the range.

Protected Member Functions

- void [draw](#) ()
Draws the widget.
- int [handle](#) (int)
Handles the specified event.
- void [value_damage](#) ()
Asks for partial redraw.

Additional Inherited Members

31.3.1 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.

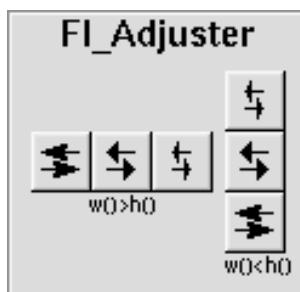


Figure 31.1 Fl_Adjuster

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 $100 * \text{step}()$, the next by $10 * \text{step}()$ and that smallest button by $\text{step}()$. Clicking on the buttons increments by 10 times the amount dragging by a pixel does. Shift + click decrements by 10 times the amount.

31.3.2 Constructor & Destructor Documentation

31.3.2.1 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.

31.3.3 Member Function Documentation

31.3.3.1 draw()

```
void Fl_Adjuster::draw ( ) [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](#).

31.3.3.2 handle()

```
int Fl_Adjuster::handle (
    int event ) [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.

Parameters

<code>in</code>	<code>event</code>	the kind of event received
-----------------	--------------------	----------------------------

Return values

<code>0</code>	if the event was not used or understood
<code>1</code>	if the event was used and can be deleted

See also

[Fl_Event](#)

Reimplemented from [Fl_Widget](#).

31.3.3.3 soft() [1/2]

```
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.

31.3.3.4 soft() [2/2]

```
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.

The documentation for this class was generated from the following files:

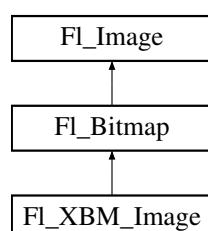
- [Fl_Adjuster.H](#)
- [Fl_Adjuster.cxx](#)

31.4 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

- virtual [FI_Image * copy](#) (int W, int H)
Creates a resized copy of the specified image.
- [FI_Image * copy \(\)](#)
- 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.
- void [draw](#) (int X, int Y)
- [FI_Bitmap](#) (const uchar *bits, int W, int H)
The constructors create a new bitmap from the specified bitmap data.
- [FI_Bitmap](#) (const char *bits, int W, int H)
The constructors create a new bitmap from the specified bitmap data.
- virtual void [label](#) ([FI_Widget](#) *w)
The [label\(\)](#) methods are an obsolete way to set the image attribute of a widget or menu item.
- virtual void [label](#) ([FI_Menu_Item](#) *m)
The [label\(\)](#) methods are an obsolete way to set the image attribute of a widget or menu item.
- virtual void [uncache](#) ()
If the image has been cached for display, delete the cache data.
- virtual ~[FI_Bitmap](#) ()
The destructor frees all memory and server resources that are used by the bitmap.

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 [FI_Graphics_Driver](#)

Additional Inherited Members

31.4.1 Detailed Description

The [FI_Bitmap](#) class supports caching and drawing of mono-color (bitmap) images.

Images are drawn using the current color.

31.4.2 Constructor & Destructor Documentation

31.4.2.1 Fl_Bitmap() [1/2]

```
Fl_Bitmap::Fl_Bitmap (
    const uchar * bits,
    int W,
    int H ) [inline]
```

The constructors create a new bitmap from the specified bitmap data.

31.4.2.2 Fl_Bitmap() [2/2]

```
Fl_Bitmap::Fl_Bitmap (
    const char * array,
    int W,
    int H ) [inline]
```

The constructors create a new bitmap from the specified bitmap data.

31.4.3 Member Function Documentation

31.4.3.1 copy()

```
Fl_Image * Fl_Bitmap::copy (
    int W,
    int H ) [virtual]
```

Creates a resized copy of the specified image.

The image should be deleted (or in the case of [Fl_Shared_Image](#), released) when you are done with it.

Parameters

<i>W,H</i>	width and height of the returned copied image
------------	---

Reimplemented from [Fl_Image](#).

31.4.3.2 draw()

```
void Fl_Bitmap::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.

In other words: `fl_push_clip(X, Y, W, H)` is applied, the image is drawn with its upper-left corner at X-cx, Y-cy and its own width and height, `fl_pop_clip()` is applied.

Reimplemented from [Fl_Image](#).

31.4.3.3 `label()` [1/2]

```
void Fl_Bitmap::label (  
    Fl_Widget * widget ) [virtual]
```

The [label\(\)](#) methods are an obsolete way to set the image attribute of a widget or menu item.

Use the `image()` or `deimage()` methods of the [Fl_Widget](#) and [Fl_Menu_Item](#) classes instead.

Reimplemented from [Fl_Image](#).

31.4.3.4 `label()` [2/2]

```
void Fl_Bitmap::label (  
    Fl_Menu_Item * m ) [virtual]
```

The [label\(\)](#) methods are an obsolete way to set the image attribute of a widget or menu item.

Use the `image()` or `deimage()` methods of the [Fl_Widget](#) and [Fl_Menu_Item](#) classes instead.

Reimplemented from [Fl_Image](#).

31.4.3.5 `uncache()`

```
void Fl_Bitmap::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 from [Fl_Image](#).

The documentation for this class was generated from the following files:

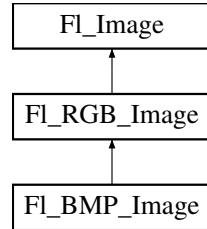
- [Fl_Bitmap.H](#)
- [Fl_Bitmap.cxx](#)

31.5 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)
Read a BMP image from memory.

Protected Member Functions

- void [load_bmp_](#) (class [Fl_Image_Reader](#) &rdr)

Additional Inherited Members

31.5.1 Detailed Description

The [Fl_BMP_Image](#) class supports loading, caching, and drawing of Windows Bitmap (BMP) image files.

31.5.2 Constructor & Destructor Documentation

31.5.2.1 [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	<i>filename</i>	a full path and name pointing to a valid BMP file.
----	-----------------	--

See also

[Fl_BMP_Image::Fl_BMP_Image\(const char *imagename, const unsigned char *data\)](#)

31.5.2.2 Fl_BMP_Image() [2/2]

```
Fl_BMP_Image::Fl_BMP_Image (
    const char * imagename,
    const unsigned char * data )
```

Read 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.

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	<i>imagename</i>	A name given to this image or NULL
in	<i>data</i>	Pointer to the start of the BMP image in memory. This code will not check for buffer overruns.

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:

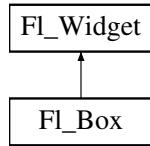
- [Fl_BMP_Image.H](#)
- [Fl_BMP_Image.cxx](#)

31.6 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 (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 *)`
See `Fl_Box::Fl_Box(int x, int y, int w, int h, const char * = 0)`
- virtual int `handle (int)`
Handles the specified event.

Protected Member Functions

- void `draw ()`
Draws the widget.

Additional Inherited Members

31.6.1 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.

31.6.2 Constructor & Destructor Documentation

31.6.2.1 Fl_Box()

```
Fl_Box::Fl_Box (
    int X,
    int Y,
    int W,
    int H,
    const char * l = 0 )
```

- The first 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(n)`.
- The second form of the constructor sets the box to the specified box type.

The destructor removes the box.

31.6.3 Member Function Documentation

31.6.3.1 draw()

```
void Fl_Box::draw ( )  [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](#).

31.6.3.2 handle()

```
int Fl_Box::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.

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:

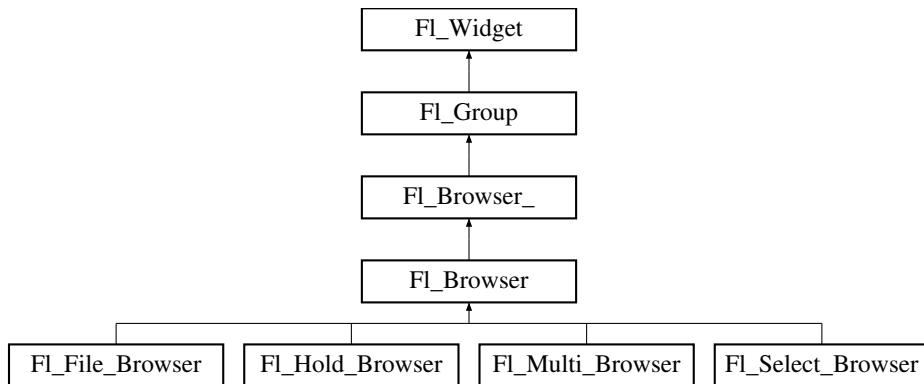
- Fl_Box.H
- Fl_Box.cxx

31.7 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 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** (int line)
Makes *line* invisible, preventing selection by the user.
- void **hide** ()
Hides the entire *Fl_Browser* widget – opposite of *show()*.
- void **icon** (int line, **Fl_Image** *icon)
Set the image *icon* for *line* to the value *icon*.
- **Fl_Image** * **icon** (int line) const
Returns the icon currently defined for *line*.
- 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** (int line)
Makes *line* visible, and available for selection by user.
- void **show** ()
Shows the entire *Fl_Browser* widget – opposite of *hide()*.

- 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.
- **FI_Fontsize** **textsize** () const
Gets the default text size (in pixels) for the lines in the browser.
- void **textsize** (**FI_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.
- **~FI_Browser** ()
The destructor deletes all list items and destroys the browser.

Protected Member Functions

- **FL_BLINE** * **_remove** (int line)
Removes the item at the specified line.
- **FL_BLINE** * **find_line** (int line) const
Returns the item for specified line.
- int **full_height** () const
*The height of the entire list of all **visible()** items in pixels.*
- int **incr_height** () const
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
Return the item at specified line.
- void **item_draw** (void *item, int X, int Y, int W, int H) const
Draws item at the position specified by X Y W H.
- void * **item_first** () const
Returns the very first item in the list.
- int **item_height** (void *item) const
Returns height of item in pixels.
- void * **item_last** () const
Returns the very last item in the list.
- void * **item_next** (void *item) const
Returns the next item after item.

- void * **item_prev** (void *item) const
Returns the previous item before item.
- void **item_select** (void *item, int val)
Change the selection state of item to the value val.
- int **item_selected** (void *item) const
See if item is selected.
- void **item_swap** (void *a, void *b)
Swap the items a and b.
- const char * **item_text** (void *item) const
Returns the label text for item.
- int **item_width** (void *item) const
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.

Additional Inherited Members

31.7.1 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 31.2 FL_Hold_Browser



Figure 31.3 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\(int\)](#).

31.7.2 Constructor & Destructor Documentation

31.7.2.1 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.

31.7.3 Member Function Documentation

31.7.3.1 _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	<i>line</i>	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\(\)](#)

31.7.3.2 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	<i>newtext</i>	The label text used for the added item
in	<i>d</i>	Optional user data() for the item (0 if unspecified)

See also

[add\(\)](#), [insert\(\)](#), [remove\(\)](#), [swap\(int,int\)](#), [clear\(\)](#)

31.7.3.3 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	<i>line</i>	The line to be displayed at the bottom.
----	-------------	---

See also

[topline\(\)](#), [middleline\(\)](#), [bottomline\(\)](#), [displayed\(\)](#), [lineposition\(\)](#)

31.7.3.4 clear()

```
void Fl_Browser::clear ( )
```

Removes all the lines in the browser.

See also

[add\(\)](#), [insert\(\)](#), [remove\(\)](#), [swap\(int,int\)](#), [clear\(\)](#)

31.7.3.5 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\(\)](#)

31.7.3.6 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\(\)](#)

31.7.3.7 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 its 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\(\)](#)

31.7.3.8 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\(\)](#)

31.7.3.9 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	<i>line</i>	The line number of the item whose data() is returned. (1 based)
----	-------------	---

Returns

The user data pointer (can be NULL)

31.7.3.10 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	<i>line</i>	The line of the item whose data() is to be changed. (1 based)
in	<i>d</i>	The new data to be assigned to the item. (can be NULL)

31.7.3.11 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\(\)](#)

31.7.3.12 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	<i>line</i>	The line to be checked
----	-------------	------------------------

Returns

1 if visible, 0 if not visible.

See also

[topline\(\)](#), [middleline\(\)](#), [bottomline\(\)](#), [displayed\(\)](#), [lineposition\(\)](#)

31.7.3.13 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	<i>line</i>	The line number of the item to return. (1 based)
----	-------------	--

Return values

<i>item</i>	that was found.
<i>NULL</i>	if line is out of range.

See also

[item_at\(\)](#), [find_line\(\)](#), [lineno\(\)](#)

31.7.3.14 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.

31.7.3.15 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

31.7.3.16 full_height()

```
int Fl_Browser::full_height ( ) const [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_](#).

31.7.3.17 hide() [1/2]

```
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	<i>line</i>	The line to be hidden. (1 based)
----	-------------	----------------------------------

See also

[show\(int\)](#), [hide\(int\)](#), [display\(\)](#), [visible\(\)](#), [make_visible\(\)](#)

31.7.3.18 hide() [2/2]

```
void Fl_Browser::hide ( ) [inline], [virtual]
```

Hides the entire [Fl_Browser](#) widget – opposite of [show\(\)](#).

Reimplemented from [Fl_Widget](#).

31.7.3.19 icon() [1/2]

```
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	<code>line</code>	The line to be modified. If out of range, nothing is done.
in	<code>icon</code>	The image icon to be assigned to the <code>line</code> . If NULL, any previous icon is removed.

31.7.3.20 icon() [2/2]

```
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	<code>line</code>	The line whose icon is returned.
----	-------------------	----------------------------------

Returns

The icon defined, or NULL if none.

31.7.3.21 incr_height()

```
int Fl_Browser::incr_height ( ) const [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_](#).

31.7.3.22 `insert()` [1/2]

```
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	<code>line</code>	The new line will be inserted above this line (1 based).
in	<code>item</code>	The item to be added.

31.7.3.23 `insert()` [2/2]

```
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	<code>line</code>	Line position for insert. (1 based) If <code>line > size()</code> , the entry will be added at the end.
in	<code>newtext</code>	The label text for the new line.
in	<code>d</code>	Optional pointer to user data to be associated with the new line.

31.7.3.24 `item_at()`

```
void* Fl_Browser::item_at (
    int line ) const [inline], [protected], [virtual]
```

Return the item at specified `line`.

Parameters

in	<i>line</i>	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_](#).

31.7.3.25 item_draw()

```
void Fl_Browser::item_draw (
    void * item,
    int X,
    int Y,
    int W,
    int H ) const [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	<i>item</i>	The item to be drawn
in	<i>X,Y,W,H</i>	position and size.

Implements [Fl_Browser_](#).

31.7.3.26 item_first()

```
void * Fl_Browser::item_first ( ) const [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_](#).

31.7.3.27 item_height()

```
int Fl_Browser::item_height (
    void * item ) const [protected], [virtual]
```

Returns height of *item* in pixels.

This takes into account embedded @ codes within the [text\(\)](#) label.

Parameters

in	<i>item</i>	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_](#).

31.7.3.28 item_last()

```
void * Fl_Browser::item_last ( ) const [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_](#).

31.7.3.29 item_next()

```
void * Fl_Browser::item_next (
    void * item ) const [protected], [virtual]
```

Returns the next item after `item`.

Parameters

in	<i>item</i>	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_](#).

31.7.3.30 item_prev()

```
void * Fl_Browser::item_prev (
    void * item ) const [protected], [virtual]
```

Returns the previous item before `item`.

Parameters

in	<i>item</i>	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_](#).

31.7.3.31 item_select()

```
void Fl_Browser::item_select (
    void * item,
    int val ) [protected], [virtual]
```

Change the selection state of `item` to the value `val`.

Parameters

in	<i>item</i>	The item to be changed.
in	<i>val</i>	The new selection state: 1 selects, 0 de-selects.

See also

[select\(\)](#), [selected\(\)](#), [value\(\)](#), [item_select\(\)](#), [item_selected\(\)](#)

Reimplemented from [Fl_Browser_](#).

31.7.3.32 item_selected()

```
int Fl_Browser::item_selected (
    void * item ) const [protected], [virtual]
```

See if `item` is selected.

Parameters

in	<i>item</i>	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_](#).

31.7.3.33 item_swap()

```
void Fl_Browser::item_swap (
    void * a,
    void * b ) [inline], [protected], [virtual]
```

Swap the items *a* and *b*.

You must call [redraw\(\)](#) to make any changes visible.

Parameters

in	<i>a,b</i>	the items to be swapped.
----	------------	--------------------------

See also

[swap\(int,int\)](#), [item_swap\(\)](#)

Reimplemented from [Fl_Browser_](#).

31.7.3.34 item_text()

```
const char * Fl_Browser::item_text (
    void * item ) const [protected], [virtual]
```

Returns the label text for *item*.

Parameters

in	<i>item</i>	The item whose label text is returned.
----	-------------	--

Returns

The item's text string. (Can be NULL)

Reimplemented from [Fl_Browser_](#).

31.7.3.35 item_width()

```
int Fl_Browser::item_width (
    void * item ) const [protected], [virtual]
```

Returns width of *item* in pixels.

This takes into account embedded @ codes within the [text\(\)](#) label.

Parameters

in	<i>item</i>	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_](#).

31.7.3.36 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	<i>item</i>	The item to be found
----	-------------	----------------------

Returns

The line number of the item, or 0 if not found.

See also

[item_at\(\)](#), [find_line\(\)](#), [lineno\(\)](#)

31.7.3.37 lineposition()

```
void Fl_Browser::lineposition (
    int line,
    Fl_Line_Position pos )
```

Updates the browser so that *line* is shown at position *pos*.

Parameters

in	<i>line</i>	line number. (1 based)
in	<i>pos</i>	position.

See also

[topline\(\)](#), [middleline\(\)](#), [bottomline\(\)](#)

31.7.3.38 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	<i>filename</i>	The filename to load
----	-----------------	----------------------

Returns

1 if OK, 0 on error (errno has reason)

See also

[add\(\)](#)

31.7.3.39 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	<i>line</i>	The line to be made visible.
----	-------------	------------------------------

See also

[show\(int\)](#), [hide\(int\)](#), [display\(\)](#), [visible\(\)](#), [make_visible\(\)](#)

31.7.3.40 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	<i>line</i>	The line to be displayed in the middle.
----	-------------	---

See also

[topline\(\)](#), [middleline\(\)](#), [bottomline\(\)](#), [displayed\(\)](#), [lineposition\(\)](#)

31.7.3.41 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	<i>to</i>	Destination line number (calculated <i>after</i> line <i>from</i> is removed)
in	<i>from</i>	Line number of item to be moved

31.7.3.42 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	<i>line</i>	Line to be removed. (1 based) If <i>line</i> is out of range, no action is taken.
----	-------------	--

See also

[add\(\)](#), [insert\(\)](#), [remove\(\)](#), [swap\(int,int\)](#), [clear\(\)](#)

31.7.3.43 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	<i>line</i>	The line whose icon is to be removed.
----	-------------	---------------------------------------

31.7.3.44 replace()

```
void Fl_Browser::replace (
    int a,
    const char * b ) [inline]
```

For back compatibility only.

31.7.3.45 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	<i>line</i>	The line number of the item to be changed. (1 based)
in	<i>val</i>	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\(\)](#)

31.7.3.46 selected()

```
int Fl_Browser::selected (
    int line ) const
```

Returns 1 if specified *line* is selected, 0 if not.

Parameters

in	<i>line</i>	The line being checked (1 based)
----	-------------	----------------------------------

Returns

1 if item selected, 0 if not.

See also

[select\(\)](#), [selected\(\)](#), [value\(\)](#), [item_select\(\)](#), [item_selected\(\)](#)

31.7.3.47 show() [1/2]

```
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	<i>line</i>	The line to be shown. (1 based)
----	-------------	---------------------------------

See also

[show\(int\)](#), [hide\(int\)](#), [display\(\)](#), [visible\(\)](#), [make_visible\(\)](#)

31.7.3.48 show() [2/2]

```
void Fl_Browser::show ( )  [inline], [virtual]
```

Shows the entire [Fl_Browser](#) widget – opposite of [hide\(\)](#).

Reimplemented from [Fl_Widget](#).

31.7.3.49 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.

31.7.3.50 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	<i>a,b</i>	The two items to be swapped.
----	------------	------------------------------

See also

[swap\(int,int\)](#), [item_swap\(\)](#)

31.7.3.51 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	<i>a,b</i>	The two lines to be swapped. (both 1 based)
----	------------	---

See also

[swap\(int,int\)](#), [item_swap\(\)](#)

31.7.3.52 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	<i>line</i>	The line number of the item whose text is returned. (1 based)
----	-------------	---

Returns

The text string (can be NULL)

31.7.3.53 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	<i>line</i>	The line of the item whose text will be changed. (1 based)
in	<i>newtext</i>	The new string to be assigned to the item.

31.7.3.54 `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.

31.7.3.55 `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.

31.7.3.56 `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	<i>line</i>	The line to be displayed at the top.
----	-------------	--------------------------------------

See also

[topline\(\)](#), [middleline\(\)](#), [bottomline\(\)](#), [displayed\(\)](#), [lineposition\(\)](#)

31.7.3.57 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\(\)](#)

31.7.3.58 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\(\)](#)

31.7.3.59 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	<i>line</i>	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:

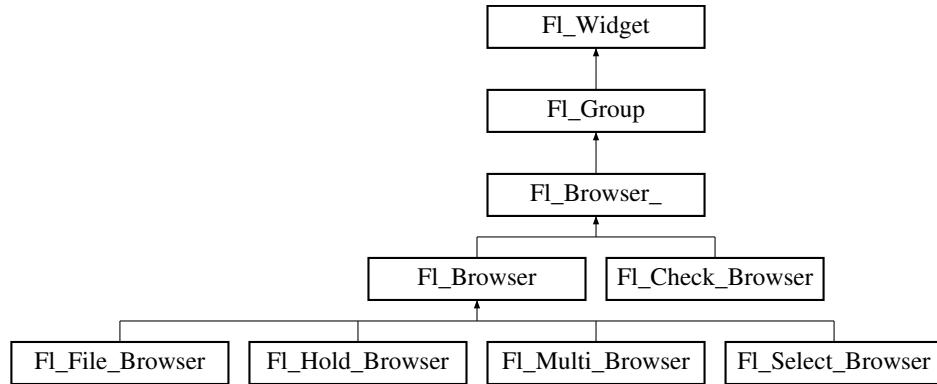
- Fl_Browser.H
- Fl_Browser.cxx
- Fl_Browser_load.cxx

31.8 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)
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 **position** () const

- Gets the vertical scroll position of the list as a pixel position *pos*.
 - void **position** (int *pos*)
Sets the vertical scroll position of the list to pixel position *pos*.
 - void **resize** (int X, int Y, int W, int H)
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 **scrollbar_width** () const
Returns the global value `Fl::scrollbar_size()`.
 - void **scrollbar_width** (int *width*)
Sets the global `Fl::scrollbar_size()`, and forces this instance of the widget to use it.
 - 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 textszie** () const
Gets the default text size (in pixels) for the lines in the browser.
 - void **textszie** (**Fl_Fontsize** *newSize*)
Sets the default text size (in pixels) for the lines in the browser to *size*.

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** ()

Draws the list within the normal widget bounding box.
- void * **find_item** (int ypos)

This method returns the item under mouse y position ypos.
- **FI_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.

Additional Inherited Members

31.8.1 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.

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.

31.8.2 Member Enumeration Documentation

31.8.2.1 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.

31.8.3 Constructor & Destructor Documentation

31.8.3.1 F1_Browser_()

```
F1_Browser_::F1_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.

31.8.4 Member Function Documentation

31.8.4.1 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)
-----	---------	---

31.8.4.2 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.
----	------	-------------------------

31.8.4.3 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	<i>d callbacks</i>	If non-zero, invokes widget callback if item changed. If 0, doesn't do callback (default).
----	--------------------	---

31.8.4.4 display()

```
void Fl_Browser_::display (
    void * item )
```

Displays the *item*, scrolling the list as necessary.

Parameters

in	<i>item</i>	The item to be displayed.
----	-------------	---------------------------

See also

[display\(\)](#), [displayed\(\)](#)

31.8.4.5 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	<i>item</i>	The item to check
----	-------------	-------------------

Returns

1 if visible, 0 if not visible.

See also

[display\(\)](#), [displayed\(\)](#)

31.8.4.6 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

31.8.4.7 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](#).

31.8.4.8 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.

31.8.4.9 handle()

```
int Fl_Browser_::handle (
    int event ) [virtual]
```

Handles the event within the normal widget bounding box.

Parameters

in	<i>event</i>	The event to process.
----	--------------	-----------------------

Returns

1 if event was processed, 0 if not.

Reimplemented from [Fl_Widget](#).

Reimplemented in [Fl_Check_Browser](#).

31.8.4.10 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.

31.8.4.11 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\(\)](#)

31.8.4.12 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	<i>pos</i>	The horizontal position (in pixels) to scroll the browser to.
----	------------	---

See also[position\(\)](#), [hposition\(\)](#)**31.8.4.13 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](#).

31.8.4.14 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	<i>a</i>	The starting item position
in	<i>b</i>	The new item being inserted

31.8.4.15 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	<i>index</i>	The index of the item to be returned
----	--------------	--------------------------------------

Returns

The item at the specified *index*.

Reimplemented in [Fl_Browser](#), and [Fl_Check_Browser](#).

31.8.4.16 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](#).

31.8.4.17 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	<i>item</i>	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_Browser](#), and [Fl_Check_Browser](#).

31.8.4.18 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](#).

31.8.4.19 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_Browser](#), and [Fl_Check_Browser](#).

31.8.4.20 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_Browser](#), and [Fl_Check_Browser](#).

31.8.4.21 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	<i>item</i>	The item whose height to return.
----	-------------	----------------------------------

Returns

The height, in pixels.

31.8.4.22 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	<i>item</i>	The item to be selected
in	<i>val</i>	The optional selection state; 1=select, 0=de-select. The default is to select the item (1).

Reimplemented in [Fl_Browser](#), and [Fl_Check_Browser](#).

31.8.4.23 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	<i>item</i>	The item to test.
----	-------------	-------------------

Reimplemented in [Fl_Browser](#), and [Fl_Check_Browser](#).

31.8.4.24 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	<i>a,b</i>	The two items to be swapped.
----	------------	------------------------------

Reimplemented in [Fl_Browser](#), and [Fl_Check_Browser](#).

31.8.4.25 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	<i>item</i>	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](#).

31.8.4.26 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	<i>item</i>	The item whose width is returned.
----	-------------	-----------------------------------

Returns

The width of the item in pixels.

Implemented in [Fl_Browser](#), and [Fl_Check_Browser](#).

31.8.4.27 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\(\)](#)

31.8.4.28 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.

31.8.4.29 position() [1/2]

```
int Fl_Browser_::position ( ) 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\(\)](#)

31.8.4.30 position() [2/2]

```
void Fl_Browser_::position ( 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

<code>in</code>	<code>pos</code>	The vertical position (in pixels) to scroll the browser to.
-----------------	------------------	---

See also

[position\(\)](#), [hposition\(\)](#)

31.8.4.31 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

<code>in</code>	<code>item</code>	The item that needs to be redrawn.
-----------------	-------------------	------------------------------------

See also

[redraw_lines\(\)](#), [redraw_line\(\)](#)

31.8.4.32 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\(\)](#)

31.8.4.33 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	<i>a</i>	Item being replaced
in	<i>b</i>	Item to replace 'a'

31.8.4.34 resize()

```
void Fl_Browser_::resize (
    int X,
    int Y,
    int W,
    int H ) [virtual]
```

Repositions and/or resizes the browser.

Parameters

in	<i>X,Y,W,H</i>	The new position and size for the browser, in pixels.
----	----------------	---

Reimplemented from [Fl_Widget](#).

31.8.4.35 scrollbar_left()

```
void Fl_Browser_::scrollbar_left () [inline]
```

Moves the vertical scrollbar to the lefthand side of the list.

For back compatibility.

31.8.4.36 scrollbar_right()

```
void Fl_Browser_::scrollbar_right () [inline]
```

Moves the vertical scrollbar to the righthand side of the list.

For back compatibility.

31.8.4.37 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\(int\)](#)

31.8.4.38 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	<i>newSize</i>	Sets the scrollbar size in pixels. If 0 (default), scrollbar size tracks the global Fl::scrollbar_size()
----	----------------	---

See also

[Fl::scrollbar_size\(\)](#)

31.8.4.39 scrollbar_width() [1/2]

```
int Fl_Browser_::scrollbar_width ( ) const [inline]
```

Returns the global value [Fl::scrollbar_size\(\)](#).

Deprecated Use [scrollbar_size\(\)](#) instead.

Todo This method should eventually be removed in 1.4+

31.8.4.40 scrollbar_width() [2/2]

```
void Fl_Browser_::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 in 1.4+

31.8.4.41 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	<i>item</i>	The item whose selection state is to be changed
in	<i>val</i>	The new selection state (1=select, 0=de-select)
in	<i>docallbacks</i>	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.

31.8.4.42 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	<i>item</i>	The item to select.
in	<i>docallbacks</i>	If non-zero, invokes widget callback if item changed. If 0, doesn't do callback (default).

31.8.4.43 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.

31.8.4.44 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	<code>flags</code>	FL_SORT_ASCENDING – sort in ascending order FL_SORT_DESCENDING – sort in descending order Values other than the above will cause undefined behavior Other flags may appear in the future.
----	--------------------	--

Todo Add a flag to ignore case

31.8.4.45 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	<code>a,b</code>	Items being swapped.
----	------------------	----------------------

31.8.4.46 textfont()

```
Fl_Font Fl_Browser_::textfont ( ) const [inline]
```

Gets the default text font for the lines in the browser.

See also

[textfont\(\)](#), [textsize\(\)](#), [textcolor\(\)](#)

31.8.5 Member Data Documentation

31.8.5.1 hscrollbar

`Fl_Scrollbar Fl_Browser_::hscrollbar`

Horizontal scrollbar.

Public, so that it can be accessed directly.

31.8.5.2 scrollbar

`Fl_Scrollbar Fl_Browser_::scrollbar`

Vertical scrollbar.

Public, so that it can be accessed directly.

The documentation for this class was generated from the following files:

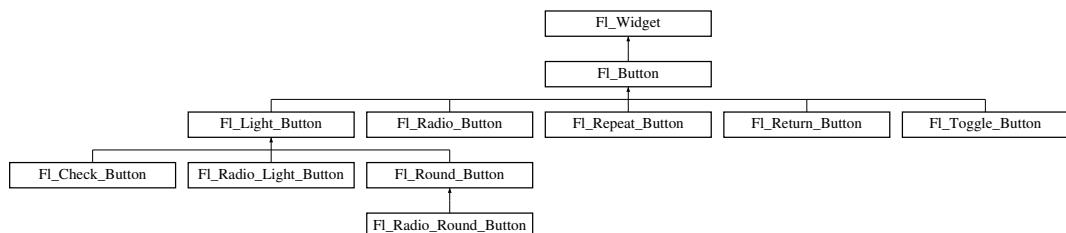
- `Fl_Browser_.H`
- `Fl_Browser_.cxx`

31.9 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).
- `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.
- virtual int `handle (int)`
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 (int s)`
Sets the shortcut key to s.
- void `shortcut (const char *s)`
(for backwards compatibility)
- int `value (int v)`
Sets the current value of the button.
- char `value () const`
Returns the current value of the button (0 or 1).

Protected Member Functions

- virtual void `draw ()`
Draws the widget.
- void `simulate_key_action ()`

Static Protected Member Functions

- static void `key_release_timeout (void *)`

Static Protected Attributes

- static `Fl_Widget_Tracker * key_release_tracker = 0`

Additional Inherited Members

31.9.1 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\(\)](#). 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\(int\)](#)

Todo Refactor the doxygen comments for [Fl_Button type\(\)](#) documentation.

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.

Todo Refactor the doxygen comments for [Fl_Button when\(\)](#) documentation.

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.
- [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).

31.9.2 Constructor & Destructor Documentation

31.9.2.1 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.

Parameters

in	<i>X,Y,W,H</i>	position and size of the widget
in	<i>L</i>	widget label, default is no label

31.9.3 Member Function Documentation**31.9.3.1 clear()**

```
int Fl_Button::clear ( ) [inline]
```

Same as [value\(0\)](#).

See also

[value\(int v\)](#)

31.9.3.2 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

<i>Fl_Boxtype</i>	<input type="button" value=""/>
-------------------	---------------------------------

31.9.3.3 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	<i>b</i>	down box type
----	----------	---------------

31.9.3.4 draw()

```
void Fl_Button::draw ( )  [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](#), and [Fl_Return_Button](#).

31.9.3.5 handle()

```
int Fl_Button::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.

Parameters

in	<i>event</i>	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_Return_Button](#), and [Fl_Repeat_Button](#).

31.9.3.6 set()

```
int Fl_Button::set ( ) [inline]
```

Same as `value(1)`.

See also[value\(int v\)](#)**31.9.3.7 shortcut() [1/2]**

```
int Fl_Button::shortcut ( ) const [inline]
```

Returns the current shortcut key for the button.

Return values

<i>int</i>	
------------	--

31.9.3.8 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
----	---	-----------------------------------

31.9.3.9 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

31.10 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 \(void *c\)](#)
Sets the gc c to keep track on.
- void * [gc \(\) const](#)
Gets the last gc attached to a cc.
- void [window \(void *w\)](#)
Sets the window w to keep track on.
- void * [window \(\) const](#)
Gets the last window attached to a cc.

31.10.1 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 when configure has the –enable-cairo option

31.10.2 Member Function Documentation

31.10.2.1 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:

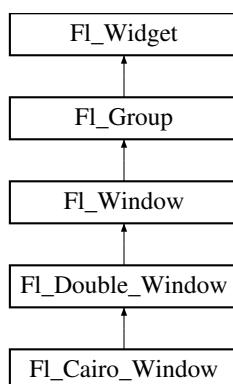
- Fl_Cairo.H
- Fl_Cairo.cxx

31.11 Fl_Cairo_Window Class Reference

This defines a 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)`
This defines the cairo draw callback prototype that you must further.

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 which will implement your cairo rendering.

Protected Member Functions

- `void draw ()`
Overloaded to provide cairo callback support.

Additional Inherited Members

31.11.1 Detailed Description

This defines a 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.

Note

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 it only when your window is the current window.

See also

[Fl_Window::current\(\)](#)

31.11.2 Member Function Documentation

31.11.2.1 set_draw_cb()

```
void Fl_Cairo_Window::set_draw_cb (
    cairo_draw_cb cb ) [inline]
```

You must provide a draw callback which will implement your cairo rendering.

This method will permit you to set your cairo callback to `cb`.

The documentation for this class was generated from the following file:

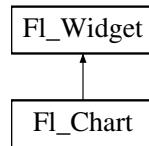
- `Fl_Cairo_Window.H`

31.12 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)
Add the data value `val` with optional label `str` and color `col` to the chart.
- uchar [autosize](#) () const
Get whether the chart will automatically adjust the bounds of the chart.
- void [autosize](#) (uchar n)
Set 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)
Set the maximum number of data values for a chart.
- void [replace](#) (int ind, double val, const char *str=0, unsigned col=0)
Replace 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)
- [Fl_Color](#) [textcolor](#) () const
Gets the chart's text color.
- void [textcolor](#) ([Fl_Color](#) n)
gets 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)
gets the chart's text size to `s`.
- [~Fl_Chart](#) ()
Destroys the `Fl_Chart` widget and all of its data.

Protected Member Functions

- void [draw \(\)](#)
Draws the widget.

Additional Inherited Members

31.12.1 Detailed Description

[Fl_Chart](#) displays simple charts.

It is provided for Forms compatibility.

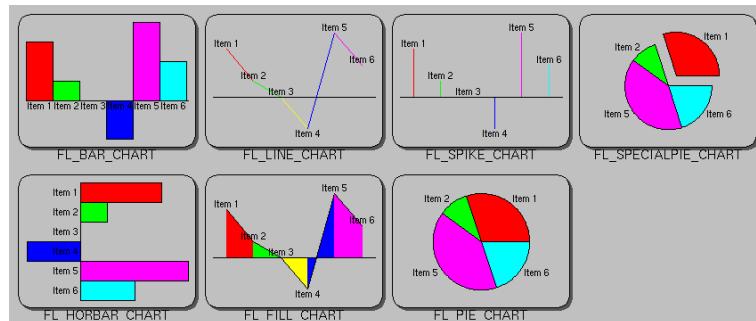


Figure 31.4 Fl_Chart

Todo Refactor [Fl_Chart::type\(\)](#) information.

The type of an [Fl_Chart](#) object can be set using [type\(uchar t\)](#) to:

- **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.
- **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.

31.12.2 Constructor & Destructor Documentation

31.12.2.1 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	<i>X,Y,W,H</i>	position and size of the widget
in	<i>L</i>	widget label, default is no label

31.12.3 Member Function Documentation**31.12.3.1 add()**

```
void Fl_Chart::add (
    double val,
    const char * str = 0,
    unsigned col = 0 )
```

Add the data value `val` with optional label `str` and color `col` to the chart.

Parameters

in	<i>val</i>	data value
in	<i>str</i>	optional data label
in	<i>col</i>	optional data color

31.12.3.2 autosize() [1/2]

```
uchar Fl_Chart::autosize ( ) const [inline]
```

Get whether the chart will automatically adjust the bounds of the chart.

Returns

non-zero if auto-sizing is enabled and zero if disabled.

31.12.3.3 autosize() [2/2]

```
void Fl_Chart::autosize (
    uchar n ) [inline]
```

Set whether the chart will automatically adjust the bounds of the chart.

Parameters

in	<i>n</i>	non-zero to enable automatic resizing, zero to disable.
----	----------	---

31.12.3.4 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	<i>a,b</i>	are set to lower, upper
-----	------------	-------------------------

31.12.3.5 bounds() [2/2]

```
void Fl_Chart::bounds (
    double a,
    double b )
```

Sets the lower and upper bounds of the chart values.

Parameters

in	<i>a,b</i>	are used to set lower, upper
----	------------	------------------------------

31.12.3.6 draw()

```
void Fl_Chart::draw ( ) [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](#).

31.12.3.7 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	<i>ind</i>	insertion position
in	<i>val</i>	data value
in	<i>str</i>	optional data label
in	<i>col</i>	optional data color

31.12.3.8 maxsize()

```
void Fl_Chart::maxsize (
    int m )
```

Set 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	<i>m</i>	maximum number of data values allowed.
----	----------	--

31.12.3.9 replace()

```
void Fl_Chart::replace (
    int ind,
    double val,
    const char * str = 0,
    unsigned col = 0 )
```

Replace a data value *val* at the given position *ind*.

Position 1 is the first data value.

Parameters

in	<i>ind</i>	insertion position
in	<i>val</i>	data value
in	<i>str</i>	optional data label
in	<i>col</i>	optional data color

31.12.3.10 textcolor()

```
void Fl_Chart::textcolor (
    Fl_Color n ) [inline]
```

gets the chart's text color to n.

31.12.3.11 textfont()

```
void Fl_Chart::textfont (
    Fl_Font s ) [inline]
```

Sets the chart's text font to s.

31.12.3.12 textszie()

```
void Fl_Chart::textszie (
    Fl_Fontsize s ) [inline]
```

gets the chart's text size to s.

The documentation for this class was generated from the following files:

- Fl_Chart.H
- Fl_Chart.cxx

31.13 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.

31.13.1 Detailed Description

For internal use only.

31.13.2 Member Data Documentation

31.13.2.1 col

```
unsigned FL_CHART_ENTRY::col
```

For internal use only.

31.13.2.2 str

```
char FL_CHART_ENTRY::str[FL_CHART_LABEL_MAX+1]
```

For internal use only.

31.13.2.3 val

```
float FL_CHART_ENTRY::val
```

For internal use only.

The documentation for this struct was generated from the following file:

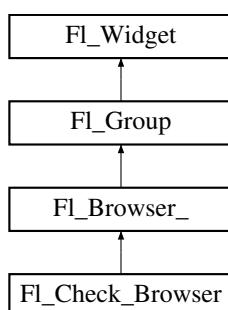
- Fl_Chart.H

31.14 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

This method must be provided by the subclass to return the item for the specified index.
- void **item_swap** (int ia, int ib)

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.

Protected Member Functions

- int **handle** (int)

Handles the event within the normal widget bounding box.
- void **item_draw** (void *, int, int, int, int) const

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

This method must be provided by the subclass to return the first item in the list.

- int `item_height` (void *) const

This method must be provided by the subclass to return the height of `item` in pixels.

- void * `item_next` (void *) const

This method must be provided by the subclass to return the item in the list after `item`.

- void * `item_prev` (void *) const

This method must be provided by the subclass to return the item in the list before `item`.

- void `item_select` (void *, int)

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

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

This optional method returns a string (label) that may be used for sorting.

- int `item_width` (void *) const

This method must be provided by the subclass to return the width of the `item` in pixels.

Additional Inherited Members

31.14.1 Detailed Description

The `Fl_Check_Browser` widget displays a scrolling list of text lines that may be selected and/or checked by the user.

31.14.2 Constructor & Destructor Documentation

31.14.2.1 `Fl_Check_Browser()`

```
Fl_Check_Browser::Fl_Check_Browser (
    int X,
    int Y,
    int W,
    int H,
    const char * l = 0 )
```

The constructor makes an empty browser.

31.14.2.2 `~Fl_Check_Browser()`

```
Fl_Check_Browser::~Fl_Check_Browser ( ) [inline]
```

The destructor deletes all list items and destroys the browser.

31.14.3 Member Function Documentation

31.14.3.1 add() [1/2]

```
int Fl_Check_Browser::add (
    char * s )
```

Add a new unchecked line to the end of the browser.

See also

[add\(char *s, int b\)](#)

31.14.3.2 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.

31.14.3.3 check_all()

```
void Fl_Check_Browser::check_all ( )
```

Sets all the items checked.

31.14.3.4 check_none()

```
void Fl_Check_Browser::check_none ( )
```

Sets all the items unchecked.

31.14.3.5 checked() [1/2]

```
int Fl_Check_Browser::checked (
    int i ) const
```

Gets the current status of item item.

31.14.3.6 `checked()` [2/2]

```
void Fl_Check_Browser::checked (
    int i,
    int b )
```

Sets the check status of item item to b.

31.14.3.7 `clear()`

```
void Fl_Check_Browser::clear ( )
```

Remove every item from the browser.

31.14.3.8 `handle()`

```
int Fl_Check_Browser::handle (
    int event ) [protected], [virtual]
```

Handles the event within the normal widget bounding box.

Parameters

in	<i>event</i>	The event to process.
----	--------------	-----------------------

Returns

1 if event was processed, 0 if not.

Reimplemented from [Fl_Browser_](#).

31.14.3.9 `item_at()`

```
void * Fl_Check_Browser::item_at (
    int index ) const [virtual]
```

This method must be provided by the subclass to return the item for the specified index.

Parameters

in	<i>index</i>	The index of the item to be returned
----	--------------	--------------------------------------

Returns

The item at the specified `index`.

Reimplemented from [Fl_Browser_](#).

31.14.3.10 item_first()

```
void * Fl_Check_Browser::item_first ( ) const [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_](#).

31.14.3.11 item_height()

```
int Fl_Check_Browser::item_height (
    void * item ) const [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	<code>item</code>	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_](#).

31.14.3.12 item_next()

```
void * Fl_Check_Browser::item_next (
    void * item ) const [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_](#).

31.14.3.13 item_prev()

```
void * Fl_Check_Browser::item_prev (
    void * item ) const [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_](#).

31.14.3.14 item_select()

```
void Fl_Check_Browser::item_select (
    void * item,
    int val ) [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	<code>item</code>	The item to be selected
in	<code>val</code>	The optional selection state; 1=select, 0=de-select. The default is to select the item (1).

Reimplemented from [Fl_Browser_](#).

31.14.3.15 item_selected()

```
int Fl_Check_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 from [Fl_Browser_](#).

31.14.3.16 item_swap()

```
void Fl_Check_Browser::item_swap (
    void * a,
    void * b ) [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_](#).

31.14.3.17 item_text()

```
const char * Fl_Check_Browser::item_text (
    void * item ) const [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_](#).

31.14.3.18 item_width()

```
int Fl_Check_Browser::item_width (
    void * item ) const [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	<i>item</i>	The item whose width is returned.
----	-------------	-----------------------------------

Returns

The width of the item in pixels.

Implements [Fl_Browser_](#).

31.14.3.19 nchecked()

```
int Fl_Check_Browser::nchecked ( ) const [inline]
```

Returns how many items are currently checked.

31.14.3.20 nitems()

```
int Fl_Check_Browser::nitems ( ) const [inline]
```

Returns how many lines are in the browser.

The last line number is equal to this.

31.14.3.21 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.

31.14.3.22 set_checked()

```
void Fl_Check_Browser::set_checked (
    int item ) [inline]
```

Equivalent to Fl_Check_Browser::checked(item, 1).

31.14.3.23 text()

```
char * Fl_Check_Browser::text (
    int i ) const
```

Return a pointer to an internal buffer holding item item's text.

31.14.3.24 value()

```
int Fl_Check_Browser::value ( ) const
```

Returns the index of the currently selected item.

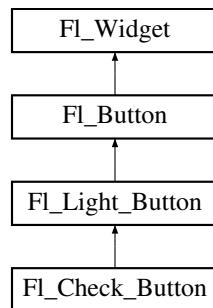
The documentation for this class was generated from the following files:

- Fl_Check_Browser.H
- Fl_Check_Browser.cxx

31.15 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.

Additional Inherited Members

31.15.1 Detailed Description

A button with a "checkmark" to show its status.



Figure 31.5 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.

31.15.2 Constructor & Destructor Documentation

31.15.2.1 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:

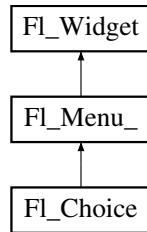
- [Fl_Check_Button.H](#)
- [Fl_Check_Button.cxx](#)

31.16 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)
Handles the specified event.
- int **value** () const
Gets the index of the last item chosen by the user.
- int **value** (int v)
Sets the currently selected value using the index into the menu item array.
- int **value** (const Fl_Menu_Item *v)
Sets the currently selected value using a pointer to menu item.

Protected Member Functions

- void **draw** ()
Draws the widget.

Additional Inherited Members

31.16.1 Detailed Description

A button that is used to pop up a menu.

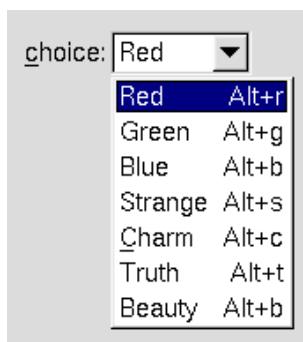


Figure 31.6 Fl_Choice

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_Choice::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.

Todo Refactor the doxygen comments for [Fl_Choice changed\(\)](#) documentation.

- int [Fl_Widget::changed\(\) const](#) This value is true 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.
- Fl_Boxtype [Fl_Choice::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_Choice::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();
}
```

31.16.2 Constructor & Destructor Documentation

31.16.2.1 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 boxtyle 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

31.16.3 Member Function Documentation

31.16.3.1 draw()

```
void Fl_Choice::draw ( ) [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](#).

31.16.3.2 handle()

```
int Fl_Choice::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.

Parameters

in	<i>event</i>	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](#).

31.16.3.3 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.

31.16.3.4 value() [2/3]

```
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

<code>in</code>	<code>v</code>	index of value in the menu item array.
-----------------	----------------	--

Returns

non-zero if the new value is different to the old one.

31.16.3.5 value() [3/3]

```
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

<code>in</code>	<code>v</code>	pointer to menu item 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:

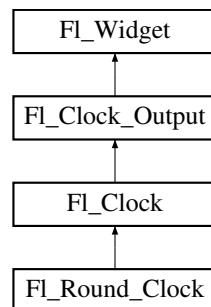
- Fl_Choice.H
- Fl_Choice.cxx

31.17 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)
Handles the specified event.
- **`~FL_Clock`** ()
The destructor removes the clock.

Additional Inherited Members

31.17.1 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.

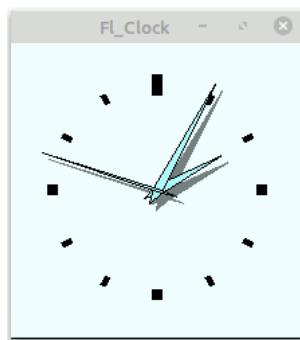


Figure 31.7 `FL_SQUARE_CLOCK` type



Figure 31.8 `FL_ROUND_CLOCK` type

See also

class `FL_Clock_Output`

31.17.2 Constructor & Destructor Documentation

31.17.2.1 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 boxtyle is `FL_UP_BOX`.

Parameters

in	<code>X,Y,W,H</code>	position and size of the widget
in	<code>L</code>	widget label, default is no label

31.17.2.2 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 boxtyle 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 boxtyle with [box\(\)](#).

Parameters

in	<code>t</code>	type of clock: <code>FL_ROUND_CLOCK</code> or <code>FL_SQUARE_CLOCK</code> (0)
in	<code>X,Y,W,H</code>	position and size of the widget
in	<code>L</code>	widget label, default is no label

See also

class [Fl_Clock_Output](#)

31.17.3 Member Function Documentation

31.17.3.1 handle()

```
int Fl_Clock::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.

Parameters

in	<i>event</i>	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:

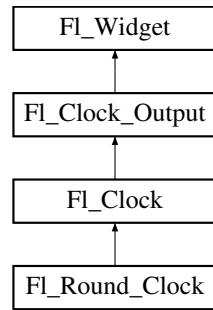
- [Fl_Clock.H](#)
- [Fl_Clock.cxx](#)

31.18 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.
- void **value** (ulong v)
Set the displayed time.
- void **value** (int H, int m, int s)
Set the displayed time.
- ulong **value** () const
Returns the displayed time.

Protected Member Functions

- void **draw** ()
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.

Additional Inherited Members

31.18.1 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.

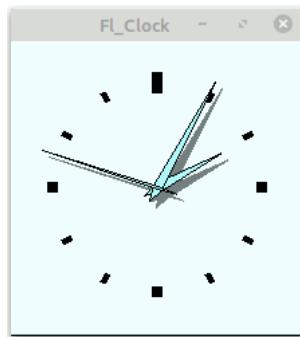


Figure 31.9 FL_SQUARE_CLOCK type



Figure 31.10 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
```

31.18.2 Constructor & Destructor Documentation

31.18.2.1 [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 boxtyle is `FL_UP_BOX`.

Parameters

in	X,Y,W,H	position and size of the widget
in	L	widget label, default is no label

31.18.3 Member Function Documentation

31.18.3.1 draw()

```
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
----	---------	-------------------

31.18.3.2 hour()

```
int Fl_Clock_Output::hour ( ) const [inline]
```

Returns the displayed hour (0 to 23).

See also

[value\(\)](#), [minute\(\)](#), [second\(\)](#)

31.18.3.3 minute()

```
int Fl_Clock_Output::minute ( ) const [inline]
```

Returns the displayed minute (0 to 59).

See also

[value\(\)](#), [hour\(\)](#), [second\(\)](#)

31.18.3.4 second()

```
int Fl_Clock_Output::second ( ) const [inline]
```

Returns the displayed second (0 to 60, 60=leap second).

See also

[value\(\)](#), [hour\(\)](#), [minute\(\)](#)

31.18.3.5 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)

31.18.3.6 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
----	------	---------------------------------------

31.18.3.7 value() [1/3]

```
void Fl_Clock_Output::value (
```

<code>ulong</code>	<code>v</code>
--------------------	----------------

```
)
```

Set the displayed time.

Set the time in seconds since the UNIX epoch (January 1, 1970).

Parameters

<code>in</code>	<code>v</code>	seconds since epoch
-----------------	----------------	---------------------

See also

[value\(\)](#)

31.18.3.8 value() [2/3]

```
void Fl_Clock_Output::value (
```

<code>int</code>	<code>H</code> ,
<code>int</code>	<code>m</code> ,
<code>int</code>	<code>s</code>

```
)
```

Set the displayed time.

Set the time in hours, minutes, and seconds.

Parameters

<code>in</code>	<code>H,m,s</code>	displayed time
-----------------	--------------------	----------------

See also

[hour\(\)](#), [minute\(\)](#), [second\(\)](#)

31.18.3.9 value() [3/3]

```
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\(ulong\)](#)

The documentation for this class was generated from the following files:

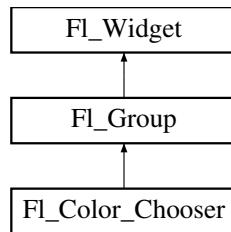
- Fl_Clock.H
- Fl_Clock.cxx

31.19 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)
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** ()
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.

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.

Related Functions

(Note that these are not member functions.)

- 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

31.19.1 Detailed Description

The [Fl_Color_Chooser](#) widget provides a standard RGB color chooser.

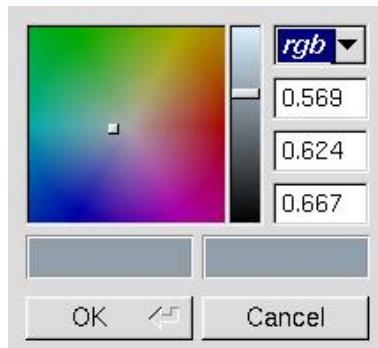


Figure 31.11 fl_color_chooser()

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.

31.19.2 Constructor & Destructor Documentation

31.19.2.1 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

31.19.3 Member Function Documentation

31.19.3.1 b()

```
double Fl_Color_Chooser::b ( ) const [inline]
```

Returns the current blue value.

$0 \leq b \leq 1.$

31.19.3.2 g()

```
double Fl_Color_Chooser::g ( ) const [inline]
```

Returns the current green value.

$0 \leq g \leq 1.$

31.19.3.3 handle()

```
int Fl_Color_Chooser::handle (
    int e ) [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](#).

31.19.3.4 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.

31.19.3.5 `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

31.19.3.6 `hue()`

```
double Fl_Color_Chooser::hue ( ) const [inline]
```

Returns the current hue.

$0 \leq \text{hue} < 6$. 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.*

31.19.3.7 `mode()` [1/2]

```
int Fl_Color_Chooser::mode ( ) [inline]
```

Returns which [Fl_Color_Chooser](#) variant is currently active.

Returns

color modes are `rgb(0)`, `byte(1)`, `hex(2)`, or `hsv(3)`

31.19.3.8 mode() [2/2]

```
void Fl_Color_Chooser::mode (
    int newMode )
```

Set which [Fl_Color_Chooser](#) variant is currently active.

Parameters

in	<i>newMode</i>	color modes are rgb(0), byte(1), hex(2), or hsv(3)
----	----------------	--

31.19.3.9 r()

```
double Fl_Color_Chooser::r () const [inline]
```

Returns the current red value.

$0 \leq r \leq 1$.

31.19.3.10 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	<i>R,G,B</i>	color components.
----	--------------	-------------------

Returns

1 if a new rgb value was set, 0 if the rgb value was the previous one.

31.19.3.11 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	<i>R,G,B</i>	color components
out	<i>H,S,V</i>	color components

31.19.3.12 saturation()

```
double Fl_Color_Chooser::saturation () const [inline]
```

Returns the saturation.

$0 \leq \text{saturation} \leq 1$.

31.19.3.13 value()

```
double Fl_Color_Chooser::value () const [inline]
```

Returns the value/brightness.

$0 \leq \text{value} \leq 1$.

The documentation for this class was generated from the following files:

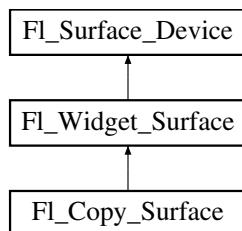
- [Fl_Color_Chooser.H](#)
- [Fl_Color_Chooser.cxx](#)

31.20 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

- virtual 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.
- [Fl_Copy_Surface \(int w, int h\)](#)
the constructor
- int [h \(\)](#)
Returns the pixel height of the copy surface.
- virtual bool [is_current \(\)](#)
Is this surface the current drawing surface?
- void [origin \(int *x, int *y\)](#)
Computes the coordinates of the current origin of graphics functions.
- void [origin \(int x, int y\)](#)
Sets the position of the origin of graphics in the drawable part of the drawing surface.
- int [printable_rect \(int *w, int *h\)](#)
Computes the width and height of the drawable area of the drawing surface.
- void [set_current \(\)](#)
Make this surface the current drawing surface.
- int [w \(\)](#)
Returns the pixel width of the copy surface.
- [~Fl_Copy_Surface \(\)](#)
the destructor

Protected Member Functions

- void [translate \(int x, int y\)](#)
Translates the current graphics origin accounting for the current rotation.
- void [untranslate \(\)](#)
Undoes the effect of a previous [translate\(\)](#) call.

Additional Inherited Members

31.20.1 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: the graphical data are copied to the clipboard as an image in BMP format.

31.20.2 Constructor & Destructor Documentation

31.20.2.1 Fl_Copy_Surface()

```
Fl_Copy_Surface::Fl_Copy_Surface (
    int w,
    int h )
```

the constructor

Parameters

<code>w,h</code>	Width and height of the drawing surface in FLTK units
------------------	---

31.20.3 Member Function Documentation

31.20.3.1 draw_decorated_window()

```
void Fl_Copy_Surface::draw_decorated_window (
    Fl_Window * win,
    int win_offset_x = 0,
    int win_offset_y = 0 ) [virtual]
```

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.

Reimplemented from [Fl_Widget_Surface](#).

31.20.3.2 origin() [1/2]

```
void Fl_Copy_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 from [Fl_Widget_Surface](#).

31.20.3.3 origin() [2/2]

```
void Fl_Copy_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 from [Fl_Widget_Surface](#).

31.20.3.4 printable_rect()

```
int Fl_Copy_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 from [Fl_Widget_Surface](#).

31.20.3.5 set_current()

```
void Fl_Copy_Surface::set_current (
    void ) [virtual]
```

Make this surface the current drawing surface.

This surface will receive all future graphics requests. Starting from FLTK 1.4.0, another convenient API to set/unset the current drawing surface is [Fl_Surface_Device::push_current\(\)](#) / [Fl_Surface_Device::pop_current\(\)](#).

Reimplemented from [Fl_Surface_Device](#).

31.20.3.6 translate()

```
void Fl_Copy_Surface::translate (
    int x,
    int y ) [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](#).

The documentation for this class was generated from the following files:

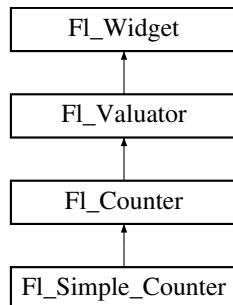
- [Fl_Copy_Surface.H](#)
- [Fl_Copy_Surface.cxx](#)

31.21 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)
Handles the specified event.
- void `lstep` (double a)
Sets the increment for the large step buttons.
- void `step` (double a, double b)
Sets the increments for the normal and large step buttons.
- void `step` (double a)
Sets the increment for the normal step buttons.
- double `step` () const
Returns the increment for normal 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.

Protected Member Functions

- void `draw` ()
Draws the widget.

Additional Inherited Members

31.21.1 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.

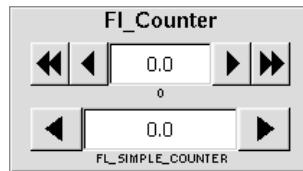


Figure 31.12 Fl_Counter

Todo Refactor the doxygen comments for [Fl_Counter type\(\)](#) documentation.

The type of an [Fl_Counter](#) object can be set using `type(uchar t)` to:

- `FL_NORMAL_COUNTER`: Displays a counter with 4 arrow buttons.
- `FL_SIMPLE_COUNTER`: Displays a counter with only 2 arrow buttons.

31.21.2 Constructor & Destructor Documentation

31.21.2.1 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

31.21.3 Member Function Documentation

31.21.3.1 draw()

```
void Fl_Counter::draw ( )  [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](#).

31.21.3.2 handle()

```
int Fl_Counter::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.

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](#).

31.21.3.3 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.
----	---	-----------------------

31.21.3.4 step() [1/2]

```
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.
----	-----	-----------------------------------

31.21.3.5 step() [2/2]

```
void Fl_Counter::step (
    double a ) [inline]
```

Sets the increment for the normal step buttons.

Parameters

in	a	normal step increment.
----	---	------------------------

The documentation for this class was generated from the following files:

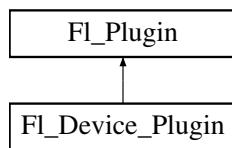
- Fl_Counter.H
- Fl_Counter.cxx

31.22 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

- [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.

Static Public Member Functions

- [static Fl_Device_Plugin * opengl_plugin \(\)](#)
Returns the OpenGL plugin.

31.22.1 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`.

31.22.2 Member Function Documentation

31.22.2.1 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:

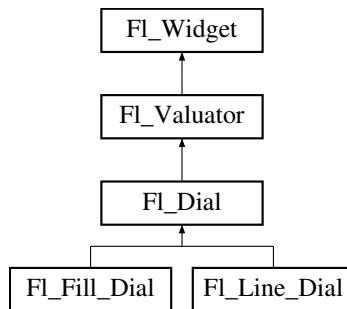
- [Fl_Device.H](#)
- [Fl_Device.cxx](#)

31.23 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 \(int\)](#)
Allow subclasses to handle event based on current position and size.

Protected Member Functions

- void [draw](#) (int X, int Y, int W, int H)
Draws dial at given position and size.
- void [draw](#) ()
Draws dial at current 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.

Additional Inherited Members

31.23.1 Detailed Description

The [Fl_Dial](#) widget provides a circular dial to control a single floating point value.

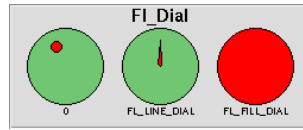


Figure 31.13 Fl_Dial

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.

31.23.2 Constructor & Destructor Documentation

31.23.2.1 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](#).

31.23.3 Member Function Documentation

31.23.3.1 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.

31.23.3.2 draw()

```
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
----	---------	-------------------

31.23.3.3 handle()

```
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.
----	---------------	---

The documentation for this class was generated from the following files:

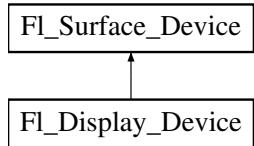
- Fl_Dial.H
- Fl_Dial.cxx

31.24 Fl_Display_Device Class Reference

A display to which the computer can draw.

```
#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.

Additional Inherited Members

31.24.1 Detailed Description

A display to which the computer can draw.

When the program begins running, an object of class [Fl_Display_Device](#) has been created and made the current drawing surface.

The documentation for this class was generated from the following files:

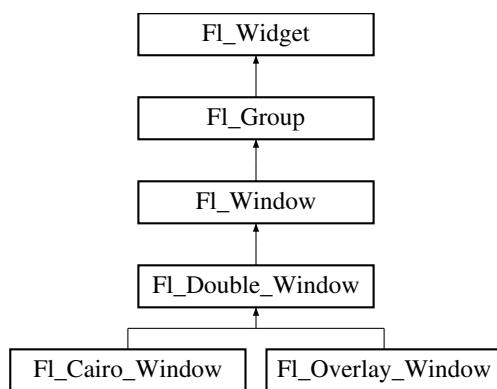
- [Fl_Device.H](#)
- [Fl_Device.cxx](#)

31.25 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

- virtual `Fl_Double_Window * as_double_window ()`
Return non-null if this is an `Fl_Overlay_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 ()`
Forces the window to be drawn, this window is also made current and calls `draw()`.
- void `hide ()`
Removes the window from the screen.
- void `resize (int, int, int, int)`
Changes the size and position of the window.
- void `show ()`
Puts the window on the screen.
- void `show (int a, char **b)`
- `~Fl_Double_Window ()`
The destructor also deletes all the children.

Additional Inherited Members

31.25.1 Detailed Description

The `Fl_Double_Window` provides a double-buffered window.

If possible this will use the X double buffering extension (Xdbe). If not, it will draw the window data into an off-screen pixmap, and then copy it to the on-screen window.

It is highly recommended that you put the following code before the first `show()` of *any* window in your program:

```
Fl::visual (FL_DOUBLE | FL_INDEX)
```

This makes sure you can use Xdbe on servers where double buffering does not exist for every visual.

31.25.2 Constructor & Destructor Documentation

31.25.2.1 `~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.

31.25.3 Member Function Documentation

31.25.3.1 flush()

```
void Fl_Double_Window::flush ( ) [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](#).

31.25.3.2 hide()

```
void Fl_Double_Window::hide ( ) [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_Window](#).

Reimplemented in [Fl_Overlay_Window](#).

31.25.3.3 resize()

```
void Fl_Double_Window::resize (
    int X,
    int Y,
    int W,
    int H ) [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 `size(x,y)` and `position(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_Window](#).

Reimplemented in [Fl_Overlay_Window](#).

31.25.3.4 show()

```
void Fl_Double_Window::show ( ) [virtual]
```

Puts the window on the screen.

Usually (on X) this has the side effect of opening the display.

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_Window](#).

Reimplemented in [Fl_Overlay_Window](#).

The documentation for this class was generated from the following files:

- [Fl_Double_Window.H](#)
- [Fl_Double_Window.cxx](#)

31.26 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\(\)](#)*

31.26.1 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:

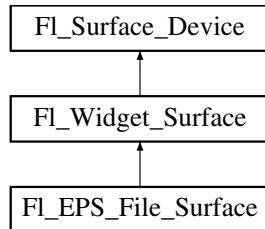
- [Fl_Group.H](#)

31.27 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 \(\)](#)
Closes using fclose() the underlying FILE pointer.
- FILE * [file \(\)](#)
Returns the underlying FILE pointer.
- [Fl_EPS_File_Surface](#) (int width, int height, FILE *eps, [Fl_Color](#) background=FL_WHITE)
Constructor.
- virtual void [origin](#) (int x, int y)
Sets the position of the origin of graphics in the drawable part of the drawing surface.
- virtual void [origin](#) (int *px, int *py)
Computes the coordinates of the current origin of graphics functions.
- 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.
- [~Fl_EPS_File_Surface \(\)](#)
Destructor.

Protected Member Functions

- `F1_PostScript_Graphics_Driver * driver ()`
Returns the PostScript driver of this drawing surface.

Additional Inherited Members

31.27.1 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:

```
F1_Window *win = ...// Window to draw to an .eps file
int ww = win->decorated_w();
int wh = win->decorated_h();
FILE *eps = f1_fopen("/path/to/mywindow.eps", "w");
if (eps) {
    F1_EPS_File_Surface *surface = new
        F1_EPS_File_Surface(ww, wh, eps, win->color());
    F1_Surface_Device::push_current(surface);
    surface->draw_decorated_window(win);
    F1_Surface_Device::pop_current();
    delete surface; // the .eps file is not complete until the destructor was run
    fclose(eps);
}
```

31.27.2 Constructor & Destructor Documentation

31.27.2.1 F1_EPS_File_Surface()

```
F1_EPS_File_Surface::F1_EPS_File_Surface (
    int width,
    int height,
    FILE * eps,
    F1_Color background = FL_WHITE )
```

Constructor.

Parameters

<code>width,height</code>	Width and height of the EPS drawing area
<code>eps</code>	A writable FILE pointer where the Encapsulated PostScript data will be sent
<code>background</code>	Color expected to cover the background of the EPS drawing area. This parameter affects only the drawing of transparent <code>F1_RGB_Image</code> objects: transparent areas of RGB images are blended with the background color.

31.27.2.2 ~Fl_EPS_File_Surface()

```
Fl_EPS_File_Surface::~Fl_EPS_File_Surface ( )
```

Destructor.

The underlying FILE pointer remains open after destruction of the [Fl_EPS_File_Surface](#) object unless [close\(\)](#) was called.

31.27.3 Member Function Documentation

31.27.3.1 close()

```
int Fl_EPS_File_Surface::close ( )
```

Closes using `fclose()` the underlying FILE pointer.

The only operation possible with the [Fl_EPS_File_Surface](#) object after calling [close\(\)](#) is its destruction.

31.27.3.2 driver()

```
Fl_PostScript_Graphics_Driver* Fl_EPS_File_Surface::driver ( ) [inline], [protected]
```

Returns the PostScript driver of this drawing surface.

31.27.3.3 origin() [1/2]

```
virtual void Fl_EPS_File_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 from [Fl_Widget_Surface](#).

31.27.3.4 origin() [2/2]

```
virtual void Fl_EPS_File_Surface::origin (
    int * x,
    int * y )  [virtual]
```

Computes the coordinates of the current origin of graphics functions.

Parameters

<code>out</code>	<code>x,y</code>	If non-null, <code>*x</code> and <code>*y</code> are set to the horizontal and vertical coordinates of the graphics origin.
------------------	------------------	---

Reimplemented from [Fl_Widget_Surface](#).

31.27.3.5 printable_rect()

```
virtual int Fl_EPS_File_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 from [Fl_Widget_Surface](#).

31.27.3.6 translate()

```
virtual void Fl_EPS_File_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 from [Fl_Widget_Surface](#).

The documentation for this class was generated from the following file:

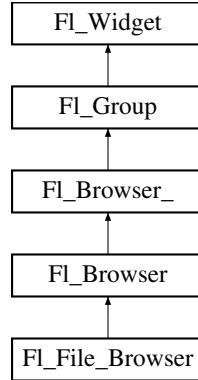
- [Fl_PostScript.H](#)

31.28 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 Member Functions

- void **errmsg** (const char *emsg)

Sets OS error message to a string, which can be NULL.
- const char * **errmsg** () const

Returns OS error messages, or NULL if none.
- 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.
- void **filter** (const char *pattern)

Sets or gets the filename filter.
- const char * **filter** () const

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)

Additional Inherited Members

31.28.1 Detailed Description

The [Fl_File_Browser](#) widget displays a list of filenames, optionally with file-specific icons.

31.28.2 Constructor & Destructor Documentation

31.28.2.1 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.

31.28.3 Member Function Documentation

31.28.3.1 errmsg() [1/2]

```
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\(const char *emsg\)](#);

31.28.3.2 errmsg() [2/2]

```
const char* Fl_File_Browser::errmsg ( ) const [inline]
```

Returns OS error messages, or NULL if none.

Use when advised.

31.28.3.3 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.

31.28.3.4 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.

31.28.3.5 filter() [1/2]

```
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.

31.28.3.6 filter() [2/2]

```
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.

31.28.3.7 iconsize() [1/2]

```
uchar Fl_File_Browser::iconsize ( ) const [inline]
```

Sets or gets the size of the icons.

The default size is 20 pixels.

31.28.3.8 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.

31.28.3.9 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

31.29 Fl_File_Chooser Class Reference

The [Fl_File_Chooser](#) widget displays a standard file selection dialog that supports various selection modes.

Public Types

- enum { **SINGLE** = 0, **MULTI** = 1, **CREATE** = 2, **DIRECTORY** = 4 }

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.
- [void color \(Fl_Color c\)](#)
Sets the background color of the [Fl_File_Browser](#) list.
- [Fl_Color color \(\)](#)
Gets the background color of the [Fl_File_Browser](#) list.
- [int count \(\)](#)
Returns the number of selected files.
- [void directory \(const char *d\)](#)
Sets the current directory.
- [char * directory \(\)](#)
Gets the current directory.
- [void filter \(const char *p\)](#)
Sets the current filename filter patterns.
- [const char * filter \(\)](#)

- `int filter_value ()`
 Gets the current filename filter patterns.
- `void filter_value (int f)`
 Gets the current filename filter selection.
- `Fl_File_Chooser (const char *d, const char *p, int t, const char *title)`
 The constructor creates the `Fl_File_Chooser` dialog shown.
- `void hide ()`
 Hides the `Fl_File_Chooser` window.
- `void iconsize (uchar s)`
 Sets the size of the icons in the `Fl_File_Browser`.
- `uchar iconsize ()`
 Gets the size of the icons in the `Fl_File_Browser`.
- `void label (const char *)`
 Sets the title bar text for the `Fl_File_Chooser`.
- `const char * label ()`
 Gets the title bar text for the `Fl_File_Chooser`.
- `void ok_label (const char *)`
 Sets the label for the "ok" button in the `Fl_File_Chooser`.
- `const char * ok_label ()`
 Gets the label for the "ok" button in the `Fl_File_Chooser`.
- `void preview (int e)`
 Enable or disable the preview tile.
- `int preview () const`
 Returns the current state of the preview box.
- `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 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 textcolor (Fl_Color c)`
 Sets the current `Fl_File_Browser` text color.
- `Fl_Color textcolor ()`
 Gets the current `Fl_File_Browser` text color.
- `void textfont (Fl_Font f)`
 Sets the current `Fl_File_Browser` text font.
- `Fl_Font textfont ()`
 Gets the current `Fl_File_Browser` text font.
- `void textszie (Fl_Fontsize s)`
 Sets the current `Fl_File_Browser` text size.
- `Fl_Fontsize textszie ()`
 Gets the current `Fl_File_Browser` text size.
- `void type (int t)`
 Sets the current type of `Fl_File_Chooser`.
- `int type ()`
 Gets 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.
- const char * `value` (int f=1)
Gets the current value of the selected file(s).
- void `value` (const char *filename)
Sets the current value of the selected file.
- int `visible` ()
Returns 1 if the `Fl_File_Chooser` window is visible.
- `~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 * `addFavorites_label` = "Add to Favorites"
[standard text may be customized at run-time]
- static const char * `allFiles_label` = "All Files (*)"
[standard text may be customized at run-time]
- static const char * `customFilter_label` = "Custom Filter"
[standard text may be customized at run-time]
- static const char * `existingFile_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 * `manageFavorites_label` = "Manage Favorites"
[standard text may be customized at run-time]
- static const char * `newDirectory_label` = "New Directory?"
[standard text may be customized at run-time]
- static const char * `newDirectory_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 Functions

(Note that these are not member functions.)

- 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 *)
Set the "OK" button label.

31.29.1 Detailed Description

The [FI_File_Chooser](#) widget displays a standard file selection dialog that supports various selection modes.

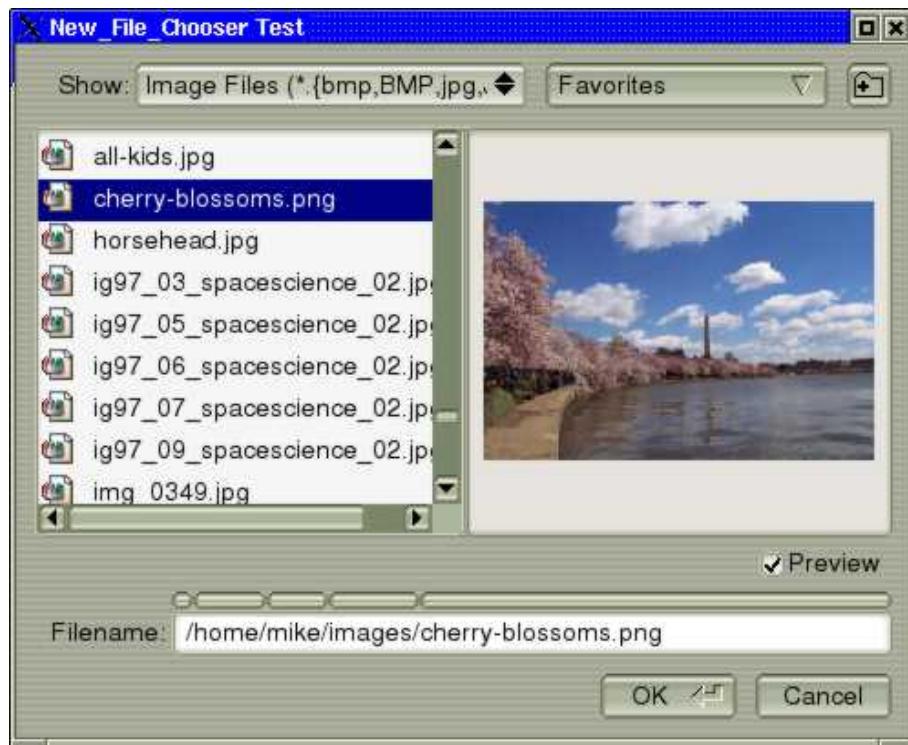


Figure 31.14 FI_File_Chooser

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 [FI_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 [FI_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
addFavorites_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:"
manageFavorites_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.

31.29.2 Constructor & Destructor Documentation

31.29.2.1 `Fl_File_Chooser()`

```
Fl_File_Chooser::Fl_File_Chooser (
    const char * pathname,
    const char * pattern,
    int type,
    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 argument can be one of the following:

- SINGLE - allows the user to select a single, existing file.
- MULTI - allows the user to select one or more existing files.
- CREATE - allows the user to select a single, existing file or specify a new filename.
- DIRECTORY - allows the user to select a single, existing directory.

The title argument is used to set the title bar text for the `Fl_File_Chooser` window.

31.29.2.2 `~Fl_File_Chooser()`

```
Fl_File_Chooser::~Fl_File_Chooser ()
```

Destroys the widget and frees all memory used by it.

31.29.3 Member Function Documentation

31.29.3.1 add_extra()

```
F1_Widget * Fl_File_Chooser::add_extra (
    F1_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	<code>extra</code>	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.

31.29.3.2 color() [1/2]

```
void Fl_File_Chooser::color (
    F1_Color c )
```

Sets the background color of the [Fl_File_Browser](#) list.

31.29.3.3 color() [2/2]

```
Fl_Color Fl_File_Chooser::color ( )
```

Gets the background color of the [Fl_File_Browser](#) list.

31.29.3.4 count()

```
int Fl_File_Chooser::count ( )
```

Returns the number of selected files.

31.29.3.5 directory() [1/2]

```
void Fl_File_Chooser::directory (
    const char * d )
```

Sets the current directory.

31.29.3.6 directory() [2/2]

```
const char * Fl_File_Chooser::directory ( )
```

Gets the current directory.

31.29.3.7 filter() [1/2]

```
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.

31.29.3.8 filter() [2/2]

```
const char * Fl_File_Chooser::filter ( )
```

Gets the current filename filter patterns.

31.29.3.9 filter_value() [1/2]

```
int Fl_File_Chooser::filter_value ( )
```

Gets the current filename filter selection.

31.29.3.10 filter_value() [2/2]

```
void Fl_File_Chooser::filter_value ( int f )
```

Sets the current filename filter selection.

31.29.3.11 hide()

```
void Fl_File_Chooser::hide ( )
```

Hides the [Fl_File_Chooser](#) window.

31.29.3.12 iconsize() [1/2]

```
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\(\)](#).

31.29.3.13 iconsize() [2/2]

```
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\(\)](#).

31.29.3.14 label() [1/2]

```
void Fl_File_Chooser::label (
    const char * l )
```

Sets the title bar text for the [Fl_File_Chooser](#).

31.29.3.15 label() [2/2]

```
const char * Fl_File_Chooser::label ( )
```

Gets the title bar text for the [Fl_File_Chooser](#).

31.29.3.16 preview() [1/2]

```
void Fl_File_Chooser::preview (
    int e )
```

Enable or disable the preview tile.

1 = enable preview, 0 = disable preview.

31.29.3.17 preview() [2/2]

```
int Fl_File_Chooser::preview ( ) const [inline]
```

Returns the current state of the preview box.

31.29.3.18 rescan()

```
void Fl_File_Chooser::rescan ( )
```

Reloads the current directory in the [Fl_File_Browser](#).

31.29.3.19 show()

```
void Fl_File_Chooser::show ( )
```

Shows the [Fl_File_Chooser](#) window.

31.29.3.20 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\(\)](#)

31.29.3.21 textcolor() [1/2]

```
void Fl_File_Chooser::textcolor (
    Fl_Color c )
```

Sets the current [Fl_File_Browser](#) text color.

31.29.3.22 textcolor() [2/2]

```
Fl_Color Fl_File_Chooser::textcolor ( )
```

Gets the current [Fl_File_Browser](#) text color.

31.29.3.23 textfont() [1/2]

```
void Fl_File_Chooser::textfont (
    Fl_Font f )
```

Sets the current [Fl_File_Browser](#) text font.

31.29.3.24 textfont() [2/2]

```
Fl_Font Fl_File_Chooser::textfont ( )
```

Gets the current [Fl_File_Browser](#) text font.

31.29.3.25 textsize() [1/2]

```
void Fl_File_Chooser::textsize (
    Fl_Fontsize s )
```

Sets the current [Fl_File_Browser](#) text size.

31.29.3.26 textsize() [2/2]

```
Fl_Fontsize Fl_File_Chooser::textsize ( )
```

Gets the current [Fl_File_Browser](#) text size.

31.29.3.27 type() [1/2]

```
void Fl_File_Chooser::type (
    int t )
```

Sets the current type of [Fl_File_Chooser](#).

31.29.3.28 type() [2/2]

```
int Fl_File_Chooser::type ( )
```

Gets the current type of [Fl_File_Chooser](#).

31.29.3.29 user_data()

```
void * Fl_File_Chooser::user_data ( ) const
```

Gets the file chooser user data.

31.29.3.30 value()

```
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);
...
}
```

31.29.3.31 visible()

```
int Fl_File_Chooser::visible ( )
```

Returns 1 if the [Fl_File_Chooser](#) window is visible.

31.29.4 Member Data Documentation

31.29.4.1 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`

31.30 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** (**FI_Color** c)
Adds a color 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.
- short * **add_vertex** (float x, float 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, **FI_Color** ic, int active=1)
Draws an icon in the indicated area.
- **FI_File_Icon** (const char *p, int t, int nd=0, short *d=0)
*Creates a new **FI_File_Icon** with the specified information.*
- void **label** (**FI_Widget** *w)
*Applies the icon to the widget, registering the **FI_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.
- **FI_File_Icon** * **next** ()
Returns next file icon object.
- const char * **pattern** ()
Returns the filename matching pattern for the icon.
- int **size** ()
Returns the number of words of data used by the icon.
- int **type** ()
Returns the filetype associated with the icon, which can be one of the following:
- short * **value** ()
Returns the data array for the icon.
- **~FI_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.

31.30.1 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.

31.30.2 Constructor & Destructor Documentation

31.30.2.1 `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	<code>p</code>	filename pattern
in	<code>t</code>	file type
in	<code>nd</code>	number of data values
in	<code>d</code>	data values

31.30.3 Member Function Documentation

31.30.3.1 `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
----	---	------------

31.30.3.2 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
----	---	-------------

31.30.3.3 add_vertex() [1/2]

```
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
----	-----	--------------------

31.30.3.4 add_vertex() [2/2]

```
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
----	-----	--------------------

31.30.3.5 clear()

```
void Fl_File_Icon::clear ( ) [inline]
```

Clears all icon data from the icon.

31.30.3.6 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	<i>x,y,w,h</i>	position and size
in	<i>ic</i>	icon color
in	<i>active</i>	status, default is active [non-zero]

31.30.3.7 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	<i>filename</i>	name of file
in	<i>filetype</i>	enumerated file type

Returns

matching file icon or NULL

31.30.3.8 first()

```
static Fl_File_Icon* Fl_File_Icon::first ( ) [inline], [static]
```

Returns a pointer to the first icon in the list.

31.30.3.9 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
----	---	--

31.30.3.10 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	<i>o</i>	label data
in	<i>x,y,w,h</i>	position and size of label
in	<i>a</i>	label alignment [not used]

31.30.3.11 load()

```
void Fl_File_Icon::load (
    const char * f )
```

Loads the specified icon image.

The format is deduced from the filename.

Parameters

in	<i>f</i>	filename
----	----------	----------

31.30.3.12 load_fti()

```
int Fl_File_Icon::load_fti (
    const char * fti )
```

Loads an SGI icon file.

Parameters

in	<i>fti</i>	icon filename
----	------------	---------------

Returns

0 on success, non-zero on error

31.30.3.13 load_image()

```
int Fl_File_Icon::load_image (
    const char * ifile )
```

Load an image icon file from an image filename.

Parameters

in	<i>ifile</i>	image filename
----	--------------	----------------

Returns

0 on success, non-zero on error

31.30.3.14 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();
```

31.30.3.15 next()

```
Fl_File_Icon* Fl_File_Icon::next ( ) [inline]
```

Returns next file icon object.

See [Fl_File_Icon::first\(\)](#)

31.30.3.16 pattern()

```
const char* Fl_File_Icon::pattern ( ) [inline]
```

Returns the filename matching pattern for the icon.

31.30.3.17 size()

```
int Fl_File_Icon::size ( ) [inline]
```

Returns the number of words of data used by the icon.

31.30.3.18 type()

```
int Fl_File_Icon::type ( ) [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.

31.30.3.19 value()

```
short* Fl_File_Icon::value ( ) [inline]
```

Returns the data array for the icon.

The documentation for this class was generated from the following files:

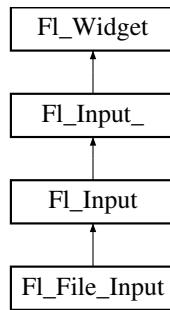
- `Fl_File_Icon.H`
- `Fl_File_Icon.cxx`
- `Fl_File_Icon2.cxx`

31.31 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_Color errorcolor \(\) const](#)
Gets the current error color.
- [void errorcolor \(Fl_Color c\)](#)
Sets the current error color to c.
- [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.
- [virtual int handle \(int event\)](#)
Handle events in the widget.
- [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.
- [const char * value \(\)](#)
Returns the current value, which is a pointer to an internal buffer and is valid only until the next event is handled.

Protected Member Functions

- [virtual void draw \(\)](#)
Draws the file input widget.

Additional Inherited Members

31.31.1 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.

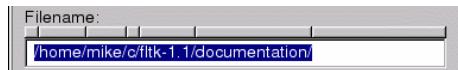


Figure 31.15 Fl_File_Input

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.

31.31.2 Constructor & Destructor Documentation

31.31.2.1 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 boxtyle is `FL_DOWN_BOX`.

Parameters

in	X,Y,W,H	position and size of the widget
in	L	widget label, default is no label

31.31.3 Member Function Documentation

31.31.3.1 down_box() [1/2]

```
F1_Boxtype F1_File_Input::down_box ( ) const [inline]
```

Gets the box type used for the navigation bar.

31.31.3.2 down_box() [2/2]

```
void F1_File_Input::down_box ( F1_Boxtype b ) [inline]
```

Sets the box type to use for the navigation bar.

31.31.3.3 errorcolor() [1/2]

```
F1_Color F1_File_Input::errorcolor ( ) const [inline]
```

Gets the current error color.

Returns FL_RED since FLTK 1.4.0 (default in 1.3.x). Retained for backwards compatibility.

Deprecated Will be removed in FLTK 1.5.0 or higher.

Todo Remove [F1_File_Input::errorcolor\(\)](#) in FLTK 1.5.0 or higher.

31.31.3.4 errorcolor() [2/2]

```
void F1_File_Input::errorcolor ( F1_Color c ) [inline]
```

Sets the current error color to c.

Does nothing since FLTK 1.4.0. Retained for backwards compatibility.

Deprecated Will be removed in FLTK 1.5.0 or higher.

Todo Remove [F1_File_Input::errorcolor\(F1_Color\)](#) in FLTK 1.5.0 or higher.

31.31.3.5 handle()

```
int F1_File_Input::handle ( int event ) [virtual]
```

Handle events in the widget.

Return non zero if event is handled.

Parameters

in	<i>event</i>	
----	--------------	--

Reimplemented from [Fl_Widget](#).

31.31.3.6 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	<i>str</i>	new string value
----	------------	------------------

31.31.3.7 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	<i>str</i>	new string value
in	<i>len</i>	length of value

The documentation for this class was generated from the following files:

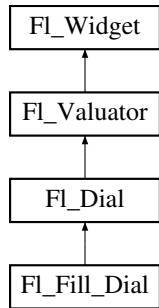
- [Fl_File_Input.H](#)
- [Fl_File_Input.cxx](#)

31.32 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)
Creates a filled dial, also setting its type to FL_FILL_DIAL.

Additional Inherited Members

31.32.1 Detailed Description

Draws a dial with a filled arc.

31.32.2 Constructor & Destructor Documentation

31.32.2.1 Fl_Fill_Dial()

```
Fl_Fill_Dial::Fl_Fill_Dial (
    int X,
    int Y,
    int W,
    int H,
    const char * L )
```

Creates a filled dial, also setting its type to FL_FILL_DIAL.

The documentation for this class was generated from the following files:

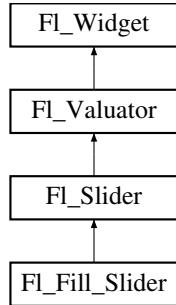
- [Fl_Fill_Dial.H](#)
- [Fl_Dial.cxx](#)

31.33 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.

Additional Inherited Members

31.33.1 Detailed Description

Widget that draws a filled horizontal slider, useful as a progress or value meter.

31.33.2 Constructor & Destructor Documentation

31.33.2.1 Fl_Fill_Slider()

```
Fl_Fill_Slider::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.

The documentation for this class was generated from the following files:

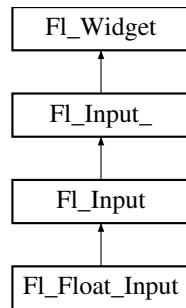
- [Fl_Fill_Slider.H](#)
- [Fl_Slider.cxx](#)

31.34 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.

Additional Inherited Members

31.34.1 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).

31.34.2 Constructor & Destructor Documentation

31.34.2.1 [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 boxtyle 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:

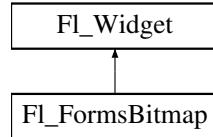
- [Fl_Float_Input.H](#)
- [Fl_Input.cxx](#)

31.35 Fl_FormsBitmap Class Reference

Forms compatibility Bitmap Image Widget.

```
#include <Fl_FormsBitmap.h>
```

Inheritance diagram for Fl_FormsBitmap:



Public Member Functions

- void [bitmap \(Fl_Bitmap *B\)](#)
Sets a new bitmap.
- [Fl_Bitmap * bitmap \(\) const](#)
Gets a the current associated Fl_Bitmap objects.
- [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.

Protected Member Functions

- void [draw \(\)](#)
Draws the bitmap and its associated box.

Additional Inherited Members

31.35.1 Detailed Description

Forms compatibility Bitmap Image Widget.

31.35.2 Member Function Documentation

31.35.2.1 bitmap() [1/2]

```
void Fl_FormsBitmap::bitmap (
    Fl_Bitmap * B ) [inline]
```

Sets a new bitmap.

31.35.2.2 bitmap() [2/2]

```
Fl_Bitmap* Fl_FormsBitmap::bitmap () const [inline]
```

Gets a the current associated [Fl_Bitmap](#) objects.

31.35.2.3 draw()

```
void Fl_FormsBitmap::draw (
    void ) [protected], [virtual]
```

Draws the bitmap and its associated box.

Implements [Fl_Widget](#).

31.35.2.4 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:

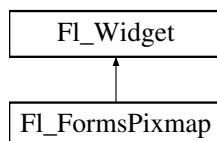
- [Fl_FormsBitmap.H](#)
- [forms_bitmap.cxx](#)

31.36 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.
- `void Pixmap (Fl_Pixmap *B)`
Set the internal pixmap pointer to an existing pixmap.
- `Fl_Pixmap * Pixmap () const`
Get the internal pixmap pointer.
- `void set (char *const *bits)`
Set/create the internal pixmap using raw data.

Protected Member Functions

- `void draw ()`
Draws the widget.

Additional Inherited Members

31.36.1 Detailed Description

Forms pixmap drawing routines.

31.36.2 Constructor & Destructor Documentation

31.36.2.1 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	<code>t</code>	box type
in	<code>X,Y,W,H</code>	position and size
in	<code>L</code>	widget label, default is no label

31.36.3 Member Function Documentation

31.36.3.1 draw()

```
void Fl_FormsPixmap::draw ( ) [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](#).

31.36.3.2 Pixmap() [1/2]

```
void Fl_FormsPixmap::Pixmap (
    Fl_Pixmap * B ) [inline]
```

Set the internal pixmap pointer to an existing pixmap.

Parameters

in	<i>B</i>	existing pixmap
----	----------	-----------------

31.36.3.3 Pixmap() [2/2]

```
Fl_Pixmap* Fl_FormsPixmap::Pixmap ( ) const [inline]
```

Get the internal pixmap pointer.

31.36.3.4 set()

```
void Fl_FormsPixmap::set (
    char *const * bits )
```

Set/create the internal pixmap using raw data.

Parameters

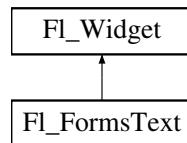
in	<i>bits</i>	raw data
----	-------------	----------

The documentation for this class was generated from the following files:

- Fl_FormsPixmap.H
- formsPixmap.cxx

31.37 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)

Protected Member Functions

- void [draw\(\)](#)
Draws the widget.

Additional Inherited Members

31.37.1 Member Function Documentation

31.37.1.1 draw()

```
void Fl_FormsText::draw ( ) [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:

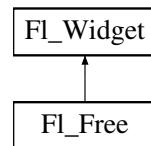
- forms.H

31.38 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 \(int e\)](#)
Handles the specified event.
- [~Fl_Free \(\)](#)
The destructor will call the handle function with the event FL_FREE_MEM.

Protected Member Functions

- void [draw \(\)](#)
Draws the widget.

Additional Inherited Members

31.38.1 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`.

31.38.2 Constructor & Destructor Documentation

31.38.2.1 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	<i>t</i>	type
in	<i>X,Y,W,H</i>	position and size
in	<i>L</i>	widget label
in	<i>hdl</i>	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
```

31.38.3 Member Function Documentation

31.38.3.1 draw()

```
void Fl_Free::draw ( ) [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](#).

31.38.3.2 handle()

```
int Fl_Free::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.

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:

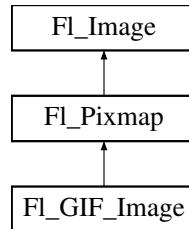
- [Fl_Free.H](#)
- [forms_free.cxx](#)

31.39 Fl_GIF_Image Class Reference

The [Fl_GIF_Image](#) class supports loading, caching, and drawing of Compuserve GIFSM images.

```
#include <Fl_GIF_Image.h>
```

Inheritance diagram for [Fl_GIF_Image](#):



Public Member Functions

- [Fl_GIF_Image](#) (const char *filename)
The constructor loads the named GIF image.
- [Fl_GIF_Image](#) (const char *imagename, const unsigned char *data)
The constructor loads a GIF image from memory.

Protected Member Functions

- void [load_gif_](#) (class [Fl_Image_Reader](#) &rdr)

Additional Inherited Members

31.39.1 Detailed Description

The [Fl_GIF_Image](#) class supports loading, caching, and drawing of Compuserve GIFSM images.

The class loads the first image and supports transparency.

31.39.2 Constructor & Destructor Documentation

31.39.2.1 [Fl_GIF_Image\(\)](#) [1/2]

```
Fl_GIF_Image::Fl_GIF_Image (
    const char * filename )
```

The constructor loads the named GIF image.

If a GIF 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	<i>filename</i>	a full path and name pointing to a valid GIF file.
----	-----------------	--

See also

[FI_GIF_Image::FI_GIF_Image\(const char *imagename, const unsigned char *data\)](#)

31.39.2.2 FI_GIF_Image() [2/2]

```
FI_GIF_Image::FI_GIF_Image (
    const char * imagename,
    const unsigned char * data )
```

The 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 is animated, [FI_GIF_Image](#) will only read and display the first frame of the animation.

Use [FI_Image::fail\(\)](#) to check if [FI_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	<i>imagename</i>	A name given to this image or NULL
in	<i>data</i>	Pointer to the start of the GIF image in memory. This code will not check for buffer overruns.

See also

[FI_GIF_Image::FI_GIF_Image\(const char *filename\)](#)
[FI_Shared_Image](#)

The documentation for this class was generated from the following files:

- [FI_GIF_Image.H](#)
- [FI_GIF_Image.cxx](#)

31.40 FI_GI_Choice Class Reference**Public Member Functions**

- [FI_GI_Choice](#) (int m, const int *alistp, [FI_GI_Choice](#) *n)

Friends

- class [Fl_Gl_Window_Driver](#)

The documentation for this class was generated from the following file:

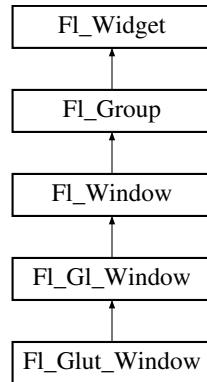
- [Fl_Gl_Choice.H](#)

31.41 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

- virtual [Fl_Gl_Window * as_gl_window \(\)](#)
Returns an [Fl_Gl_Window](#) pointer if this widget is an [Fl_Gl_Window](#).
- int [can_do \(\)](#)
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 GLContext that this window is using.
- 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 ()`
Forces the window to be drawn, this window is also made current and calls `draw()`.
- `Fl_Gl_Window_Driver * gl_driver ()`
Returns a pointer to the window's `Fl_Gl_Window_Driver` object.
- int `handle (int)`
Handle some FLTK events as needed.
- void `hide ()`
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 capabilites of the window.
- int `mode (int a)`
Set or change the OpenGL capabilites of the window.
- int `mode (const int *a)`
Set the OpenGL capabilites of the window using platform-specific data.
- 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)`
Changes the size and position of the window.
- void `show ()`
Puts the window on the screen.
- 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.
- 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.

Static Public Member Functions

- static int [can_do](#) (int m)
Returns non-zero if the hardware supports the given OpenGL mode.
- static int [can_do](#) (const int *m)
Returns non-zero if the hardware supports the given OpenGL mode.

Protected Member Functions

- virtual void [draw](#) ()
Draws the [FI_GI_Window](#).

Friends

- class [FI_GI_Window_Driver](#)

Additional Inherited Members

31.41.1 Detailed Description

The [FI_GI_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. [FI_GI_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 [FI_GI_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.

31.41.2 Constructor & Destructor Documentation

31.41.2.1 `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 boxtyle is FL_NO_BOX. The default mode is FL_RGB|FL_DOUBLE|FL_DEPTH.

31.41.2.2 `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 boxtyle is FL_NO_BOX. The default mode is FL_RGB|FL_DOUBLE|FL_DEPTH.

31.41.3 Member Function Documentation

31.41.3.1 `as_gl_window()`

```
virtual Fl_Gl_Window* Fl_Gl_Window::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

<code>NULL</code>	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](#).

31.41.3.2 can_do() [1/3]

```
static int Fl_Gl_Window::can_do (
    int m ) [inline], [static]
```

Returns non-zero if the hardware supports the given OpenGL mode.

31.41.3.3 can_do() [2/3]

```
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\(const int *a\)](#)

31.41.3.4 can_do() [3/3]

```
int Fl_Gl_Window::can_do ( ) [inline]
```

Returns non-zero if the hardware supports the current OpenGL mode.

31.41.3.5 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.

31.41.3.6 context() [1/2]

```
GLContext Fl_Gl_Window::context ( ) const [inline]
```

Returns a pointer to the GLContext that this window is using.

See also

[void context\(GLContext c, int destroy_flag\)](#)

31.41.3.7 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.

31.41.3.8 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.

31.41.3.9 draw()

```
void Fl_Gl_Window::draw (
    void ) [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();
}

```

Implements [Fl_Widget](#).

Reimplemented in [Fl_Glut_Window](#).

31.41.3.10 flush()

```
void Fl_Gl_Window::flush ( ) [virtual]
```

Forces the window to be drawn, this window is also made current and calls [draw\(\)](#).

Reimplemented from [Fl_Widget](#).

31.41.3.11 hide_overlay()

```
void Fl_Gl_Window::hide_overlay ( )
```

Hides the window if it is not this window, does nothing in Windows.

31.41.3.12 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.

31.41.3.13 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.

31.41.3.14 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)`.

31.41.3.15 mode() [2/3]

```
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` - depth buffer
- `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 required to access OpenGL version 3 or more under the X11 and MacOS platforms; it's optional under Windows. See more details in [Using OpenGL 3.0 \(or higher versions\)](#).

Version

the `FL_OPENGL3` flag appeared in version 1.3.4

31.41.3.16 mode() [3/3]

```
int Fl_Gl_Window::mode (
    const int * a ) [inline]
```

Set the OpenGL capabilities of the window using platform-specific data.

Parameters

<i>a</i>	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.

31.41.3.17 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.

31.41.3.18 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

31.41.3.19 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

31.41.3.20 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

31.41.3.21 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.

31.41.3.22 resize()

```
void Fl_Gl_Window::resize (
    int X,
    int Y,
    int W,
    int H )  [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 `size(x,y)` and `position(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_Window](#).

31.41.3.23 show()

```
void Fl_Gl_Window::show ( ) [virtual]
```

Puts the window on the screen.

Usually (on X) this has the side effect of opening the display.

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_Window](#).

31.41.3.24 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.

31.41.3.25 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](#)

31.42 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**

31.42.1 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

31.43 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

31.44 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

31.45 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

31.46 Fl_Glut_StrokeVertex Struct Reference

Public Attributes

- GLfloat **X**
- GLfloat **Y**

The documentation for this struct was generated from the following file:

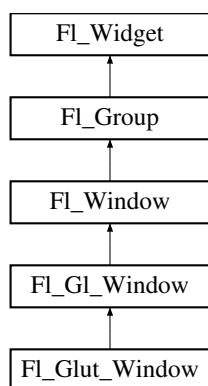
- glut.H

31.47 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

- `FI_Glut_Window (int w, int h, const char *t=0)`
Creates a glut window, registers to the glut windows list.
- `FI_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 ()`
- `~FI_Glut_Window ()`
Destroys the glut window, first unregister it from the glut windows list.

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 ()`
Draws the FI_GL_Window.
- `void draw_overlay ()`
You must implement this virtual function if you want to draw into the overlay.
- `int handle (int)`
Handle some FLTK events as needed.

Additional Inherited Members

31.47.1 Detailed Description

GLUT is emulated using this window class and these static variables (plus several more static variables hidden in `glut_compatibility.cxx`):

31.47.2 Constructor & Destructor Documentation

31.47.2.1 Fl_Glut_Window() [1/2]

```
Fl_Glut_Window::Fl_Glut_Window (
    int W,
    int H,
    const char * t = 0 )
```

Creates a glut window, registers to the glut windows list.

31.47.2.2 Fl_Glut_Window() [2/2]

```
Fl_Glut_Window::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.

31.47.3 Member Function Documentation

31.47.3.1 draw()

```
void Fl_Glut_Window::draw (
    void ) [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();
}
```

Reimplemented from [Fl_Gl_Window](#).

31.47.3.2 draw_overlay()

```
void Fl_Glut_Window::draw_overlay ( ) [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](#).

The documentation for this class was generated from the following files:

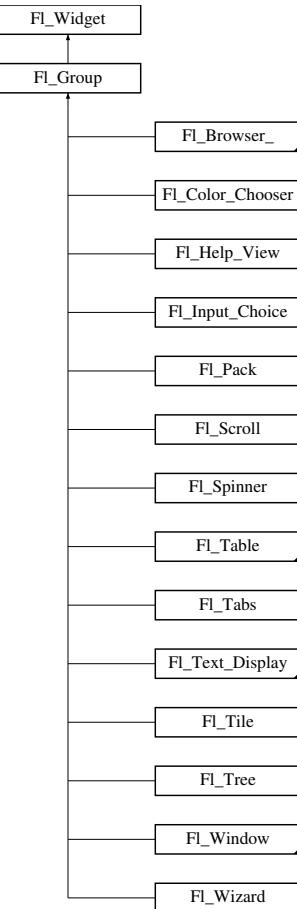
- glut.H
- glut_compatibility.cxx

31.48 Fl_Group Class Reference

The [Fl_Group](#) class is the 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 array of children.
- `virtual Fl_Group * as_group ()`
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 array()[n].
- `int children () const`
Returns how many child widgets the group has.
- `void clear ()`
Deletes all child widgets from memory recursively.
- `void clip_children (int c)`

Controls whether the group widget clips the drawing of child widgets to its bounding box.

- `unsigned int clip_children ()`

Returns the current clipping mode.
- `void end ()`

Exactly the same as current(this->parent()).
- `int find (const FL_Widget *) const`

Searches the child array for the widget and returns the index.
- `int find (const FL_Widget &o) const`

*See int FL_Group::find(const FL_Widget *w) const.*
- `FL_Group (int, int, int, int, const char *=0)`

Creates a new `FL_Group` widget using the given position, size, and label string.
- `void focus (FL_Widget *W)`
- `void forms_end ()`

This is for forms compatibility only.
- `int handle (int)`

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 (int index)`

Removes the widget at `index` from the group but does not delete it.
- `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 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.
- `FL_Widget * resizable () const`

Returns the group's resizable widget.
- `void resize (int, int, int, int)`

Resizes the `FL_Group` widget and all of its children.
- `virtual ~FL_Group ()`

The destructor also deletes all the children.

Static Public Member Functions

- `static FL_Group * current ()`

Returns the currently active group.
- `static void current (FL_Group *g)`

Sets the current group.

Protected Member Functions

- `Fl_Rect * bounds ()`
Returns the internal array of widget sizes and positions.
- `void draw ()`
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.

Additional Inherited Members

31.48.1 Detailed Description

The `Fl_Group` class is the FLTK container widget.

It maintains an array 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 or to enforce resize behavior.

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.

31.48.2 Constructor & Destructor Documentation

31.48.2.1 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 boxtyle is `FL_NO_BOX`.

31.48.2.2 ~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 child widgets' destructors would be called twice!

31.48.3 Member Function Documentation

31.48.3.1 array()

```
Fl_Widget *const * Fl_Group::array ( ) const
```

Returns a pointer to the array of children.

Note

This pointer is only valid until the next time a child is added or removed.

31.48.3.2 as_group()

```
virtual Fl_Group* Fl_Group::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

<code>NULL</code>	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](#).

31.48.3.3 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!

31.48.3.4 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

31.48.3.5 child()

```
Fl\_Widget* Fl\_Group::child (
    int n ) const [inline]
```

Returns [array\(\)](#)[n].

No range checking is done!

31.48.3.6 clear()

```
void Fl\_Group::clear ( )
```

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.

31.48.3.7 clip_children() [1/2]

```
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.

31.48.3.8 clip_children() [2/2]

```
unsigned int Fl_Group::clip_children ( ) [inline]
```

Returns the current clipping mode.

Returns

true, if clipping is enabled, false otherwise.

See also

[void Fl_Group::clip_children\(int c\)](#)

31.48.3.9 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\(0\)](#).

31.48.3.10 current() [2/2]

```
void Fl_Group::current (
    Fl_Group * g ) [static]
```

Sets the current group.

See also

[Fl_Group::current\(\)](#)

31.48.3.11 draw()

```
void Fl_Group::draw ( ) [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_Table](#), [Fl_Text_Display](#), [Fl_Tree](#), [Fl_Help_View](#), [Fl_Tabs](#), [Fl_Simple_Terminal](#), [Fl_Scroll](#), [Fl_Window](#), [Fl_Pack](#), [Fl_Wizard](#), and [Fl_Glut_Window](#).

31.48.3.12 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.

31.48.3.13 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.

31.48.3.14 draw_outside_label()

```
void Fl_Group::draw_outside_label (
    const Fl_Widget & widget ) const [protected]
```

Parents normally call this to draw outside labels of child widgets.

31.48.3.15 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.

31.48.3.16 find()

```
int Fl_Group::find (
    const Fl_Widget * o ) const
```

Searches the child array for the widget and returns the index.

Returns [children\(\)](#) if the widget is NULL or not found.

31.48.3.17 focus()

```
void Fl_Group::focus (
    Fl_Widget * W ) [inline]
```

Deprecated This is for backwards compatibility only.

You should use `W->take_focus()` instead.

See also

[Fl_Widget::take_focus\(\);](#)

31.48.3.18 handle()

```
int Fl_Group::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.

Parameters

in	<code>event</code>	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_Tree](#), [Fl_Help_View](#), [Fl_Table](#), [Fl_Tabs](#), [Fl_Window](#), [Fl_Text_Display](#), [Fl_Scroll](#), [Fl_Table_Row](#), [Fl_Spinner](#), [Fl_Text_Editor](#), [Fl_Glut_Window](#), and [Fl_Tile](#).

31.48.3.19 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)

31.48.3.20 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.

31.48.3.21 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.

31.48.3.22 remove() [1/3]

```
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

31.48.3.23 remove() [2/3]

```
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.

31.48.3.24 remove() [3/3]

```
void Fl_Group::remove (
    Fl_Widget * o ) [inline]
```

Removes the widget *o* from the group.

See also

[void remove\(Fl_Widget&\)](#)

31.48.3.25 resizable() [1/3]

```
void Fl_Group::resizable (
    Fl_Widget & o ) [inline]
```

Sets the group's resizable widget.

See [void Fl_Group::resizable\(Fl_Widget *o\)](#)

31.48.3.26 `resizable()` [2/3]

```
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.

In these examples the gray area is the resizable:

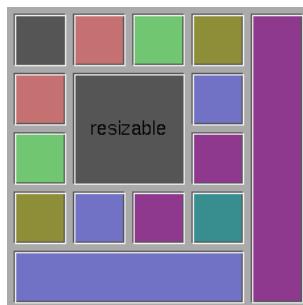


Figure 31.16 before resize

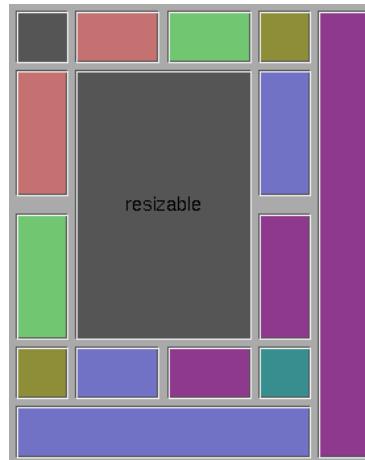


Figure 31.17 after resize

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.

31.48.3.27 resizable() [3/3]

```
Fl_Widget* Fl_Group::resizable( ) const [inline]
```

Returns the group's resizable widget.

See void [Fl_Group::resizable\(Fl_Widget *\)](#)

31.48.3.28 resize()

```
void Fl_Group::resize( int X, int Y, int W, int H ) [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_Table](#), [Fl_Text_Display](#), [Fl_Tree](#), [Fl_Help_View](#), [Fl_Window](#), [Fl_Scroll](#), [Fl_Input_Choice](#), [Fl_Spinner](#), [Fl_Overlay_Window](#), and [Fl_Tile](#).

31.48.3.29 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\(\)](#)

31.48.3.30 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](#)

31.49 Fl_Help_Block Struct Reference**Public Attributes**

- [Fl_Color bgcolor](#)
- [uchar border](#)
- [const char * end](#)
- [int h](#)
- [int line \[32\]](#)
- [const char * start](#)
- [int w](#)
- [int x](#)
- [int y](#)

The documentation for this struct was generated from the following file:

- [Fl_Help_View.H](#)

31.50 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.*
- [void fontsize \(Fl_Fontsize s\)](#)
Sets or gets the default text size for the help view.
- [Fl_Fontsize fontsize \(\)](#)
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.
- [void value \(const char *f\)](#)
The first form sets the current buffer to the string provided and reformats the text.
- [const char * value \(\) const](#)
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.

31.50.1 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`.

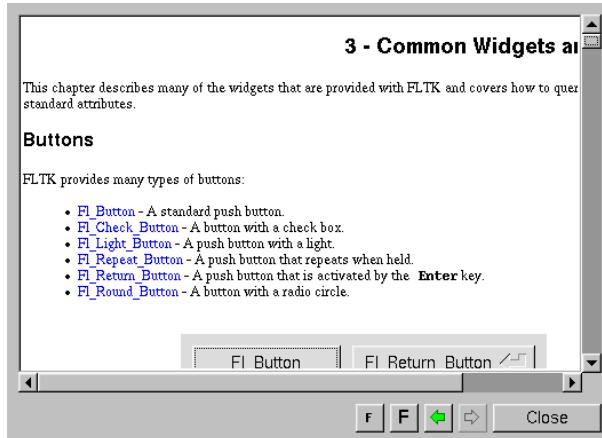


Figure 31.18 Fl_Help_Dialog

31.50.2 Constructor & Destructor Documentation

31.50.2.1 Fl_Help_Dialog()

```
Fl_Help_Dialog::Fl_Help_Dialog ( )
```

The constructor creates the dialog pictured above.

31.50.3 Member Function Documentation

31.50.3.1 h()

```
int Fl_Help_Dialog::h ( )
```

Returns the position and size of the help dialog.

31.50.3.2 hide()

```
void Fl_Help_Dialog::hide ( )
```

Hides the [Fl_Help_Dialog](#) window.

31.50.3.3 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, MSWindows-style backslashes are not supported here

Parameters

in	<i>f</i>	the name and path of an HTML file
----	----------	-----------------------------------

Returns

0 on success, -1 on error

See also

[Fl_Help_View::load\(\)](#), [fl_load_uri\(\)](#)

31.50.3.4 position()

```
void Fl_Help_Dialog::position (
    int x,
    int y )
```

Set the screen position of the dialog.

31.50.3.5 resize()

```
void Fl_Help_Dialog::resize (
    int xx,
    int yy,
    int ww,
    int hh )
```

Change the position and size of the dialog.

31.50.3.6 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.

31.50.3.7 textsize() [1/2]

```
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

31.50.3.8 textsize() [2/2]

```
uchar Fl_Help_Dialog::textsize ( )
```

Sets or gets the default text size for the help view.

31.50.3.9 value() [1/2]

```
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.

31.50.3.10 value() [2/2]

```
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.

31.50.3.11 visible()

```
int Fl_Help_Dialog::visible ( )
```

Returns 1 if the [Fl_Help_Dialog](#) window is visible.

31.50.3.12 w()

```
int Fl_Help_Dialog::w ( )
```

Returns the position and size of the help dialog.

31.50.3.13 x()

```
int Fl_Help_Dialog::x ( )
```

Returns the position and size of the help dialog.

31.50.3.14 y()

```
int Fl_Help_Dialog::y ( )
```

Returns the position and size of the help dialog.

The documentation for this class was generated from the following files:

- Fl_Help_Dialog.H
- Fl_Help_Dialog.cxx
- Fl_Help_Dialog_Dox.cxx

31.51 Fl_Help_Font_Stack Struct Reference

Public Member Functions

- `size_t count () const`
Gets the current count of font style elements in the stack.
- `Fl_Help_Font_Stack ()`
font stack construction, initialize attributes.
- `void init (Fl_Font f, Fl_Fontsize s, Fl_Color c)`
- `void pop (Fl_Font &f, Fl_Fontsize &s, Fl_Color &c)`
Pops from the stack the font style triplet and calls fl_font() & fl_color() adequately.
- `void push (Fl_Font f, Fl_Fontsize s, Fl_Color c)`
Pushes the font style triplet on the stack, also calls fl_font() & fl_color() adequately.
- `void top (Fl_Font &f, Fl_Fontsize &s, Fl_Color &c)`
Gets the top (current) element on the stack.

Protected Attributes

- `Fl_Help_Font_Style elts_ [MAX_FL_HELP_FS_ELTS]`
font elements
- `size_t nffonts_`
current number of fonts in stack

31.51.1 Constructor & Destructor Documentation

31.51.1.1 Fl_Help_Font_Stack()

```
Fl_Help_Font_Stack::Fl_Help_Font_Stack ( ) [inline]
```

font stack construction, initialize attributes.

31.51.2 Member Function Documentation

31.51.2.1 count()

```
size_t Fl_Help_Font_Stack::count ( ) const [inline]
```

Gets the current count of font style elements in the stack.

31.51.2.2 top()

```
void Fl_Help_Font_Stack::top (
    Fl_Font & f,
    Fl_Fontsize & s,
    Fl_Color & c ) [inline]
```

Gets the top (current) element on the stack.

The documentation for this struct was generated from the following file:

- Fl_Help_View.H

31.52 Fl_Help_Font_Style Struct Reference

[Fl_Help_View](#) font stack element definition.

```
#include <Fl_Help_View.H>
```

Public Member Functions

- **Fl_Help_Font_Style** ([Fl_Font](#) afont, [Fl_Fontsize](#) asize, [Fl_Color](#) acolor)
- void [get](#) ([Fl_Font](#) &afont, [Fl_Fontsize](#) &asize, [Fl_Color](#) &acolor)
Gets current font attributes.
- void [set](#) ([Fl_Font](#) afont, [Fl_Fontsize](#) asize, [Fl_Color](#) acolor)
Sets current font attributes.

Public Attributes

- **Fl_Color c**
Font Color.
- **Fl_Font f**
Font.
- **Fl_Fontsize s**
Font Size.

31.52.1 Detailed Description

`Fl_Help_View` font stack element definition.

The documentation for this struct was generated from the following file:

- `Fl_Help_View.H`

31.53 Fl_Help_Link Struct Reference

Definition of a link for the html viewer.

```
#include <Fl_Help_View.H>
```

Public Attributes

- **char filename [192]**
Reference filename.
- **int h**
Height of link text.
- **char name [32]**
Link target (blank if none)
- **int w**
Width of link text.
- **int x**
X offset of link text.
- **int y**
Y offset of link text.

31.53.1 Detailed Description

Definition of a link for the html viewer.

The documentation for this struct was generated from the following file:

- `Fl_Help_View.H`

31.54 Fl_Help_Target Struct Reference

[Fl_Help_Target](#) structure.

```
#include <Fl_Help_View.H>
```

Public Attributes

- char [name](#) [32]
Target name.
- int [y](#)
Y offset of target.

31.54.1 Detailed Description

[Fl_Help_Target](#) structure.

The documentation for this struct was generated from the following file:

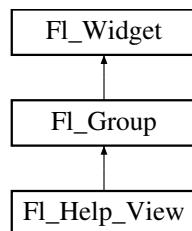
- [Fl_Help_View.H](#)

31.55 Fl_Help_View Class Reference

The [Fl_Help_View](#) widget displays HTML text.

```
#include <Fl_Help_View.H>
```

Inheritance diagram for [Fl_Help_View](#):



Public Member Functions

- void [clear_selection \(\)](#)
Removes the current text selection.
- const char * [directory \(\) const](#)
Returns the current directory for the text in the buffer.
- const char * [filename \(\) const](#)
Returns 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 \(int xx, int yy, int ww, int hh, const char *l=0\)](#)
*The constructor creates the *Fl_Help_View* widget at the specified position and size.*
- int [handle \(int\)](#)
Handles events in the widget.
- void [leftline \(int\)](#)
Scrolls the text to the indicated position, given a pixel column.
- int [leftline \(\) const](#)
Gets the left position in pixels.
- void [link \(Fl_Help_Func *fn\)](#)
*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.*
- int [load \(const char *f\)](#)
Loads the specified file.
- void [resize \(int, int, int, int\)](#)
Resizes the help widget.
- 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 [select_all \(\)](#)
Selects all the text in the view.
- int [size \(\) const](#)
Gets the size of the help view.
- void [size \(int W, int H\)](#)
- void [textcolor \(Fl_Color c\)](#)
Sets the default text color.
- [Fl_Color textcolor \(\) const](#)
Returns the current default text color.
- void [textfont \(Fl_Font f\)](#)
Sets the default text font.
- [Fl_Font textfont \(\) const](#)
Returns the current default text font.
- void [textsize \(Fl_Fontsize s\)](#)
Sets the default text size.
- [Fl_Fontsize textsize \(\) const](#)
Gets the default text size.
- const char * [title \(\)](#)
Returns the current document title, or NULL if there is no title.
- void [topline \(const char *n\)](#)
Scrolls the text to the indicated position, given a named destination.
- void [topline \(int\)](#)

- Scrolls the text to the indicated position, given a pixel line.
- int `topline () const`
Returns the current top line in pixels.
- void `value (const char *val)`
Sets the current help text buffer to the string provided and reformats the text.
- const char * `value () const`
Returns the current buffer contents.
- `~FI_Help_View ()`
Destroys the `FI_Help_View` widget.

Protected Member Functions

- void `draw ()`
Draws the `FI_Help_View` widget.

Additional Inherited Members

31.55.1 Detailed Description

The `FI_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 luml 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

31.55.2 Constructor & Destructor Documentation

31.55.2.1 ~Fl_Help_View()

```
Fl_Help_View::~Fl_Help_View ( )
```

Destroys the [Fl_Help_View](#) widget.

The destructor destroys the widget and frees all memory that has been allocated for the current document.

31.55.3 Member Function Documentation

31.55.3.1 clear_selection()

```
void Fl_Help_View::clear_selection ( )
```

Removes the current text selection.

31.55.3.2 directory()

```
const char* Fl_Help_View::directory ( ) const [inline]
```

Returns the current directory for the text in the buffer.

31.55.3.3 draw()

```
void Fl_Help_View::draw ( void ) [protected], [virtual]
```

Draws the [Fl_Help_View](#) widget.

Reimplemented from [Fl_Group](#).

31.55.3.4 filename()

```
const char* Fl_Help_View::filename ( ) const [inline]
```

Returns the current filename for the text in the buffer.

31.55.3.5 find()

```
int Fl_Help_View::find (
    const char * s,
    int p = 0 )
```

Finds the specified string *s* at starting position *p*.

Returns

the matching position or -1 if not found

31.55.3.6 handle()

```
int Fl_Help_View::handle (
    int event ) [virtual]
```

Handles events in the widget.

Reimplemented from [Fl_Group](#).

31.55.3.7 leftline() [1/2]

```
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	<i>left</i>	left column number in pixels (0 = left side)
----	-------------	--

31.55.3.8 leftline() [2/2]

```
int Fl_Help_View::leftline ( ) const [inline]
```

Gets the left position in pixels.

31.55.3.9 link()

```
void Fl_Help_View::link (
    Fl_Help_Func * fn ) [inline]
```

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 NULL:

```
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 NULL, 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 NULL.

31.55.3.10 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 MSWindows 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\(\)](#)

31.55.3.11 resize()

```
void Fl_Help_View::resize (
    int xx,
    int yy,
    int ww,
    int hh )  [virtual]
```

Resizes the help widget.

Reimplemented from [Fl_Group](#).

31.55.3.12 scrollbar_size() [1/2]

```
int Fl_Help_View::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\(int\)](#)

31.55.3.13 scrollbar_size() [2/2]

```
void Fl_Help_View::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	<code>newSize</code>	Sets the scrollbar size in pixels. If 0 (default), scrollbar size tracks the global Fl::scrollbar_size()
----	----------------------	---

See also

[Fl::scrollbar_size\(\)](#)

31.55.3.14 select_all()

```
void Fl_Help_View::select_all ( )
```

Selects all the text in the view.

31.55.3.15 size()

```
int Fl_Help_View::size ( ) const [inline]
```

Gets the size of the help view.

31.55.3.16 textcolor() [1/2]

```
void Fl_Help_View::textcolor (
    Fl_Color c ) [inline]
```

Sets the default text color.

31.55.3.17 textcolor() [2/2]

```
Fl_Color Fl_Help_View::textcolor ( ) const [inline]
```

Returns the current default text color.

31.55.3.18 textfont() [1/2]

```
void Fl_Help_View::textfont (
    Fl_Font f ) [inline]
```

Sets the default text font.

31.55.3.19 textfont() [2/2]

```
Fl_Font Fl_Help_View::textfont ( ) const [inline]
```

Returns the current default text font.

31.55.3.20 textsize() [1/2]

```
void Fl_Help_View::textsize (
    Fl_Fontsize s ) [inline]
```

Sets the default text size.

31.55.3.21 textsize() [2/2]

```
Fl_Fontsize Fl_Help_View::textsize ( ) const [inline]
```

Gets the default text size.

31.55.3.22 title()

```
const char* Fl_Help_View::title ( ) [inline]
```

Returns the current document title, or NULL if there is no title.

31.55.3.23 topline() [1/3]

```
void Fl_Help_View::topline (
    const char * n )
```

Scrolls the text to the indicated position, given a named destination.

Parameters

in	<i>n</i>	target name
----	----------	-------------

31.55.3.24 topline() [2/3]

```
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	<code>top</code>	top line number in pixels (0 = start of document)
----	------------------	---

31.55.3.25 topline() [3/3]

```
int Fl_Help_View::topline () const [inline]
```

Returns the current top line in pixels.

31.55.3.26 value() [1/2]

```
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 NULL, then the widget is cleared.

31.55.3.27 value() [2/2]

```
const char* Fl_Help_View::value () const [inline]
```

Returns the current buffer contents.

The documentation for this class was generated from the following files:

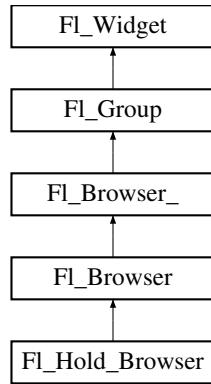
- `Fl_Help_View.H`
- `Fl_Help_Viewcxx`

31.56 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.

Additional Inherited Members

31.56.1 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 31.19 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.

31.56.2 Constructor & Destructor Documentation

31.56.2.1 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.

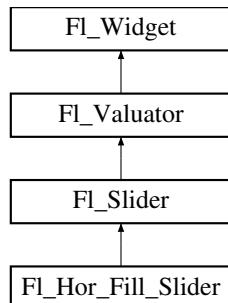
The default boxtyle is FL_DOWN_BOX. The constructor specializes [Fl_Browser\(\)](#) by setting the type to FL_HOLD_BROWSER. The destructor destroys the widget and frees all memory that has been allocated.

The documentation for this class was generated from the following files:

- [Fl_Hold_Browser.H](#)
- [Fl_Browser.cxx](#)

31.57 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)

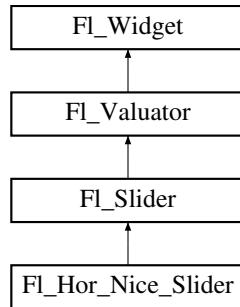
Additional Inherited Members

The documentation for this class was generated from the following files:

- [Fl_Hor_Fill_Slider.H](#)
- [Fl_Slider.cxx](#)

31.58 Fl_Hor_Nice_Slider Class Reference

Inheritance diagram for Fl_Hor_Nice_Slider:



Public Member Functions

- [Fl_Hor_Nice_Slider \(int X, int Y, int W, int H, const char *L=0\)](#)

Additional Inherited Members

The documentation for this class was generated from the following files:

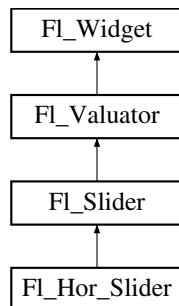
- [Fl_Hor_Nice_Slider.H](#)
- [Fl_Slider.cxx](#)

31.59 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.

Additional Inherited Members

31.59.1 Detailed Description

Horizontal Slider class.

See also

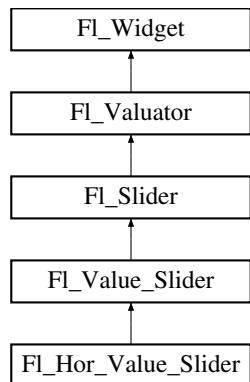
class [Fl_Slider](#).

The documentation for this class was generated from the following files:

- Fl_Hor_Slider.H
- Fl_Slider.hxx

31.60 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)

Additional Inherited Members

The documentation for this class was generated from the following files:

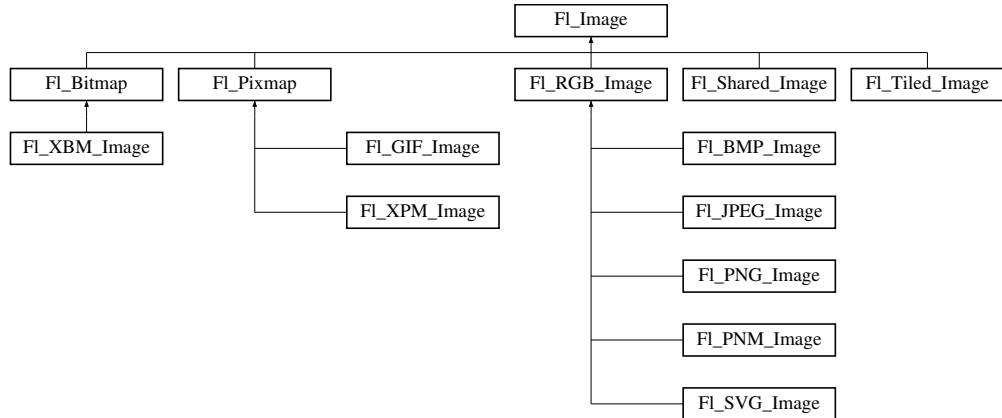
- Fl_Hor_Value_Slider.H
- Fl_Value_Slider.hxx

31.61 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 FLTK color value `c`.
- virtual `Fl_Image * copy (int W, int H)`
Creates a resized copy of the specified image.
- `Fl_Image * copy ()`
Creates a copy of the specified image.
- int `count () const`
The `count()` method 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.
- 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.
- void `draw (int X, int Y)`
Draws the image to the current drawing surface.
- int `fail ()`
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_Widget *w)`

The `label()` methods are an obsolete way to set the image attribute of a widget or menu item.
- virtual void `label (FL_Menu_Item *m)`

The `label()` methods are 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 void `RGB_scaling (FL_RGB_Scaling)`

Sets the RGB image scaling method used for `copy(int, int)`.
- static `FL_RGB_Scaling RGB_scaling ()`

Returns the currently used RGB image scaling method.
- static void `scaling_algorithm (FL_RGB_Scaling algorithm)`

Sets what algorithm is used when resizing a source image to draw it.
- static `FL_RGB_Scaling scaling_algorithm ()`

Gets 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_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 array 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`

31.61.1 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 image raw 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()`. The 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 `draw()` method results in a box with an X in it being drawn instead.

31.61.2 Constructor & Destructor Documentation

31.61.2.1 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.

31.61.3 Member Function Documentation

31.61.3.1 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<Fl_Image>](#).

Since

1.4.0

Reimplemented in [Fl_Shared_Image](#).

31.61.3.2 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 FLTK color value c.

The i 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 before changes are applied, to avoid modifying the original image.

Reimplemented in [Fl_RGB_Image](#), [Fl_SVG_Image](#), [Fl_Shared_Image](#), [Fl_Pixmap](#), and [Fl_Tiled_Image](#).

31.61.3.3 copy() [1/2]

```
Fl_Image * Fl_Image::copy (
    int W,
    int H ) [virtual]
```

Creates a resized copy of the specified image.

The image should be deleted (or in the case of [Fl_Shared_Image](#), released) when you are done with it.

Parameters

<code>W,H</code>	width and height of the returned copied image
------------------	---

Reimplemented in [Fl_RGB_Image](#), [Fl_SVG_Image](#), [Fl_Shared_Image](#), [Fl_Pixmap](#), [Fl_Bitmap](#), and [Fl_Tiled_Image](#).

31.61.3.4 `copy()` [2/2]

```
Fl_Image* Fl_Image::copy ( ) [inline]
```

Creates a copy of the specified image.

The image should be deleted (or in the case of [Fl_Shared_Image](#), released) when you are done with it.

31.61.3.5 `count()`

```
int Fl_Image::count ( ) const [inline]
```

The [count\(\)](#) method 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.

31.61.3.6 `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.

31.61.3.7 `data()`

```
const char* const* Fl_Image::data ( ) const [inline]
```

Returns a pointer to the current image data array.

Use the [count\(\)](#) method to find the size of the data array.

31.61.3.8 `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 before changes are applied, to avoid modifying the original image.

Reimplemented in [Fl_RGB_Image](#), [Fl_SVG_Image](#), [Fl_Shared_Image](#), [Fl_Pixmap](#), and [Fl_Tiled_Image](#).

31.61.3.9 draw() [1/2]

```
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.

In other words: `fl_push_clip(X, Y, W, H)` is applied, the image is drawn with its upper-left corner at `X-cx, Y-cy` and its own width and height, `fl_pop_clip()` is applied.

Reimplemented in [Fl_RGB_Image](#), [Fl_SVG_Image](#), [Fl_Shared_Image](#), [Fl_Pixmap](#), [Fl_Bitmap](#), and [Fl_Tiled_Image](#).

31.61.3.10 draw() [2/2]

```
void Fl_Image::draw (
    int X,
    int Y )  [inline]
```

Draws the image to the current drawing surface.

Parameters

<code>X,Y</code>	specify the upper-lefthand corner of the image.
------------------	---

31.61.3.11 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.

31.61.3.12 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.

31.61.3.13 fail()

```
int Fl_Image::fail ( )
```

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

ERR_NO_IMAGE if no image was found

ERR_FILE_ACCESS if there was a file access related error (errno should be set)

ERR_FORMAT if image decoding failed.

31.61.3.14 h() [1/2]

```
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\(\)](#).

31.61.3.15 h() [2/2]

```
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.

31.61.3.16 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.

31.61.3.17 label() [1/2]

```
void Fl_Image::label (
    Fl_Widget * widget )  [virtual]
```

The [label\(\)](#) methods are an obsolete way to set the image attribute of a widget or menu item.

Use the [image\(\)](#) or [deimage\(\)](#) methods of the [Fl_Widget](#) and [Fl_Menu_Item](#) classes instead.

Reimplemented in [Fl_RGB_Image](#), [Fl_Pixmap](#), and [Fl_Bitmap](#).

31.61.3.18 label() [2/2]

```
void Fl_Image::label (
    Fl_Menu_Item * m )  [virtual]
```

The [label\(\)](#) methods are an obsolete way to set the image attribute of a widget or menu item.

Use the [image\(\)](#) or [deimage\(\)](#) methods of the [Fl_Widget](#) and [Fl_Menu_Item](#) classes instead.

Reimplemented in [Fl_RGB_Image](#), [Fl_Pixmap](#), and [Fl_Bitmap](#).

31.61.3.19 ld() [1/2]

```
void Fl_Image::ld (
    int LD )  [inline], [protected]
```

Sets the current line data size in bytes.

Color images may contain extra data 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 `w() * d()` bytes.

If `LD` is non-zero, then it must be positive and larger than `w() * d()` to account for the extra data per line.

31.61.3.20 ld() [2/2]

```
int Fl_Image::ld ( ) const [inline]
```

Returns the current line data size in bytes.

See also

[ld\(int\)](#)

31.61.3.21 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` this virtual method is reimplemented and maintains shared images.

This virtual method makes it possible to `delete` all image types in the same way by calling

```
image->release();
```

Reasoning: If you have an '`Fl_Image *`' 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`.

31.61.3.22 RGB_scaling() [1/2]

```
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.

31.61.3.23 RGB_scaling() [2/2]

```
Fl_RGB_Scaling Fl_Image::RGB_scaling ( ) [static]
```

Returns the currently used RGB image scaling method.

31.61.3.24 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

<i>width,height</i>	maximum values, in FLTK units, that w() and h() should return
<i>proportional</i>	if not null, keep the values returned by w() and h() proportional to data_w() and data_h()
<i>can_expand</i>	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.

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
```

31.61.3.25 scaling_algorithm() [1/2]

```
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

31.61.3.26 scaling_algorithm() [2/2]

```
static Fl_RGB_Scaling Fl_Image::scaling_algorithm ( ) [inline], [static]
```

Gets what algorithm is used when resizing a source image to draw it.

31.61.3.27 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_RGB_Image](#), [Fl_Shared_Image](#), [Fl_Pixmap](#), and [Fl_Bitmap](#).

31.61.3.28 w() [1/2]

```
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\(\)](#).

31.61.3.29 w() [2/2]

```
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.

The documentation for this class was generated from the following files:

- [Fl_Image.H](#)
- [Fl_Image.cxx](#)

31.62 Fl_Image_Reader Class Reference

Public Member Functions

- `const char * name()`
- `int open(const char *filename)`
- `int open(const char *imagename, const unsigned char *data)`
- `unsigned char read_byte()`
- `unsigned int read_dword()`
- `int read_long()`
- `unsigned short read_word()`
- `void seek(unsigned int n)`

The documentation for this class was generated from the following files:

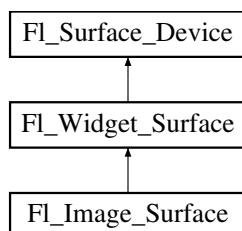
- [Fl_Image_Reader.h](#)
- [Fl_Image_Reader.cxx](#)

31.63 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 an image made of all drawings sent to the `Fl_Image_Surface` object.
- `virtual bool is_current ()`
Is this surface the current drawing surface?
- `Fl_Offscreen offscreen ()`
Returns the `Fl_Offscreen` object associated to the image surface.
- `void origin (int *x, int *y)`
Computes the coordinates of the current origin of graphics functions.
- `void origin (int x, int y)`
Sets the position of the origin of graphics in the drawable part of the drawing surface.
- `int printable_rect (int *w, int *h)`
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 ()`
Make this surface the current drawing surface.
- `~Fl_Image_Surface ()`
The destructor.

Protected Member Functions

- `void translate (int x, int y)`
Translates the current graphics origin accounting for the current rotation.
- `void untranslate ()`
Undoes the effect of a previous `translate()` call.

Friends

- class `Fl_Graphics_Driver`

Additional Inherited Members

31.63.1 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;

```

31.63.2 Constructor & Destructor Documentation

31.63.2.1 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	and
h	set the size of the resulting image. The value of the <code>high_res</code> parameter controls whether <code>w</code> and <code>h</code> are interpreted as pixel or FLTK units.
<code>high_res</code>	If zero, the created image surface is sized at <code>w x h</code> pixels. If non-zero, the pixel size of the created image surface depends on the value of the display scale factor (see <code>Fl_Graphics_Driver::scale()</code>): the resulting image has the same number of pixels as an area of the display of size <code>w x h</code> expressed in FLTK units.
<code>off</code>	If not null, the image surface is constructed around a pre-existing <code>Fl_Offscreen</code> . The caller is responsible for both construction and destruction of this <code>Fl_Offscreen</code> object. Is mostly intended for internal use by FLTK.

Version

1.3.4 (1.3.3 without the `highres` parameter)

31.63.2.2 ~Fl_Image_Surface()

```
Fl_Image_Surface::~Fl_Image_Surface ( )
```

The destructor.

31.63.3 Member Function Documentation

31.63.3.1 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)

31.63.3.2 image()

```
Fl_RGB_Image * Fl_Image_Surface::image ( )
```

Returns an 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 the image.

31.63.3.3 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.

31.63.3.4 origin() [1/2]

```
void Fl_Image_Surface::origin (
    int * x,
    int * y )  [virtual]
```

Computes the coordinates of the current origin of graphics functions.

Parameters

<code>out</code>	<code>x,y</code>	If non-null, <code>*x</code> and <code>*y</code> are set to the horizontal and vertical coordinates of the graphics origin.
------------------	------------------	---

Reimplemented from [Fl_Widget_Surface](#).

31.63.3.5 origin() [2/2]

```
void Fl_Image_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

<code>in</code>	<code>x,y</code>	Horizontal and vertical positions in the drawing surface of the desired origin of graphics.
-----------------	------------------	---

Reimplemented from [Fl_Widget_Surface](#).

31.63.3.6 printable_rect()

```
int Fl_Image_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 from [Fl_Widget_Surface](#).

31.63.3.7 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

31.63.3.8 set_current()

```
void Fl_Image_Surface::set_current ( void ) [virtual]
```

Make this surface the current drawing surface.

This surface will receive all future graphics requests. Starting from FLTK 1.4.0, another convenient API to set/unset the current drawing surface is [Fl_Surface_Device::push_current\(\)](#) / [Fl_Surface_Device::pop_current\(\)](#).

Reimplemented from [Fl_Surface_Device](#).

31.63.3.9 translate()

```
void Fl_Image_Surface::translate ( int x, int y ) [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](#).

The documentation for this class was generated from the following files:

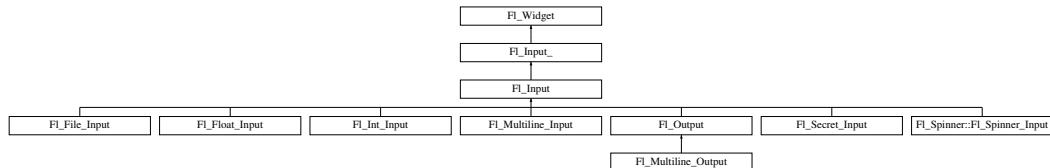
- `Fl_Image_Surface.H`
- `Fl_Image_Surface.cxx`

31.64 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)
Handles the specified event.

Protected Member Functions

- void [draw](#) ()
Draws the widget.

Friends

- class [Fl_Cocoa_Screen_Driver](#)
- class [Fl_Screen_Driver](#)

Additional Inherited Members

31.64.1 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 31.190 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. <small>Generated by Doxygen</small>
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

Table 31.191 Platform specific keyboard bindings.

Windows/Linux	Mac	Function
<code>^A</code>	Command-A	Selects all text in the widget.
<code>^C</code>	Command-C	Copy the current selection to the clipboard.
<code>^I</code>	<code>^I</code>	Insert a tab.
<code>^J</code>	<code>^J</code>	Insert a Line Feed. (Similar to literal 'Enter' character)
<code>^L</code>	<code>^L</code>	Insert a Form Feed.
<code>^M</code>	<code>^M</code>	Insert a Carriage Return.
<code>^V, Shift-Insert</code>	Command-V	Paste the clipboard. (Macs keyboards don't have "Insert" keys, but if they did, Shift-Insert would work)
<code>^X, Shift-Delete</code>	Command-X, Shift-Delete	Cut. Copy the selection to the clipboard and delete it. (If there's no selection, Shift-Delete acts like Delete)
<code>^Z</code>	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.
<code>Shift-^Z</code>	Shift-Command-Z	Redo. Currently same behavior as <code>^Z</code> . 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.

31.64.2 Constructor & Destructor Documentation

31.64.2.1 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 boxtyle is `FL_DOWN_BOX`.

31.64.3 Member Function Documentation

31.64.3.1 draw()

```
void Fl_Input::draw ( ) [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](#).

31.64.3.2 handle()

```
int Fl_Input::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.

Parameters

<code>in</code>	<code>event</code>	the kind of event received
-----------------	--------------------	----------------------------

Return values

<code>0</code>	if the event was not used or understood
<code>1</code>	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](#).

The documentation for this class was generated from the following files:

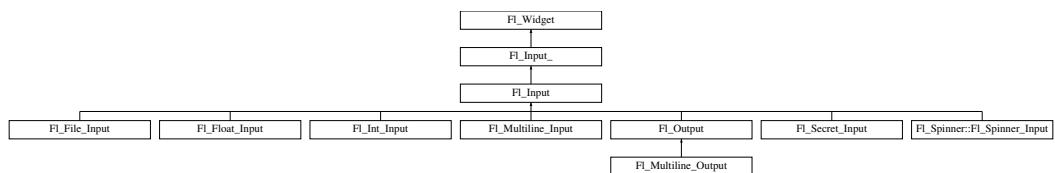
- [Fl_Input.H](#)
- [Fl_Input.cxx](#)

31.65 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.
- 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 n)
*Deletes the next *n* bytes rounded to characters before or after the cursor.*
- int **cut** (int a, int b)
*Deletes all characters between index *a* and *b*.*
- **FI_Input_** (int, int, int, int, const char *=0)
*Creates a new *FI_Input_* widget.*
- **FI_Char 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 **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.
- int **position** () const
Gets the position of the text cursor.
- int **position** (int p, int m)
Sets the index for the cursor and mark.
- int **position** (int p)
Sets the cursor position and mark.
- 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 **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)
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.
- void **tab_nav** (int val)

- `int tab_nav () const`
Sets whether the Tab key does focus navigation, or inserts tab characters into Fl_Multiline_Input.
- `Fl_Color textcolor () const`
Gets whether the Tab key causes focus navigation in multiline input fields or not.
- `Fl_Color textcolor (Fl_Color n)`
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.
- `int value (const char *)`
Changes the widget text.
- `int value (const char *, int)`
Changes the widget text.
- `const char * value () const`
Returns the text displayed in the widget.
- `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.

Protected Member Functions

- `void drawtext (int, int, int, int)`
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)`
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 ()`
- `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)`

Additional Inherited Members

31.65.1 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

[FI_Text_Display](#), [FI_Text_Editor](#) for more powerful text handling widgets
[FI_Widget::shortcut_label\(int\)](#)

31.65.2 Constructor & Destructor Documentation

31.65.2.1 [FI_Input_\(\)](#)

```
Fl_Input_::Fl_Input_ (
    int X,
    int Y,
    int W,
    int H,
    const char * l = 0 )
```

Creates a new [FI_Input_](#) widget.

This function creates a new [FI_Input_](#) widget and adds it to the current [FI_Group](#). The [value\(\)](#) is set to NULL. The default boxtyle is [FL_DOWN_BOX](#).

Parameters

<i>X,Y,W,H</i>	the dimensions of the new widget
<i>l</i>	an optional label text

31.65.2.2 ~Fl_Input()

```
Fl_Input_::~Fl_Input_ ( )
```

Destroys the widget.

The destructor clears all allocated buffers and removes the widget from the parent [Fl_Group](#).

31.65.3 Member Function Documentation**31.65.3.1 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	<i>t</i>	text that will be appended
in	<i>l</i>	length of text, or 0 if the string is terminated by nul.
in	<i>keep_selection</i>	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

31.65.3.2 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

<code>clipboard</code>	the clipboard destination 0 or 1
------------------------	----------------------------------

Returns

0 if no text is selected, 1 if the selection was copied

See also

`F1::copy(const char *, int, int)`

31.65.3.3 `copy_cuts()`

```
int F1_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()`

31.65.3.4 `cursor_color()` [1/2]

```
F1_Color F1_Input_::cursor_color ( ) const [inline]
```

Gets the color of the cursor.

Returns

the current cursor color

31.65.3.5 `cursor_color()` [2/2]

```
void F1_Input_::cursor_color (
    F1_Color n ) [inline]
```

Sets the color of the cursor.

The default color for the cursor is `FL_BLACK`.

Parameters

in	<i>n</i>	the new cursor color
----	----------	----------------------

31.65.3.6 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

31.65.3.7 cut() [2/3]

```
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

<i>n</i>	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

31.65.3.8 cut() [3/3]

```
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

<i>a,b</i>	range of bytes rounded to full characters and clamped to the buffer
------------	---

Returns

0 if no data was copied

31.65.3.9 drawtext()

```
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

<i>X,Y,W,H</i>	area that must be redrawn
----------------	---------------------------

31.65.3.10 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

31.65.3.11 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

31.65.3.12 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>i</i>	index into the value field
----	----------	----------------------------

Returns

the character at index *i*

31.65.3.13 input_type() [1/2]

```
int Fl_Input_::input_type ( ) const [inline]
```

Gets the input field type.

Returns

the current input type

31.65.3.14 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	<i>t</i>	new input type
----	----------	----------------

31.65.3.15 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	<i>t</i>	text that will be inserted
in	<i>l</i>	length of text, or 0 if the string is terminated by nul.

Returns

0 if no text was inserted

31.65.3.16 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>i</i>	starting index for the search
----	----------	-------------------------------

Returns

end of the line

31.65.3.17 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>i</i>	starting index for the search
----	----------	-------------------------------

Returns

start of the line

31.65.3.18 mark() [1/2]

```
int Fl_Input_::mark ( ) const [inline]
```

Gets the current selection mark.

Returns

index into the text

31.65.3.19 mark() [2/2]

```
int Fl_Input_::mark (
    int m ) [inline]
```

Sets the current selection mark.

`mark(n)` is the same as `position(position(), n)`.

Parameters

<i>m</i>	new index of the mark
----------	-----------------------

Returns

0 if the mark did not change

See also

[position\(\)](#), [position\(int, int\)](#)

31.65.3.20 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\(int\)](#).

31.65.3.21 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).

31.65.3.22 position() [1/3]

```
int Fl_Input_::position ( ) const [inline]
```

Gets the position of the text cursor.

Returns

the cursor position as an index in the range 0..[size\(\)](#)

See also

[position\(int, int\)](#)

31.65.3.23 position() [2/3]

```
int Fl_Input_::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

<i>p</i>	index for the cursor position
<i>m</i>	index for the mark

Returns

0 if no positions changed

See also

[position\(int\)](#), [position\(\)](#), [mark\(int\)](#)

31.65.3.24 position() [3/3]

```
int Fl_Input_::position (
    int p) [inline]
```

Sets the cursor position and mark.

`position(n)` is the same as `position(n, n)`.

Parameters

<i>p</i>	new index for cursor and mark
----------	-------------------------------

Returns

0 if no positions changed

See also

[position\(int, int\)](#), [position\(\)](#), [mark\(int\)](#)

31.65.3.25 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

31.65.3.26 `readonly()` [2/2]

```
void Fl_Input_::readonly (
    int b )  [inline]
```

Sets the read-only state of the input field.

Parameters

in	<i>b</i>	if <i>b</i> is 0, the text in this widget can be edited by the user
----	----------	---

31.65.3.27 `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	<i>b</i>	beginning index of text to be deleted
in	<i>e</i>	ending index of text to be deleted and insertion position
in	<i>text</i>	string that will be inserted
in	<i>ilen</i>	length of <i>text</i> 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).

31.65.3.28 resize()

```
void Fl_Input_::resize (
    int X,
    int Y,
    int W,
    int H ) [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](#).

31.65.3.29 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\(\)](#)

31.65.3.30 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\(\)](#)

31.65.3.31 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

31.65.3.32 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\(int, int\)](#)

31.65.3.33 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

31.65.3.34 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

31.65.3.35 tab_nav() [1/2]

```
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	<i>val</i>	If <i>val</i> is 1, Tab advances focus (default). If <i>val</i> is 0, Tab inserts a tab character (old FLTK behavior).
----	------------	---

See also

[tab_nav\(\)](#), [Fl::OPTION_ARROW_FOCUS](#).

31.65.3.36 tab_nav() [2/2]

```
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](#).

31.65.3.37 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\(Fl_Color\)](#)

31.65.3.38 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	<i>n</i>	new text color
----	----------	----------------

See also

[textcolor\(\)](#)

31.65.3.39 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

31.65.3.40 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
----	---	-------------------

31.65.3.41 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

31.65.3.42 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
----	---	------------------------------------

31.65.3.43 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.

31.65.3.44 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>i</i>	index into the beginning of a line of text
in	<i>keepmark</i>	if set, move only the cursor, but not the mark

Returns

index to new cursor position

31.65.3.45 value() [1/3]

```
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	<i>str</i>	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\(\)](#)

31.65.3.46 value() [2/3]

```
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	<i>str</i>	the new text
in	<i>len</i>	the length of the new text

Returns

non-zero if the new value is different than the current one

See also

[Fl_Input_::value\(const char*\)](#), [Fl_Input_::value\(\)](#)

31.65.3.47 value() [3/3]

```
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\(const char*\)](#)

31.65.3.48 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>i</i>	starting index for the search
----	----------	-------------------------------

Returns

end of the word

31.65.3.49 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>i</i>	starting index for the search
----	----------	-------------------------------

Returns

start of the word, or previous word

31.65.3.50 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.

31.65.3.51 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.

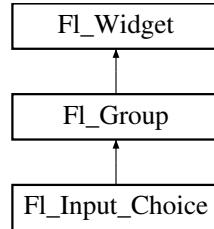
The documentation for this class was generated from the following files:

- Fl_Input_.H
- Fl_Input_.cxx

31.66 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 \(\)](#)
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 \(\)](#)
Returns a pointer to the internal [Fl_Menu_Button](#) widget.
- void [resize \(int X, int Y, int W, int H\)](#)
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.

Additional Inherited Members

31.66.1 Detailed Description

A combination of the input widget and a menu button.

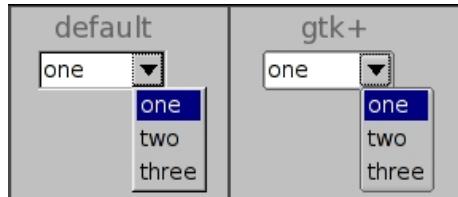


Figure 31.20 `Fl_Input_Choice` widget

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

Example use:

```
#include <stdio.h>
#include <FL/Fl.H>
#include <FL/Fl_Double_Window.H>
#include <FL/Fl_Input_Choice.H>
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();
}
```

31.66.2 Constructor & Destructor Documentation

31.66.2.1 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.

31.66.3 Member Function Documentation

31.66.3.1 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);
```

31.66.3.2 changed()

```
int Fl_Input_Choice::changed () const [inline]
```

Returns the combined [changed\(\)](#) state of the input and menu button widget.

31.66.3.3 clear()

```
void Fl_Input_Choice::clear () [inline]
```

Removes all items from the menu.

31.66.3.4 clear_changed()

```
void Fl_Input_Choice::clear_changed ()
```

Clears the [changed\(\)](#) state of both input and menu button widgets.

31.66.3.5 input()

```
Fl_Input* Fl_Input_Choice::input () [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.

31.66.3.6 menu() [1/2]

```
const Fl_Menu_Item* Fl_Input_Choice::menu ( ) [inline]
```

Gets the [Fl_Menu_Item](#) array used for the menu.

31.66.3.7 menu() [2/2]

```
void Fl_Input_Choice::menu (
    const Fl_Menu_Item * m ) [inline]
```

Sets the [Fl_Menu_Item](#) array used for the menu.

31.66.3.8 menubutton()

```
Fl_Menu_Button* Fl_Input_Choice::menubutton ( ) [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)");
}
```

31.66.3.9 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

31.66.3.10 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\(\)](#)

31.66.3.11 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:

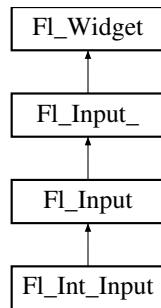
- [Fl_Input_Choice.H](#)
- [Fl_Input_Choice.cxx](#)

31.67 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.

Additional Inherited Members

31.67.1 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).

31.67.2 Constructor & Destructor Documentation

31.67.2.1 [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 boxtyle 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:

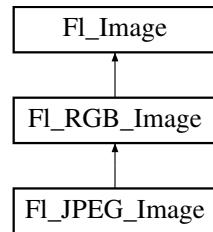
- [Fl_Int_Input.H](#)
- [Fl_Input.cxx](#)

31.68 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)
The constructor loads the JPEG image from memory.

Protected Member Functions

- void [load_jpg_](#) (const char *filename, const char *sharename, const unsigned char *data)

Additional Inherited Members

31.68.1 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.

31.68.2 Constructor & Destructor Documentation

31.68.2.1 [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	<i>filename</i>	a full path and name pointing to a valid jpeg file.
----	-----------------	---

See also

[FI_JPEG_Image::FI_JPEG_Image\(const char *imagename, const unsigned char *data\)](#)

31.68.2.2 FI_JPEG_Image() [2/2]

```
FI_JPEG_Image::FI_JPEG_Image (
    const char * name,
    const unsigned char * data )
```

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: [FI_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 [FI_Image::fail\(\)](#) to check if [FI_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

<i>name</i>	A unique name or NULL
<i>data</i>	A pointer to the memory location of the JPEG image

See also

[FI_JPEG_Image::FI_JPEG_Image\(const char *filename\)](#)
[FI_Shared_Image](#)

The documentation for this class was generated from the following files:

- [FI_JPEG_Image.H](#)
- [FI_JPEG_Image.cxx](#)

31.69 FI_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
- `Fl_Image * image`
optional image for an active label
- `Fl_Fontsize size`
size of label font
- `uchar type`
type of label.
- const char * `value`
label text

31.69.1 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.

31.69.2 Member Function Documentation

31.69.2.1 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.

31.69.2.2 measure()

```
void Fl_Label::measure (
    int & W,
    int & H ) const
```

Measures the size of the label.

Parameters

in, out	<i>W,H</i>	: this is the requested size for the label text plus image; on return, this will contain the size needed to fit the label
---------	------------	---

31.69.3 Member Data Documentation

31.69.3.1 type

```
uchar Fl_Label::type
```

type of label.

See also

[Fl_Labeltype](#)

The documentation for this struct was generated from the following files:

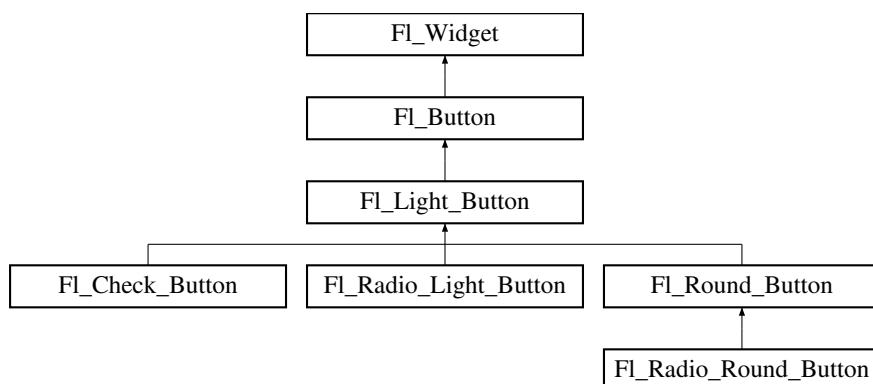
- [Fl_Widget.H](#)
- [fl_labeltype.cxx](#)

31.70 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.
- virtual int [handle](#) (int)
Handles the specified event.

Protected Member Functions

- virtual void [draw](#) ()
Draws the widget.

Additional Inherited Members

31.70.1 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 31.21 Fl_Light_Button

31.70.2 Constructor & Destructor Documentation

31.70.2.1 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 destructor deletes the check button.

31.70.3 Member Function Documentation

31.70.3.1 draw()

```
void Fl_Light_Button::draw ( ) [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](#).

31.70.3.2 handle()

```
int Fl_Light_Button::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.

Parameters

in	<code>event</code>	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](#)

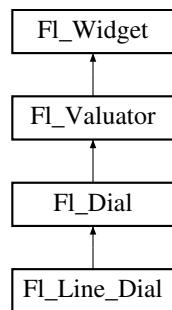
Reimplemented from [Fl_Button](#).

The documentation for this class was generated from the following files:

- [Fl_Light_Button.H](#)
- [Fl_Light_Button.cxx](#)

31.71 Fl_Line_Dial Class Reference

Inheritance diagram for [Fl_Line_Dial](#):



Public Member Functions

- [Fl_Line_Dial \(int X, int Y, int W, int H, const char *L=0\)](#)

Additional Inherited Members

The documentation for this class was generated from the following files:

- [Fl_Line_Dial.H](#)
- [Fl_Dial.cxx](#)

31.72 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 * **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.

31.72.1 Member Function Documentation

31.72.1.1 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" item, or at its place if it was removed with `Fl_Mac_App_Menu::print = ""`.

Parameters

<code>m</code>	zero-ending array of <code>Fl_Menu_Item</code> 's.
----------------	--

31.72.2 Member Data Documentation

31.72.2.1 print

```
const char* Fl_Mac_App_Menu::print [static]
```

Localizable text for the "Print Front Window" application menu item.

This menu item 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:

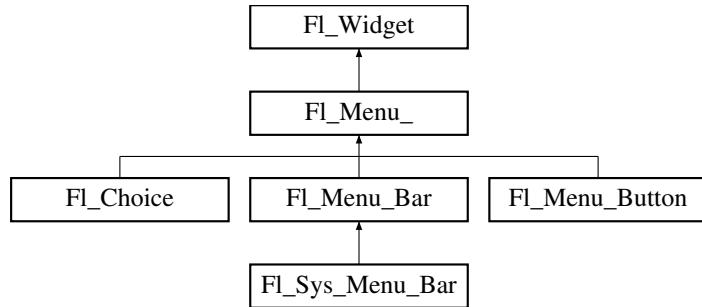
- `mac.H`

31.73 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 \(const char *, int shortcut, Fl_Callback *, void *=0, int=0\)](#)
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\(const char label, int shortcut, Fl_Callback*, void *user_data=0, int flags=0\)](#)*
- 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.
- 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 \(Fl_Boxtype b\)](#)
See [Fl_Boxtype Fl_Menu_::down_box\(\) const](#).
- [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 \(Fl_Callback *\)](#)

- **`FI_Menu_`** (int, int, int, int, const char *=0)
 - Creates a new `FI_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, FI_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, FI_Callback *c, void *d=0, int e=0)`**
 - See int `FI_Menu_::insert(const char* label, int shortcut, FI_Callback*, void *user_data=0, int flags=0)`*
- **`int item_pathname (char *name, int namelen, const FI_Menu_Item *finditem=0) const`**
 - Get the menu 'pathname' for the specified menuitem.*
- **`const FI_Menu_Item * menu () const`**
 - Returns a pointer to the array of `FI_Menu_Items`.*
- **`void menu (const FI_Menu_Item *m)`**
 - Sets the menu array pointer directly.*
- **`const FI_Menu_Item * menu_end ()`**
 - Finishes menu modifications and returns `menu()`.*
- **`void mode (int i, int fl)`**
 - Sets the flags of item `i`.*
- **`int mode (int i) const`**
 - Gets the flags of item `i`.*
- **`const FI_Menu_Item * mvalue () const`**
 - Returns a pointer to the last menu item that was picked.*
- **`const FI_Menu_Item * picked (const FI_Menu_Item *)`**
 - When user picks a menu item, call this.*
- **`void remove (int)`**
 - Deletes item `i` from the menu.*
- **`void replace (int, const char *)`**
 - Changes the text of item `i`.*
- **`void setonly (FI_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 `s`.*
- **`int size () const`**
 - This returns the number of `FI_Menu_Item` structures that make up the menu, correctly counting submenus.*
- **`void size (int W, int H)`**
- **`const FI_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`.*
- **`FI_Color textcolor () const`**
 - Get the current color of menu item labels.*
- **`void textcolor (FI_Color c)`**
 - Sets the current color of menu item labels.*
- **`FI_Font textfont () const`**
 - Gets the current font of menu item labels.*
- **`void textfont (FI_Font c)`**
 - Sets the current font of menu item labels.*
- **`FI_Fontsize textsize () const`**
 - Gets the current font size of menu item labels.*

- Gets the font size of menu item labels.
- void [textsize \(Fl_Fontsize c\)](#)
Sets the font size of menu item labels.
- int [value \(\) const](#)
Returns the index into [menu\(\)](#) of the last item chosen by the user.
- int [value \(const Fl_Menu_Item *\)](#)
The value is the index into [menu\(\)](#) of the last item chosen by the user.
- int [value \(int i\)](#)
The value is the index into [menu\(\)](#) of the last item chosen by the user.

Protected Member Functions

- int [item.pathname_ \(char *name, int namelen, const Fl_Menu_Item *finditem, const Fl_Menu_Item *menu=0\) const](#)

Protected Attributes

- uchar [alloc](#)
- uchar [down_box_](#)
- Fl_Color [textcolor_](#)
- Fl_Font [textfont_](#)
- Fl_Fontsize [textsize_](#)

Additional Inherited Members

31.73.1 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\(int\)](#)

31.73.2 Constructor & Destructor Documentation

31.73.2.1 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.

31.73.3 Member Function Documentation

31.73.3.1 add() [1/2]

```
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	<i>label</i>	The text label for the menu item.
in	<i>shortcut</i>	Optional keyboard shortcut that can be an int or string: (FL_CTRL+'a') or "^a". Default 0 if none.
in	<i>callback</i>	Optional callback invoked when user clicks the item. Default 0 if none.
in	<i>userdata</i>	Optional user data passed as an argument to the callback. Default 0 if none.
in	<i>flags</i>	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.

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 "/" character is ignored if it appears as the first character of the label string, e.g. "/File/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 [FI::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.
```

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

31.73.3.2 add() [2/2]

```
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

<i>str</i>	string containing multiple menu labels as described above
------------	---

Returns

the index into the [menu\(\)](#) array, where the entry was added

31.73.3.3 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.

31.73.3.4 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

<i>index</i>	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\(int\)](#)

31.73.3.5 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\(const Fl_Menu_Item* m\)](#).

31.73.3.6 down_box()

[Fl_BoxType](#) [Fl_Menu_::down_box](#) () const [inline]

This box type is used to surround the currently-selected items in the menus.

If this is FL_NO_BOX then it acts like FL_THIN_UP_BOX and [selection_color\(\)](#) acts like FL_WHITE, for back compatibility.

31.73.3.7 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	<i>pathname</i>	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\(\)](#)**31.73.3.8 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	<i>item</i>	The item to be found
----	-------------	----------------------

Returns

The index of the item, or -1 if not found.

See also[menu\(\)](#)**31.73.3.9 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

<code>cb</code>	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\(const char*\)](#)

31.73.3.10 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

<code>pathname</code>	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\(\)](#)

31.73.3.11 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\(const char*\)](#)

31.73.3.12 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!)

31.73.3.13 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	<i>index</i>	The menu array's index position where the new item is inserted. If -1, behavior is the same as add() .
in	<i>label</i>	The text label for the menu item. If the label is a menu pathname, <i>index</i> is ignored, and the pathname indicates the position of the new item.
in	<i>shortcut</i>	Optional keyboard shortcut. Can be an int (FL_CTRL+'a') or a string ("^a"). Default is 0.
in	<i>callback</i>	Optional callback invoked when user clicks the item. Default 0 if none.
in	<i>userdata</i>	Optional user data passed as an argument to the callback. Default 0 if none.
in	<i>flags</i>	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\(\)](#)

31.73.3.14 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\(\)](#)

31.73.3.15 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
}
```

31.73.3.16 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.

31.73.3.17 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\(\)](#)

31.73.3.18 mode() [1/2]

```
void Fl_Menu_::mode (
    int i,
    int fl ) [inline]
```

Sets the flags of item i.

For a list of the flags, see [Fl_Menu_Item](#).

31.73.3.19 mode() [2/2]

```
int Fl_Menu_::mode (
    int i ) const [inline]
```

Gets the flags of item i.

For a list of the flags, see [Fl_Menu_Item](#).

31.73.3.20 mvalue()

```
const Fl_Menu_Item* Fl_Menu_::mvalue ( ) const [inline]
```

Returns a pointer to the last menu item that was picked.

31.73.3.21 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...

31.73.3.22 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>i</i>	index into menu array
----------	-----------------------

31.73.3.23 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>i</i>	index into menu array
<i>str</i>	new label for menu item at index <i>i</i>

31.73.3.24 setonly()

```
void Fl_Menu_::setonly (
    Fl_Menu_Item * item )
```

Turns the radio item "on" for the menu item and turns "off" adjacent radio items of the same group.

31.73.3.25 shortcut()

```
void Fl_Menu_::shortcut (
    int i,
    int s ) [inline]
```

Changes the shortcut of item *i* to *s*.

31.73.3.26 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).

31.73.3.27 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.

31.73.3.28 text() [1/2]

```
const char* Fl_Menu_::text ( ) const [inline]
```

Returns the title of the last item chosen.

31.73.3.29 text() [2/2]

```
const char* Fl_Menu_::text ( int i ) const [inline]
```

Returns the title of item i.

31.73.3.30 textcolor() [1/2]

```
Fl_Color Fl_Menu_::textcolor ( ) const [inline]
```

Get the current color of menu item labels.

31.73.3.31 textcolor() [2/2]

```
void Fl_Menu_::textcolor ( Fl_Color c ) [inline]
```

Sets the current color of menu item labels.

31.73.3.32 textfont() [1/2]

```
Fl_Font Fl_Menu_::textfont ( ) const [inline]
```

Gets the current font of menu item labels.

31.73.3.33 textfont() [2/2]

```
void Fl_Menu_::textfont ( Fl_Font c ) [inline]
```

Sets the current font of menu item labels.

31.73.3.34 `textsize()` [1/2]

```
F1_Fontsize F1_Menu_::textsize ( ) const [inline]
```

Gets the font size of menu item labels.

31.73.3.35 `textsize()` [2/2]

```
void F1_Menu_::textsize (
    F1_Fontsize c ) [inline]
```

Sets the font size of menu item labels.

31.73.3.36 `value()` [1/3]

```
int F1_Menu_::value ( ) const [inline]
```

Returns the index into [menu\(\)](#) of the last item chosen by the user.

It is zero initially.

31.73.3.37 `value()` [2/3]

```
int F1_Menu_::value (
    const F1_Menu_Item * m )
```

The value is the index into [menu\(\)](#) of the last item chosen by the user.

It is zero initially. You can set it as an integer, or set it with a pointer to a menu item. The set routines return non-zero if the new value is different than the old one.

31.73.3.38 `value()` [3/3]

```
int F1_Menu_::value (
    int i ) [inline]
```

The value is the index into [menu\(\)](#) of the last item chosen by the user.

It is zero initially. You can set it as an integer, or set it with a pointer to a menu item. The set routines return non-zero if the new value is different than the old one.

The documentation for this class was generated from the following files:

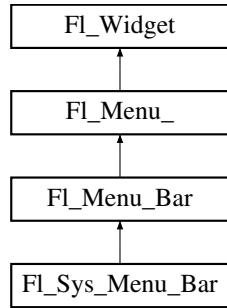
- [F1_Menu_.H](#)
- [F1_Menu_.cxx](#)
- [F1_Menu_add.cxx](#)
- [F1_Menu_global.cxx](#)

31.74 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)
Handles the specified event.
- virtual void [update](#) ()
Updates the menu bar after any change to its items.

Protected Member Functions

- void [draw](#) ()
Draws the widget.

Friends

- class [Fl_Sys_Menu_Bar_Driver](#)

Additional Inherited Members

31.74.1 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.

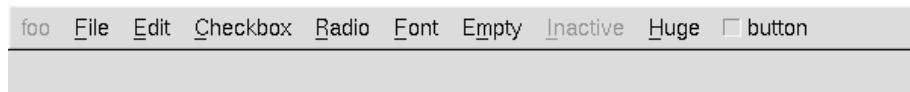


Figure 31.22 menubar

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.

31.74.2 Constructor & Destructor Documentation

31.74.2.1 `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 boxtyle 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.

31.74.3 Member Function Documentation

31.74.3.1 draw()

```
void Fl_Menu_Bar::draw ( ) [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](#).

31.74.3.2 handle()

```
int Fl_Menu_Bar::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.

Parameters

in	<code>event</code>	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](#).

31.74.3.3 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:

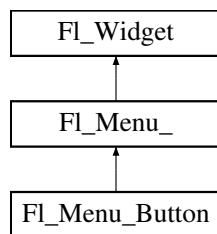
- [Fl_Menu_Bar.H](#)
- [Fl_Menu_Bar.cxx](#)

31.75 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)
Handles the specified event.
- const [Fl_Menu_Item](#) * [popup](#) ()
Act exactly as though the user clicked the button or typed the shortcut key.

Protected Member Functions

- void [draw](#) ()
Draws the widget.

Additional Inherited Members

31.75.1 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 31.23 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.

31.75.2 Member Enumeration Documentation

31.75.2.1 `popup_buttons`

```
enum F1_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. `F1_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.

31.75.3 Constructor & Destructor Documentation

31.75.3.1 `F1_Menu_Button()`

```
F1_Menu_Button::F1_Menu_Button (
    int X,
    int Y,
    int W,
    int H,
    const char * l = 0 )
```

Creates a new `F1_Menu_Button` widget using the given position, size, and label string.

The default boxtyle is `FL_UP_BOX`.

The constructor sets `menu()` to NULL. See `F1_Menu_` for the methods to set or change the menu.

31.75.4 Member Function Documentation

31.75.4.1 draw()

```
void Fl_Menu_Button::draw ( )  [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](#).

31.75.4.2 handle()

```
int Fl_Menu_Button::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.

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](#).

31.75.4.3 `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`

31.76 `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, void *p)`
Sets the menu item's callback function and `userdata()` argument.

- void `callback (FI_Callback *c)`
Sets the menu item's callback function.
- void `callback (FI_Callback0 *c)`
Sets the menu item's callback function.
- void `callback (FI_Callback1 *c, long p=0)`
Sets the menu item's callback function and userdata() argument.
- void `check ()`
back compatibility only
- int `checkbox () const`
Returns true if a checkbox will be drawn next to this item.
- int `checked () const`
back compatibility only
- 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 (FI_Widget *o) const`
Calls the `FI_Menu_Item` item's callback, and provides the `FI_Widget` argument.
- void `do_callback (FI_Widget *o, void *arg) const`
Calls the `FI_Menu_Item` item's callback, and provides the `FI_Widget` argument.
- void `do_callback (FI_Widget *o, long arg) const`
Calls the `FI_Menu_Item` item's callback, and provides the `FI_Widget` argument.
- void `draw (int x, int y, int w, int h, const FI_Menu_ *, int t=0) const`
Draws the menu item in bounding box x,y,w,h, optionally selects the item.
- const `FI_Menu_Item * find_shortcut (int *ip=0, const bool require_alt=false) const`
Search only the top level menu for a shortcut.
- const `FI_Menu_Item * first () const`
Returns the first menu item, same as `next(0)`.
- `FI_Menu_Item * first ()`
Returns the first menu item, same as `next(0)`.
- void `hide ()`
Hides an item in the menu.
- void `image (FI_Image *image)`
Compatibility API for FLUID, same as `image->label(this)`.
- void `image (FI_Image &image)`
Compatibility API for FLUID, same as `image.label(this)`.
- int `insert (int, const char *, int, FI_Callback *, void *=0, int=0)`
Inserts an item at position `index`.
- const `char * label () const`
Returns the title of the item.
- void `label (const char *a)`
See `const char FI_Menu_Item::label() const`.*
- void `label (FI_Labeltype a, const char *b)`
See `const char FI_Menu_Item::label() const`.*
- `FI_Color labelcolor () const`
Gets the menu item's label color.
- void `labelcolor (FI_Color a)`
Sets the menu item's label color.
- `FI_Font labelfont () const`
Gets the menu item's label font.
- void `labelfont (FI_Font a)`

- **`FL_Fontsize`** `labelsize () const`

Sets the menu item's label font.
- **`void`** `labelsize (FL_Fontsize a)`

Gets the label font pixel size/height.
- **`FL_Labeltype`** `labeltype () const`

Sets the label font pixel size/height.
- **`void`** `labeltype (FL_Labeltype a)`

Returns the menu item's labeltype.
- **`int`** `measure (int *h, const FL_Menu_ *) const`

Measures width of label, including effect of & characters.
- **`const FL_Menu_Item *`** `next (int i=1) const`

Advance a pointer by n items through a menu array, skipping the contents of submenus and invisible items.
- **`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 *`** `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.
- **`void`** `set ()`

Turns the check or radio item "on" for the menu item.
- **`void`** `setonly ()`

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`** `uncheck ()`

back compatibility only
- **`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.
- **`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

31.76.1 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()
    ulong 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
};
```

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"},         0,          0, 0, 0,
    {"two"},         0,          0, 0, 0,
```

```

{ "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:

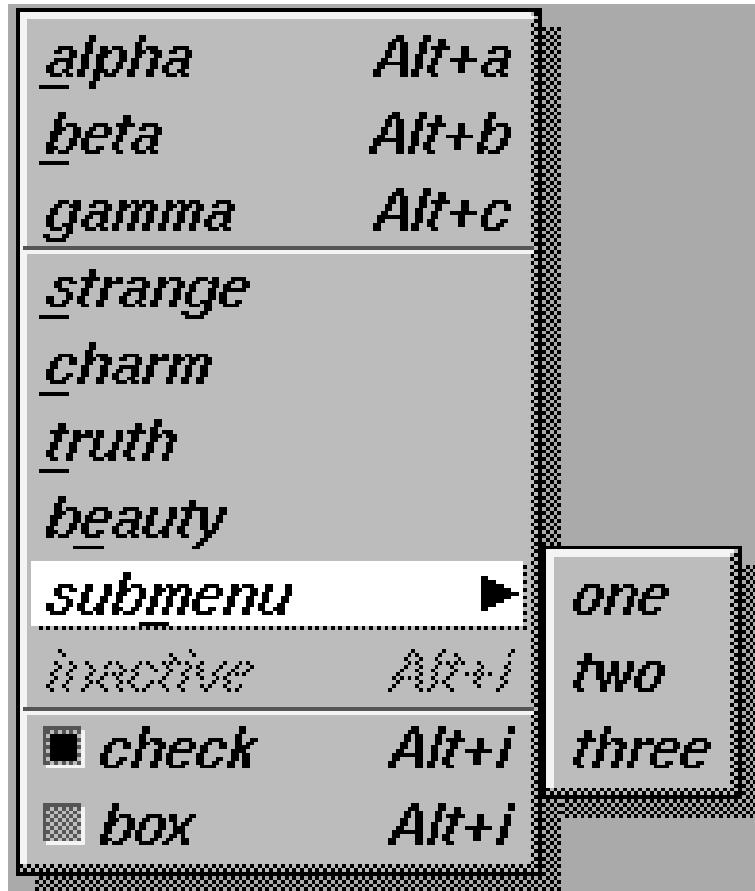


Figure 31.24 menu

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.

31.76.2 Member Function Documentation

31.76.2.1 activate()

```
void Fl_Menu_Item::activate ( ) [inline]
```

Allows a menu item to be picked.

31.76.2.2 active()

```
int Fl_Menu_Item::active () const [inline]
```

Gets whether or not the item can be picked.

31.76.2.3 activevisible()

```
int Fl_Menu_Item::activevisible () const [inline]
```

Returns non 0 if FL_INACTIVE and FL_INVISIBLE are cleared, 0 otherwise.

31.76.2.4 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\(int, const char*, int, Fl_Callback*, void*, int\)](#)

31.76.2.5 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.

31.76.2.6 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.

31.76.2.7 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.

31.76.2.8 callback() [2/5]

```
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

31.76.2.9 callback() [3/5]

```
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

31.76.2.10 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`

31.76.2.11 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.

This method does not set the 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`

31.76.2.12 check()

```
void Fl_Menu_Item::check ( )  [inline]
```

back compatibility only

Deprecated .

31.76.2.13 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.

31.76.2.14 checked()

```
int Fl_Menu_Item::checked ( ) const [inline]
```

back compatibility only

Deprecated .

31.76.2.15 clear()

```
void Fl_Menu_Item::clear ( ) [inline]
```

Turns the check or radio item "off" for the menu item.

31.76.2.16 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.

31.76.2.17 do_callback() [1/3]

```
void Fl_Menu_Item::do_callback (
    Fl_Widget * o ) 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.

31.76.2.18 do_callback() [2/3]

```
void Fl_Menu_Item::do_callback (
    Fl_Widget * o,
    void * arg ) 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.

31.76.2.19 do_callback() [3/3]

```
void Fl_Menu_Item::do_callback (
    Fl_Widget * o,
    long arg ) 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.

31.76.2.20 draw()

```
void Fl_Menu_Item::draw (
    int x,
    int y,
    int w,
    int h,
    const Fl_Menu_ * m,
    int selected = 0 ) const
```

Draws the menu item in bounding box `x,y,w,h`, optionally selects the item.

31.76.2.21 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

<code>ip</code>	returns the index of the item, if <code>ip</code> is not NULL.
<code>require_alt</code>	if true: match only if Alt key is pressed.

Returns

found [Fl_Menu_Item](#) or NULL

31.76.2.22 first() [1/2]

```
const Fl_Menu_Item* Fl_Menu_Item::first ( ) const [inline]
```

Returns the first menu item, same as next(0).

31.76.2.23 `first()` [2/2]

```
F1_Menu_Item* F1_Menu_Item::first () [inline]
```

Returns the first menu item, same as next(0).

31.76.2.24 `hide()`

```
void F1_Menu_Item::hide () [inline]
```

Hides an item in the menu.

31.76.2.25 `image()` [1/2]

```
void F1_Menu_Item::image (
    F1_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.

31.76.2.26 `image()` [2/2]

```
void F1_Menu_Item::image (
    F1_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.

31.76.2.27 `insert()`

```
int F1_Menu_Item::insert (
    int index,
    const char * mytext,
    int sc,
    F1_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 `F1_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	<i>index</i>	insert new items here
in	<i>mytext</i>	new label string, details see above
in	<i>sc</i>	keyboard shortcut for new item
in	<i>cb</i>	callback function for new item
in	<i>data</i>	user data for new item
in	<i>myflags</i>	menu flags as described in FL_Menu_Item

Returns

the index into the menu() array, where the entry was added

31.76.2.28 label()

```
const char* Fl_Menu_Item::label() const [inline]
```

Returns the title of the 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.

31.76.2.29 labelcolor() [1/2]

```
Fl_Color Fl_Menu_Item::labelcolor() const [inline]
```

Gets the menu item's label color.

This color is passed to the labelltype 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.

31.76.2.30 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](#)

31.76.2.31 `labelfont()` [1/2]

```
F1_Font F1_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.

31.76.2.32 `labelfont()` [2/2]

```
void F1_Menu_Item::labelfont (
    F1_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.

31.76.2.33 `labelsize()` [1/2]

```
F1_Fontsize F1_Menu_Item::labelsize ( ) const [inline]
```

Gets the label font pixel size/height.

31.76.2.34 `labelsize()` [2/2]

```
void F1_Menu_Item::labelsize (
    F1_Fontsize a ) [inline]
```

Sets the label font pixel size/height.

31.76.2.35 `labeltype()` [1/2]

```
F1_Labeltype F1_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.

31.76.2.36 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.

31.76.2.37 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.

31.76.2.38 next() [1/2]

```
const Fl_Menu_Item * Fl_Menu_Item::next (
    int n = 1 ) const
```

Advance 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.

31.76.2.39 next() [2/2]

```
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.

31.76.2.40 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	<i>X,Y</i>	the position of the mouse cursor, relative to the window that got the most recent event (usually you can pass <code>Fl::event_x()</code> and <code>Fl::event_y()</code> unchanged here).
in	<i>title</i>	a character string title for the menu. If non-zero a small box appears above the menu with the title in it.
in	<i>picked</i>	if this pointer is not NULL, the popup menu will appear so that the picked menu is under the mouse pointer.
in	<i>menu_button</i>	is a pointer to an <code>Fl_Menu_Item</code> 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

31.76.2.41 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.

31.76.2.42 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.

31.76.2.43 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 `set_only()` does.

31.76.2.44 setonly()

```
void Fl_Menu_Item::setonly ( )
```

Turns the radio item "on" for the menu item and turns "off" adjacent radio items set.

Deprecated This method is dangerous if radio items are first in the menu. Use [Fl_Menu_::setonly\(Fl_Menu_Item*\)](#) instead.

31.76.2.45 shortcut() [1/2]

```
int Fl_Menu_Item::shortcut ( ) const [inline]
```

Gets what key combination shortcut will trigger the menu item.

31.76.2.46 shortcut() [2/2]

```
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).

31.76.2.47 show()

```
void Fl_Menu_Item::show ( ) [inline]
```

Makes an item visible in the menu.

31.76.2.48 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
```

31.76.2.49 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.

31.76.2.50 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 '

31.76.2.51 uncheck()

```
void Fl_Menu_Item::uncheck ( ) [inline]
```

back compatibility only

Deprecated .

31.76.2.52 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. You should not rely on a particular value, only zero or non-zero.

Note

The returned value for a checked menu item as of FLTK 1.3.2 is FL_MENU_VALUE (4), but may be 1 in a future version.

31.76.2.53 visible()

```
int Fl_Menu_Item::visible ( ) const [inline]
```

Gets the visibility of an item.

The documentation for this struct was generated from the following files:

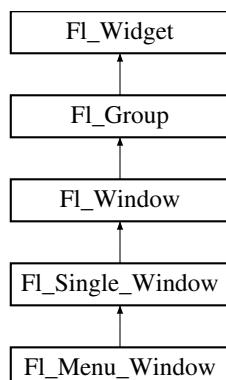
- [Fl_Menu_Item.H](#)
- [Fl_Menu.cxx](#)
- [Fl_Menu_.cxx](#)
- [Fl_Menu_add.cxx](#)

31.77 Fl_Menu_Window Class Reference

The [Fl_Menu_Window](#) widget is a window type used for menus.

```
#include <Fl_Menu_Window.H>
```

Inheritance diagram for Fl_Menu_Window:



Public Member Functions

- void [clear_overlay \(\)](#)
Tells FLTK to use normal drawing planes instead of overlay planes.
- void [erase \(\)](#)
Erases the window, does nothing if HAVE_OVERLAY is not defined in config.h.
- [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.
- void [flush \(\)](#)
Forces the window to be drawn, this window is also made current and calls [draw\(\)](#).
- void [hide \(\)](#)
Removes the window from the screen.
- unsigned int [overlay \(\)](#)
Tells if hardware overlay mode is set.
- void [set_overlay \(\)](#)
Tells FLTK to use hardware overlay planes if they are available.
- void [show \(\)](#)
Puts the window on the screen.
- [~Fl_Menu_Window \(\)](#)
Destroys the window and all of its children.

Additional Inherited Members

31.77.1 Detailed Description

The [Fl_Menu_Window](#) widget is a window type used for menus.

By default the window is drawn in the hardware overlay planes if they are available so that the menu don't force the rest of the window to redraw.

31.77.2 Constructor & Destructor Documentation

31.77.2.1 ~Fl_Menu_Window()

`Fl_Menu_Window::~Fl_Menu_Window ()`

Destroys the window and all of its children.

31.77.2.2 Fl_Menu_Window() [1/2]

```
Fl_Menu_Window::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.

31.77.2.3 Fl_Menu_Window() [2/2]

```
Fl_Menu_Window::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.

31.77.3 Member Function Documentation

31.77.3.1 clear_overlay()

```
void Fl_Menu_Window::clear_overlay () [inline]
```

Tells FLTK to use normal drawing planes instead of overlay planes.

This is usually necessary if your menu contains multi-color pixmaps.

31.77.3.2 flush()

```
void Fl_Menu_Window::flush () [virtual]
```

Forces the window to be drawn, this window is also made current and calls [draw\(\)](#).

Reimplemented from [Fl_Window](#).

31.77.3.3 hide()

```
void Fl_Menu_Window::hide ( ) [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_Window](#).

31.77.3.4 set_overlay()

```
void Fl_Menu_Window::set_overlay ( ) [inline]
```

Tells FLTK to use hardware overlay planes if they are available.

31.77.3.5 show()

```
void Fl_Menu_Window::show ( ) [virtual]
```

Puts the window on the screen.

Usually (on X) this has the side effect of opening the display.

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_Window](#).

The documentation for this class was generated from the following files:

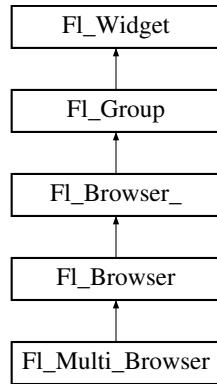
- [Fl_Menu_Window.H](#)
- [Fl_Menu_Window.cxx](#)

31.78 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.

Additional Inherited Members

31.78.1 Detailed Description

The [Fl_Multi_Browser](#) class is a subclass of [Fl_Browser](#) which lets the user select any set of the lines.



Figure 31.25 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.

31.78.2 Constructor & Destructor Documentation

31.78.2.1 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 boxtyle 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](#)

31.79 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 \(Fl_Widget *\)](#)
This method is used to associate a [Fl_Multi_Label](#) with a [Fl_Widget](#).
- void [label \(Fl_Menu_Item *\)](#)
This method is used to associate a [Fl_Multi_Label](#) with a [Fl_Menu_Item](#).

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.

31.79.1 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.

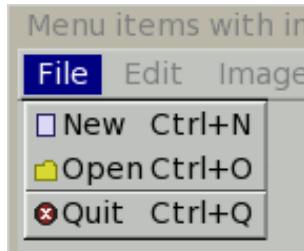


Figure 31.26 Menu items with icons using [Fl_Multi_Label](#)

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 might 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.

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);
```

See also

[Fl_Label](#) and [Fl_Labeltype](#) and examples/howto-menu-with-images.cxx

31.79.2 Member Function Documentation

31.79.2.1 `label()` [1/2]

```
void Fl_Multi_Label::label (
    Fl_Widget * o )
```

This method is used to associate a [Fl_Multi_Label](#) with a [Fl_Widget](#).

31.79.2.2 `label()` [2/2]

```
void Fl_Multi_Label::label (
    Fl_Menu_Item * o )
```

This method is used to associate a [Fl_Multi_Label](#) with a [Fl_Menu_Item](#).

31.79.3 Member Data Documentation

31.79.3.1 `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*`).

31.79.3.2 `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*`).

31.79.3.3 `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.

31.79.3.4 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:

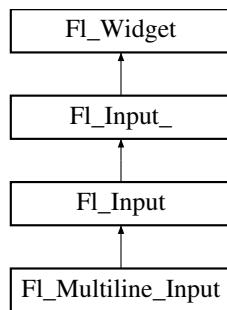
- [Fl_Multi_Label.H](#)
- [Fl_Multi_Label.cxx](#)

31.80 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.

Additional Inherited Members

31.80.1 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](#).

31.80.2 Constructor & Destructor Documentation

31.80.2.1 [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 boxtyle 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:

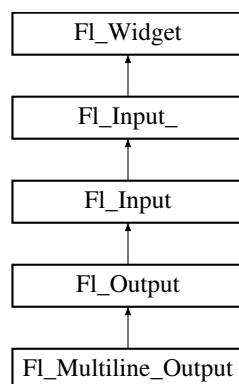
- [Fl_Multiline_Input.H](#)
- [Fl_Input.cxx](#)

31.81 [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.

Additional Inherited Members

31.81.1 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](#).

31.81.2 Constructor & Destructor Documentation

31.81.2.1 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 boxtyle 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](#)

31.82 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.
- void **directory** (const char *val)
Preset the directory the browser will show when opened.
- const char * **directory** () const
*Returns the current preset **directory()** value.*
- 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.
- void **filter_value** (int i)
Sets which filter will be initially selected.
- int **filter_value** () const
Returns which filter value was last selected by the user.
- int **filters** () const
Gets how many filters were available, not including "All Files".
- **FI_Native_File_Chooser** (int val=BROWSE_FILE)
The constructor.
- void **options** (int o)
Sets the platform specific chooser options to val.
- int **options** () const
*Gets the platform specific **FI_Native_File_Chooser::Option** flags.*
- void **preset_file** (const char *f)
Sets the default filename for the chooser.
- const char * **preset_file** () const
Get the preset filename.
- int **show** ()
Post the chooser's dialog.
- void **title** (const char *t)
Set the title of the file chooser's dialog window.
- const char * **title** () const
Get the title of the file chooser's dialog window.
- void **type** (int t)
*Sets the current **FI_Native_File_Chooser::Type** of browser.*
- int **type** () const
*Gets the current **FI_Native_File_Chooser::Type** of browser.*
- **~FI_Native_File_Chooser** ()
Destructor.

Static Public Attributes

- static const char * **file_exists_message** = "File exists. Are you sure you want to overwrite?"
Localizable message.

31.82.1 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\nt*.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 X windows, and if [Fl::OPTION_FNFC_USES_GTK](#) has not been switched off, the widget attempts to use standard GTK file chooser dialogs if they are available at run-time on the platform, and falls back to use FLTK's [Fl_File_Chooser](#) if they are not. In the first case, calling [fl_register_images\(\)](#) adds a "Preview" button to the GTK chooser dialog. In the latter case, it's best if you 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 31.27 The Fl_Native_File_Chooser on different platforms

31.82.2 Member Enumeration Documentation

31.82.2.1 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 pilots the output file extension (if supported)

31.82.2.2 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

31.82.3 Constructor & Destructor Documentation

31.82.3.1 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()`.

31.82.3.2 ~Fl_Native_File_Chooser()

```
Fl_Native_File_Chooser::~Fl_Native_File_Chooser ( )
```

Destructor.

Deallocates any resources allocated to this widget.

31.82.4 Member Function Documentation

31.82.4.1 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));
    }
}
```

31.82.4.2 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.

31.82.4.3 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.

31.82.4.4 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).

31.82.4.5 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));
    }
}
```

31.82.4.6 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.

31.82.4.7 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.

31.82.4.8 filter_value() [1/2]

```
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.

31.82.4.9 filter_value() [2/2]

```
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.

31.82.4.10 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
<code>NEW_FOLDER</code>	Shows the 'New Folder' button.	Ignored	Used	Used
<code>PREVIEW</code>	Enables the 'Preview' mode by <code>default</code> .	Ignored	Ignored	Used
<code>SAVEAS_CONFIRM</code>	Confirm dialog if <code>BROWSE_SAVE_FILE</code> file exists.	Used		
<code>USE_FILTER_EXT</code>	Used Chooser <code>filter</code> pilots the output file extension.	Ignored	Used	
	Used (GTK)			

31.82.4.11 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.

31.82.4.12 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

31.82.4.13 title() [1/2]

```
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.

31.82.4.14 title() [2/2]

```
const char * Fl_Native_File_Chooser::title ( ) const
```

Get the title of the file chooser's dialog window.

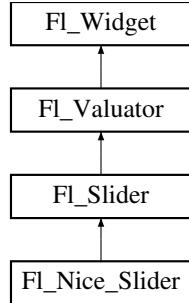
Return value may be NULL if no title was set.

The documentation for this class was generated from the following files:

- [Fl_Native_File_Chooser.H](#)
- [Fl_Native_File_Chooser.cxx](#)

31.83 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)

Additional Inherited Members

The documentation for this class was generated from the following files:

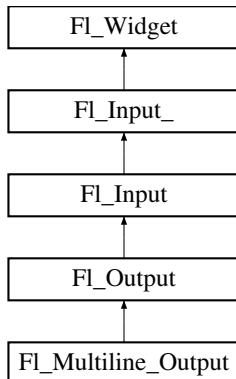
- Fl_Nice_Slider.H
- Fl_Slider.cxx

31.84 Fl_Output Class Reference

This widget displays a piece of text.

```
#include <Fl_Output.H>
```

Inheritance diagram for Fl_Output:



Public Member Functions

- [FI_Output](#) (int X, int Y, int W, int H, const char *l=0)

Creates a new [FI_Output](#) widget using the given position, size, and label string.

Additional Inherited Members

31.84.1 Detailed Description

This widget displays a piece of text.

When you set the [value\(\)](#) , [FI_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.

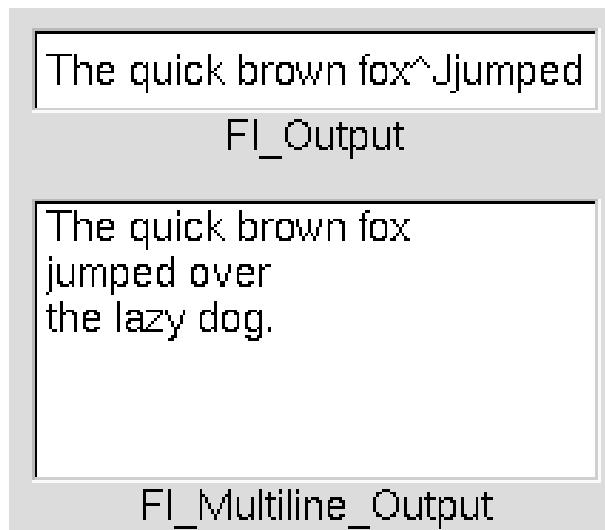


Figure 31.28 [FI_Output](#)

There is a single subclass, [FI_Multiline_Output](#), which allows you to display multiple lines of text. [FI_Multiline_Output](#) does not provide scroll bars. If a more complete text editing widget is needed, use [FI_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.

31.84.2 Constructor & Destructor Documentation

31.84.2.1 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 boxtyle 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:

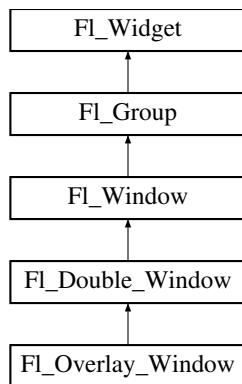
- [Fl_Output.H](#)
- [Fl_Input.cxx](#)

31.85 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

- `virtual Fl_Overlay_Window * as_overlay_window ()`
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 ()`

- void [hide \(\)](#)
Forces the window to be drawn, this window is also made current and calls [draw\(\)](#).
- void [redraw_overlay \(\)](#)
Removes the window from the screen.
- void [redraw_overlay \(\)](#)
Call this to indicate that the overlay data has changed and needs to be redrawn.
- void [resize \(int, int, int, int\)](#)
Changes the size and position of the window.
- void [show \(\)](#)
Puts the window on the screen.
- void [show \(int a, char **b\)](#)
Creates a new [Fl_Overlay_Window](#) widget using the given position, size, and label (title) string.
- [~Fl_Overlay_Window \(\)](#)
Destroys the window and all child widgets.

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.

Additional Inherited Members

31.85.1 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.

31.85.2 Constructor & Destructor Documentation

31.85.2.1 [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.

31.85.3 Member Function Documentation

31.85.3.1 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](#)>.

31.85.3.2 flush()

```
void Fl_Overlay_Window::flush () [virtual]
```

Forces the window to be drawn, this window is also made current and calls [draw\(\)](#).

Reimplemented from [Fl_Double_Window](#).

31.85.3.3 hide()

```
void Fl_Overlay_Window::hide () [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_Double_Window](#).

31.85.3.4 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\(\)](#).

31.85.3.5 `resize()`

```
void Fl_Overlay_Window::resize (
    int X,
    int Y,
    int W,
    int H ) [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 `size(x,y)` and `position(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_Double_Window`.

31.85.3.6 `show()`

```
void Fl_Overlay_Window::show () [virtual]
```

Puts the window on the screen.

Usually (on X) this has the side effect of opening the display.

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_Double_Window`.

The documentation for this class was generated from the following files:

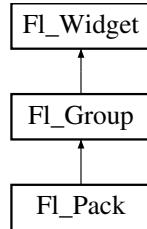
- `Fl_Overlay_Window.H`
- `Fl_Overlay_Window.cxx`

31.86 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

- **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.
- **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.

Protected Member Functions

- **void draw** ()
Draws the widget.

Additional Inherited Members

31.86.1 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

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\(\)](#)

31.86.2 Constructor & Destructor Documentation**31.86.2.1 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 boxtyle 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)

31.86.3 Member Function Documentation**31.86.3.1 draw()**

```
void Fl_Pack::draw ( ) [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](#).

31.86.3.2 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.

The documentation for this class was generated from the following files:

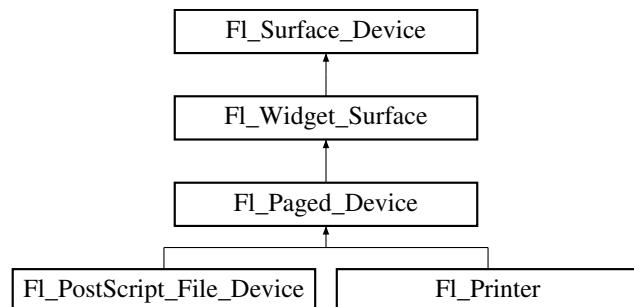
- [Fl_Pack.H](#)
- [Fl_Pack.cxx](#)

31.87 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_offset=0, int y_offset=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.

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.

Additional Inherited Members

31.87.1 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.

31.87.2 Member Enumeration Documentation

31.87.2.1 Page_Format

```
enum Fl_Paged_Device::Page_Format
```

Possible page formats.

All paper formats with pre-defined width and height.

Enumerator

A0	A0 format.
A4	A4 format.
LETTER	Letter format.

31.87.2.2 Page_Layout

```
enum Fl_Paged_Device::Page_Layout
```

Possible page layouts.

Enumerator

PORTRAIT	Portrait orientation.
LANDSCAPE	Landscape orientation.
REVERSED	Reversed orientation.
ORIENTATION	orientation

31.87.3 Member Function Documentation**31.87.3.1 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	<i>pagecount</i>	the total number of pages of the job (or 0 if you don't know the number of pages)
out	<i>frompage</i>	if non-null, <i>*frompage</i> is set to the first page the user wants printed
out	<i>ttopage</i>	if non-null, <i>*ttopage</i> is set to the last page the user wants printed
out	<i>perr_message</i>	if non-null and if the returned value is 2, <i>*perr_message</i> 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](#), and [Fl_Printer](#).

31.87.3.2 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.

Returns

0 if OK, non-zero if any error

Reimplemented in [Fl_PostScript_File_Device](#), and [Fl_Printer](#).

31.87.3.3 end_page()

```
int Fl_Paged_Device::end_page (
    void ) [virtual]
```

To be called at the end of each page.

Returns

0 if OK, non-zero if any error.

Reimplemented in [Fl_PostScript_File_Device](#), and [Fl_Printer](#).

31.87.3.4 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

<i>out</i>	<i>left</i>	If non-null, <i>*left</i> is set to the left margin size.
<i>out</i>	<i>top</i>	If non-null, <i>*top</i> is set to the top margin size.
<i>out</i>	<i>right</i>	If non-null, <i>*right</i> is set to the right margin size.
<i>out</i>	<i>bottom</i>	If non-null, <i>*bottom</i> is set to the bottom margin size.

Reimplemented in [Fl_PostScript_File_Device](#), and [Fl_Printer](#).

31.87.3.5 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

<i>angle</i>	Rotation angle in counter-clockwise degrees.
--------------	--

Reimplemented in [Fl_PostScript_File_Device](#), and [Fl_Printer](#).

31.87.3.6 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

<code>scale← _x</code>	Horizontal dimensions of plot are multiplied by this quantity.
<code>scale← _y</code>	Same as above, vertically. The value 0. is equivalent to setting <code>scale_y = scale_x</code> . Thus, <code>scale(factor);</code> is equivalent to <code>scale(factor, factor);</code>

Reimplemented in [Fl_PostScript_File_Device](#), and [Fl_Printer](#).

31.87.3.7 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

31.87.3.8 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:

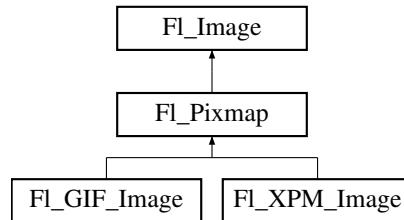
- [Fl_Paged_Device.H](#)
- [Fl_Paged_Device.cxx](#)

31.88 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

- virtual void [color_average](#) ([Fl_Color](#) c, float i)
The [color_average\(\)](#) method averages the colors in the image with the FLTK color value c.
- virtual [Fl_Image](#) * [copy](#) (int W, int H)
Creates a resized copy of the specified image.
- [Fl_Image](#) * [copy](#) ()
- virtual void [desaturate](#) ()
The [desaturate\(\)](#) method converts an image to grayscale.
- 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.
- void [draw](#) (int X, int Y)
- [Fl_Pixmap](#) (char *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.
- [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.
- virtual void [label](#) ([Fl_Widget](#) *w)
The [label\(\)](#) methods are an obsolete way to set the image attribute of a widget or menu item.
- virtual void [label](#) ([Fl_Menu_Item](#) *m)
The [label\(\)](#) methods are an obsolete way to set the image attribute of a widget or menu item.
- virtual void [uncache](#) ()
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 Attributes

- int [alloc_data](#)

Protected Member Functions

- void **measure ()**

Friends

- class **Fl_Graphics_Driver**

Additional Inherited Members

31.88.1 Detailed Description

The [Fl_Pixmap](#) class supports caching and drawing of colormap (pixmap) images, including transparency.

31.88.2 Constructor & Destructor Documentation

31.88.2.1 [Fl_Pixmap\(\)](#) [1/4]

```
Fl_Pixmap::Fl_Pixmap (
    char *const * D )  [inline], [explicit]
```

The constructors create a new pixmap from the specified XPM data.

31.88.2.2 [Fl_Pixmap\(\)](#) [2/4]

```
Fl_Pixmap::Fl_Pixmap (
    uchar *const * D )  [inline], [explicit]
```

The constructors create a new pixmap from the specified XPM data.

31.88.2.3 [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.

31.88.2.4 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.

31.88.3 Member Function Documentation

31.88.3.1 color_average()

```
void Fl_Pixmap::color_average (
    Fl_Color c,
    float i ) [virtual]
```

The [color_average\(\)](#) method averages the colors in the image with the FLTK color value c.

The i 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 before changes are applied, to avoid modifying the original image.

Reimplemented from [Fl_Image](#).

31.88.3.2 copy()

```
Fl_Image * Fl_Pixmap::copy (
    int W,
    int H ) [virtual]
```

Creates a resized copy of the specified image.

The image should be deleted (or in the case of [Fl_Shared_Image](#), released) when you are done with it.

Parameters

W,H	width and height of the returned copied image
-----	---

Reimplemented from [Fl_Image](#).

31.88.3.3 desaturate()

```
void Fl_Pixmap::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 before changes are applied, to avoid modifying the original image.

Reimplemented from [Fl_Image](#).

31.88.3.4 draw()

```
void Fl_Pixmap::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.

In other words: `fl_push_clip(X, Y, W, H)` is applied, the image is drawn with its upper-left corner at `X-cx, Y-cy` and its own width and height, `fl_pop_clip()` is applied.

Reimplemented from [Fl_Image](#).

31.88.3.5 label() [1/2]

```
void Fl_Pixmap::label (
    Fl_Widget * widget ) [virtual]
```

The [label\(\)](#) methods are an obsolete way to set the image attribute of a widget or menu item.

Use the `image()` or `deimage()` methods of the [Fl_Widget](#) and [Fl_Menu_Item](#) classes instead.

Reimplemented from [Fl_Image](#).

31.88.3.6 label() [2/2]

```
void Fl_Pixmap::label (
    Fl_Menu_Item * m ) [virtual]
```

The [label\(\)](#) methods are an obsolete way to set the image attribute of a widget or menu item.

Use the `image()` or `deimage()` methods of the [Fl_Widget](#) and [Fl_Menu_Item](#) classes instead.

Reimplemented from [Fl_Image](#).

31.88.3.7 uncache()

```
void Fl_Pixmap::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 from [Fl_Image](#).

The documentation for this class was generated from the following files:

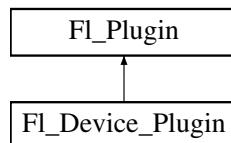
- [Fl_Pixmap.H](#)
- [Fl_Pixmap.cxx](#)

31.89 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.

31.89.1 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 initialisation 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();
}
```

31.89.2 Constructor & Destructor Documentation

31.89.2.1 Fl_Plugin()

```
Fl_Plugin::Fl_Plugin (
    const char * klass,
    const char * name )
```

Create a plugin.

Parameters

in	<i>klass</i>	plugins are grouped in classes
in	<i>name</i>	every plugin should have a unique name

The documentation for this class was generated from the following files:

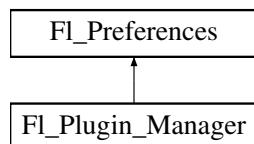
- Fl_Plugin.H
- Fl_Preferences.cxx

31.90 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 \(int index\)](#)
Return the address of a plugin by index.
- [Fl_Plugin * plugin \(const char *name\)](#)
Return the address of a plugin by name.
- [int plugins \(\)](#)
Return the number of plugins in the klass.
- [~Fl_Plugin_Manager \(\)](#)
Remove the plugin manager.

Static Public Member Functions

- static int [load](#) (const char *filename)
Load a module from disk.
- static int [loadAll](#) (const char *filepath, 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.

Additional Inherited Members

31.90.1 Detailed Description

[Fl_Plugin_Manager](#) manages link-time and run-time plugin binaries.

See also

[Fl_Plugin](#)

31.90.2 Constructor & Destructor Documentation

31.90.2.1 ~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.

31.90.3 Member Function Documentation

31.90.3.1 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.

31.90.3.2 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.

31.90.3.3 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:

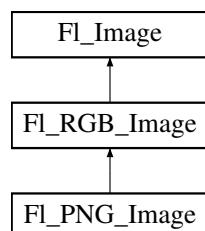
- [Fl_Plugin.H](#)
- [Fl_Preferences.cxx](#)

31.91 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.

Additional Inherited Members

31.91.1 Detailed Description

The [Fl_PNG_Image](#) class supports loading, caching, and drawing of Portable Network Graphics (PNG) image files.

The class loads colormapped and full-color images and handles color- and alpha-based transparency.

31.91.2 Constructor & Destructor Documentation

31.91.2.1 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

<code>in</code>	<code>filename</code>	Name of PNG file to read
-----------------	-----------------------	--------------------------

31.91.2.2 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

<code>name_png</code>	A name given to this image or NULL
<code>buffer</code>	Pointer to the start of the PNG image in memory
<code>maxsize</code>	Size in bytes of the memory buffer containing the PNG image

The documentation for this class was generated from the following files:

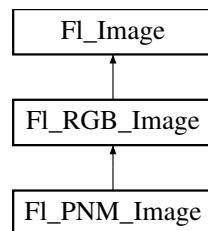
- Fl_PNG_Image.H
- Fl_PNG_Image.cxx

31.92 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.

Additional Inherited Members

31.92.1 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.

31.92.2 Constructor & Destructor Documentation

31.92.2.1 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	<i>filename</i>	a full path and name pointing to a valid jpeg file.
----	-----------------	---

The documentation for this class was generated from the following files:

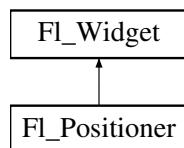
- Fl_PNM_Image.H
- Fl_PNM_Image.cxx

31.93 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)

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)

- `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.

Protected Member Functions

- `void draw (int, int, int, int)`
- `void draw ()`
Draws the widget.
- `int handle (int, int, int, int, int)`

Additional Inherited Members

31.93.1 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 31.29 FI_Positioner

31.93.2 Constructor & Destructor Documentation

31.93.2.1 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 boxtyle is FL_NO_BOX.

31.93.3 Member Function Documentation

31.93.3.1 draw()

```
void Fl_Positioner::draw ( ) [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](#).

31.93.3.2 handle()

```
int Fl_Positioner::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.

Parameters

in	<i>event</i>	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](#).

31.93.3.3 value()

```
int Fl_Positioner::value (
    double X,
    double Y )
```

Returns the current position in x and y.

31.93.3.4 xbounds()

```
void Fl_Positioner::xbounds (
    double a,
    double b )
```

Sets the X axis bounds.

31.93.3.5 xstep()

```
void Fl_Positioner::xstep (
    double a ) [inline]
```

Sets the stepping value for the X axis.

31.93.3.6 xvalue() [1/2]

```
double Fl_Positioner::xvalue ( ) const [inline]
```

Gets the X axis coordinate.

31.93.3.7 xvalue() [2/2]

```
int Fl_Positioner::xvalue (
    double X )
```

Sets the X axis coordinate.

31.93.3.8 ybounds()

```
void Fl_Positioner::ybounds (
    double a,
    double b )
```

Sets the Y axis bounds.

31.93.3.9 ystep()

```
void Fl_Positioner::ystep (
    double a ) [inline]
```

Sets the stepping value for the Y axis.

31.93.3.10 yvalue() [1/2]

```
double Fl_Positioner::yvalue ( ) const [inline]
```

Gets the Y axis coordinate.

31.93.3.11 yvalue() [2/2]

```
int Fl_Positioner::yvalue (
    double Y )
```

Sets the Y axis coordinate.

The documentation for this class was generated from the following files:

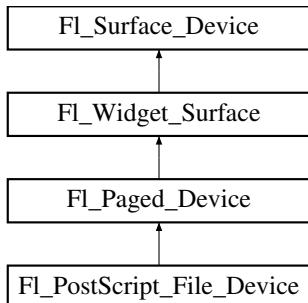
- Fl_Positioner.H
- Fl_Positioner.cxx

31.94 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](#) (int pagecount, int *from, int *to, char **perr_message)
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_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_page](#) (void)
Begins a new printed page.
- void [end_job](#) (void)
To be called at the end of a print job.
- int [end_page](#) (void)
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)
Computes the dimensions of margins that lie between the printable page area and the full page.
- void [origin](#) (int *x, int *y)
Computes the coordinates of the current origin of graphics functions.
- void [origin](#) (int x, int y)
Sets the position of the origin of graphics in the drawable part of the drawing surface.
- int [printable_rect](#) (int *w, int *h)
Computes the width and height of the drawable area of the drawing surface.
- void [rotate](#) (float angle)
Rotates the graphics operations relatively to paper.
- void [scale](#) (float scale_x, float scale_y=0.)
Changes the scaling of page coordinates.
- int [start_job](#) (int pagecount=0, enum [FI_Paged_Device::Page_Format](#) format=[FI_Paged_Device::A4](#), enum [FI_Paged_Device::Page_Layout](#) layout=[FI_Paged_Device::PORTRAIT](#))
Synonym of [begin_job\(\)](#).
- int [start_job](#) (FILE *ps_output, int pagecount=0, enum [FI_Paged_Device::Page_Format](#) format=[FI_Paged_Device::A4](#), enum [FI_Paged_Device::Page_Layout](#) layout=[FI_Paged_Device::PORTRAIT](#))
Synonym of [begin_job\(\)](#).
- void [translate](#) (int x, int y)
Translates the current graphics origin accounting for the current rotation.
- void [untranslate](#) (void)
Undoes the effect of a previous [translate\(\)](#) call.
- [~FI_PostScript_File_Device](#) ()
The destructor.

Static Public Attributes

- static const char * [file_chooser_title](#)
Label of the PostScript file chooser window.

Protected Member Functions

- [FI_PostScript_Graphics_Driver](#) * [driver](#) ()
Returns the PostScript driver of this drawing surface.

Additional Inherited Members

31.94.1 Detailed Description

To send graphical output to a PostScript file.

This class is used exactly as the [FI_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.

PostScript text uses vectorial fonts when using the FLTK standard fonts and the latin alphabet or a few other characters listed in the following table. 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	Code-point	Name	Char	Code-point	Name	Char	Code-point	Name
f	U+0192	florin	,	U+201A	quotesinglbase™	U+2122		trademark
^	U+02C6	circumflex	“	U+201C	quotedblleft	U+2202		partialdiff
ˇ	U+02C7	caron	”	U+201D	quotedblright	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	U+FB01		fi
‘	U+2018	quotyleft	/	U+2044	fraction	U+FB02		fl
’	U+2019	quoteright	€	U+20AC	Euro	U+F8FF		apple (Mac OS only)

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.

31.94.2 Member Function Documentation

31.94.2.1 begin_job() [1/3]

```
int Fl_PostScript_File_Device::begin_job (
    int pagecount,
    int * from,
    int * to,
    char ** perr_message ) [virtual]
```

Don't use with this class.

Reimplemented from [Fl_Paged_Device](#).

31.94.2.2 begin_job() [2/3]

```
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 entitled with [Fl_PostScript_File_Device::file_chooser_title](#) to select an output PostScript file.

Parameters

<i>pagecount</i>	The total number of pages to be created. Use 0 if this number is unknown when this function is called.
<i>format</i>	Desired page format.
<i>layout</i>	Desired page layout.

Returns

0 if OK, 1 if user cancelled the file dialog, 2 if fopen failed on user-selected output file.

31.94.2.3 begin_job() [3/3]

```
int F1_PostScript_File_Device::begin_job (
    FILE * ps_output,
    int pagecount = 0,
    enum F1_Paged_Device::Page_Format format = F1_Paged_Device::A4,
    enum F1_Paged_Device::Page_Layout layout = F1_Paged_Device::PORTRAIT )
```

Begins the session where all graphics requests will go to FILE pointer.

Parameters

<i>ps_output</i>	A writable FILE pointer that will receive PostScript output and that should not be closed until after end_job() has been called.
<i>pagecount</i>	The total number of pages to be created. Use 0 if this number is unknown when this function is called.
<i>format</i>	Desired page format.
<i>layout</i>	Desired page layout.

Returns

always 0.

31.94.2.4 begin_page()

```
int F1_PostScript_File_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.

Returns

0 if OK, non-zero if any error

Reimplemented from [F1_Paged_Device](#).

31.94.2.5 end_page()

```
int Fl_PostScript_File_Device::end_page (
    void ) [virtual]
```

To be called at the end of each page.

Returns

0 if OK, non-zero if any error.

Reimplemented from [Fl_Paged_Device](#).

31.94.2.6 margins()

```
void Fl_PostScript_File_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

<code>out</code>	<code>left</code>	If non-null, <code>*left</code> is set to the left margin size.
<code>out</code>	<code>top</code>	If non-null, <code>*top</code> is set to the top margin size.
<code>out</code>	<code>right</code>	If non-null, <code>*right</code> is set to the right margin size.
<code>out</code>	<code>bottom</code>	If non-null, <code>*bottom</code> is set to the bottom margin size.

Reimplemented from [Fl_Paged_Device](#).

31.94.2.7 origin() [1/2]

```
void Fl_PostScript_File_Device::origin (
    int * x,
    int * y ) [virtual]
```

Computes the coordinates of the current origin of graphics functions.

Parameters

<code>out</code>	<code>x,y</code>	If non-null, <code>*x</code> and <code>*y</code> are set to the horizontal and vertical coordinates of the graphics origin.
------------------	------------------	---

Reimplemented from [Fl_Widget_Surface](#).

31.94.2.8 origin() [2/2]

```
void Fl_PostScript_File_Device::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 from [Fl_Widget_Surface](#).

31.94.2.9 printable_rect()

```
int Fl_PostScript_File_Device::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 from [Fl_Widget_Surface](#).

31.94.2.10 rotate()

```
void Fl_PostScript_File_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

<i>angle</i>	Rotation angle in counter-clockwise degrees.
--------------	--

Reimplemented from [Fl_Paged_Device](#).

31.94.2.11 scale()

```
void Fl_PostScript_File_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

<i>scale_x</i>	Horizontal dimensions of plot are multiplied by this quantity.
<i>scale_y</i>	Same as above, vertically. The value 0. is equivalent to setting <code>scale_y = scale_x</code> . Thus, <code>scale(factor)</code> is equivalent to <code>scale(factor, factor)</code> ;

Reimplemented from [Fl_Paged_Device](#).

31.94.2.12 start_job() [1/2]

```
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

31.94.2.13 start_job() [2/2]

```
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

31.94.2.14 translate()

```
void Fl_PostScript_File_Device::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 from [Fl_Widget_Surface](#).

The documentation for this class was generated from the following file:

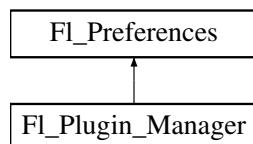
- [Fl_PostScript.H](#)

31.95 Fl_Preferences Class Reference

[Fl_Preferences](#) provides methods to 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 {**

SYSTEM = 0, **USER**, **ROOT_MASK** = 0xFF, **CORE** = 0x100,
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 `deleteAllEntries ()`
Delete all entries.
- char `deleteAllGroups ()`
Delete all groups.
- char `deleteEntry (const char *entry)`
Deletes a single name/value pair.
- char `deleteGroup (const char *group)`
Deletes a group.
- 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 `entryExists (const char *key)`
Returns non-zero if an entry with this name exists.
- `FI_Preferences (Root root, const char *vendor, const char *application)`
The constructor creates a group that manages name/value pairs and child groups.
- `FI_Preferences (const char *path, const char *vendor, const char *application)`
Use this constructor to create or read a preferences file at an arbitrary position in the file system.
- `FI_Preferences (FI_Preferences &parent, const char *group)`
Generate or read a new group of entries within another group.
- `FI_Preferences (FI_Preferences *parent, const char *group)`
Create or access a group of preferences using a name.
- `FI_Preferences (FI_Preferences &parent, int groupIndex)`
Open a child group using a given index.
- `FI_Preferences (FI_Preferences *parent, int groupIndex)`
- `FI_Preferences (const FI_Preferences &)`
Create another reference to a Preferences group.
- `FI_Preferences (ID id)`
Create a new dataset access point using a dataset ID.
- void `flush ()`
Writes all preferences to disk.
- char `get (const char *entry, int &value, int 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, double &value, double defaultValue)`
Reads an entry from the group.
- 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, 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 maxSize)`
Reads an entry from the group.
- char `getUserdataPath (char *path, int pathlen)`
Creates a path that is related to the preferences file and that is usable for additional application data.

- const char * [group](#) (int num_group)
Returns the name of the Nth (num_group) group.
- char [groupExists](#) (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](#) ()
Return an ID that can later be reused to open more references to this dataset.
- const char * [name](#) ()
Return the name of this entry.
- const char * [path](#) ()
Return the full path to this entry.
- char [set](#) (const char *entry, int value)
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, 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, const char *value)
Sets an entry (name/value pair).
- char [set](#) (const char *entry, const void *value, int size)
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 void [file_access](#) (unsigned int flags)
Tell the FLTK preferences system which files in the file system it may read, create, or write.
- static unsigned int [file_access](#) ()
Return the current file access permissions for the FLTK preferences system.
- static const char * [newUUID](#) ()
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**

31.95.1 Detailed Description

[Fl_Preferences](#) provides methods to store user settings between application starts.

It is similar to the Registry on Windows and Preferences on MacOS, and provides a simple configuration mechanism for UNIX.

[Fl_Preferences](#) uses a hierarchy to store data. It bundles similar data into groups and manages entries in these groups as name/value pairs.

Preferences are stored in text files that can be edited manually. The file format is easy to read and relatively forgiving. Preferences files are the same on all platforms. User comments in preference files are preserved. Filenames are unique for each application by using a vendor/application naming scheme. The user must provide default values for all entries to ensure proper operation should preferences be corrupted or not yet exist.

Entries can be of any length. However, the size of each preferences 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::getUserdataPath\(\)](#).

Note

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 preferences files has changed slightly. Please see [Fl_Preferences::Fl_Preferences\(Root, const char*, const char*\)](#) for details.

31.95.2 Member Typedef Documentation

31.95.2.1 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.

31.95.3 Member Enumeration Documentation

31.95.3.1 Root

```
enum Fl_Preferences::Root
```

Define the scope of the preferences.

Enumerator

SYSTEM	Preferences are used system-wide.
USER	Preferences apply only to the current user.
ROOT_MASK	masks for the values above
CORE	OR'd by FLTK to read and write core library preferences and options.

31.95.4 Constructor & Destructor Documentation

31.95.4.1 Fl_Preferences() [1/7]

```
Fl_Preferences::Fl_Preferences (
    Root root,
    const char * vendor,
    const char * application )
```

The constructor creates a group that manages name/value pairs and child groups.

Groups are ready for reading and writing at any time. The root argument is either [Fl_Preferences::USER](#) or [Fl_Preferences::SYSTEM](#).

This constructor creates the *base* instance for all following entries and reads existing databases into memory. The vendor argument is a unique text string identifying the development team or vendor of an application. A domain name or an EMail address 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 and may contain forward slashes to create deeper file structures.

A set of Preferences marked "run-time" exists exactly once per application and only as long as the application runs. It can be used as a database for volatile information. FLTK uses it to register plugins at run-time.

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 and 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, we are 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) /.fltk/\$(vendor) /\$(application).prefs . The SYSTEM directory is hardcoded as /etc/fltk/\$(vendor) /\$(application).prefs .

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) / $(application).prefs`. The `SYSTEM` directory is hard-coded as `/Library/Preferences/$(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 preferences file.

Parameters

in	<i>root</i>	can be <code>USER</code> or <code>SYSTEM</code> for user specific or system wide preferences
in	<i>vendor</i>	unique text describing the company or author of this file, must be a valid filepath segment
in	<i>application</i>	unique text describing the application, must be a valid filepath segment

Todo (Matt) Before the release of 1.4.0, I want to make a further attempt to write a preferences file smarter. I plan to use a subgroup of the "runtime" preferences to store data and stay accessible until the application exits. Data would be stored under `./$(vendor) / $(application).prefs` in RAM, but not on disk.

Todo (Matt) I want a way to access the type of the root preferences (`SYSTEM`, `USER`, `MEMORY`), and the state of the file access (OK, `FILE_SYSTEM_FAIL`, `PERMISSION_FAIL`, etc.), and probably the `dirty()` flag as well.

Todo (Matt) Also, I need to explain runtime preferences.

Todo (Matt) Lastly, I think I have to put short sample code in the Doxygen docs. The test app ist just not enough.

31.95.4.2 Fl_Preferences() [2/7]

```
Fl_Preferences::Fl_Preferences (
    const char * path,
    const char * vendor,
    const char * application )
```

Use this constructor to create or read a preferences file at an arbitrary position in the file system.

The file name is generated in the form `./prefs`. If `application` is `NULL`, `path` is taken literally as the file path and name.

Parameters

in	<i>path</i>	path to the directory that contains the preferences file
in	<i>vendor</i>	unique text describing the company or author of this file, must be a valid filepath segment
in	<i>application</i>	unique text describing the application, must be a valid filepath segment

31.95.4.3 Fl_Preferences() [3/7]

```
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	<code>parent</code>	reference object for the new group
in	<code>group</code>	name of the group to access (may contain '/'s)

31.95.4.4 Fl_Preferences() [4/7]

```
Fl_Preferences::Fl_Preferences (
    Fl_Preferences * parent,
    const char * group )
```

Create or access a group of preferences using a name.

Parameters

in	<code>parent</code>	the parameter parent is a pointer to the parent group. Parent may be NULL. It then refers to an application internal database 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.
in	<code>group</code>	a group name that is used as a key into the database

See also

[Fl_Preferences\(Fl_Preferences&, const char *group \)](#)

31.95.4.5 Fl_Preferences() [5/7]

```
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	<i>parent</i>	reference object for the new group
in	<i>groupIndex</i>	zero based index into child groups

31.95.4.6 Fl_Preferences() [6/7]

```
Fl_Preferences::Fl_Preferences (
    Fl_Preferences * parent,
    int groupIndex )
```

See also

[Fl_Preferences\(Fl_Preferences&, int groupIndex \)](#)

31.95.4.7 Fl_Preferences() [7/7]

```
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.

31.95.4.8 ~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 preferences file and deletes all internal databases.

The destructor does not remove any data from the database. It merely deletes your reference to the database.

31.95.5 Member Function Documentation**31.95.5.1 deleteEntry()**

```
char Fl_Preferences::deleteEntry (
    const char * key )
```

Deletes a single name/value pair.

This function removes the entry `key` from the database.

Parameters

in	<i>key</i>	name of entry to delete
----	------------	-------------------------

Returns

0 if deleting the entry failed

31.95.5.2 deleteGroup()

```
char Fl_Preferences::deleteGroup (
    const char * group )
```

Deletes a group.

Removes a group and all keys and groups within that group from the database.

Parameters

in	<i>group</i>	name of the group to delete
----	--------------	-----------------------------

Returns

0 if call failed

31.95.5.3 entries()

```
int Fl_Preferences::entries ( )
```

Returns the number of entries (name/value pairs) in a group.

Returns

number of entries

31.95.5.4 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	<i>index</i>	number indexing the requested entry
----	--------------	-------------------------------------

Returns

pointer to value cstring

31.95.5.5 entryExists()

```
char Fl_Preferences::entryExists (
    const char * key )
```

Returns non-zero if an entry with this name exists.

Parameters

in	<i>key</i>	name of entry that is searched for
----	------------	------------------------------------

Returns

0 if entry was not found

31.95.5.6 file_access() [1/2]

```
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\(\)](#)

31.95.5.7 file_access() [2/2]

```
unsigned int Fl_Preferences::file_access ( ) [static]
```

Return the current file access permissions for the FLTK preferences system.

See also

[Fl_Preferences::file_access\(unsigned int\)](#)

31.95.5.8 flush()

```
void Fl_Preferences::flush ( )
```

Writes all preferences to disk.

This function works only with the base preferences group. This function is rarely used as deleting the base preferences flushes automatically.

31.95.5.9 get() [1/7]

```
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	<i>key</i>	name of entry
out	<i>value</i>	returned from preferences or default value if none was set
in	<i>defaultValue</i>	default value to be used if no preference was set

Returns

0 if the default value was used

31.95.5.10 get() [2/7]

```
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	<i>key</i>	name of entry
out	<i>value</i>	returned from preferences or default value if none was set
in	<i>defaultValue</i>	default value to be used if no preference was set

Returns

0 if the default value was used

31.95.5.11 get() [3/7]

```
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	<i>key</i>	name of entry
out	<i>value</i>	returned from preferences or default value if none was set
in	<i>defaultValue</i>	default value to be used if no preference was set

Returns

0 if the default value was used

31.95.5.12 [get\(\)](#) [4/7]

```
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	<i>key</i>	name of entry
out	<i>text</i>	returned from preferences or default value if none was set
in	<i>defaultValue</i>	default value to be used if no preference was set

Returns

0 if the default value was used

31.95.5.13 [get\(\)](#) [5/7]

```
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	<i>key</i>	name of entry
out	<i>text</i>	returned from preferences or default value if none was set
in	<i>defaultValue</i>	default value to be used if no preference was set
in	<i>maxSize</i>	maximum length of value plus one byte for a trailing zero

Returns

0 if the default value was used

31.95.5.14 [get\(\)](#) [6/7]

```
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	<i>key</i>	name of entry
out	<i>data</i>	returned from preferences or default value if none was set
in	<i>defaultValue</i>	default value to be used if no preference was set
in	<i>defaultSize</i>	size of default value array

Returns

0 if the default value was used

31.95.5.15 [get\(\)](#) [7/7]

```
char Fl_Preferences::get (
    const char * key,
    void * data,
    const void * defaultValue,
    int defaultSize,
    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.

Parameters

in	<i>key</i>	name of entry
out	<i>data</i>	value returned from preferences or default value if none was set
in	<i>defaultValue</i>	default value to be used if no preference was set
in	<i>defaultSize</i>	size of default value array
in	<i>maxSize</i>	maximum length of value

Returns

0 if the default value was used

Todo maxSize should receive the number of bytes that were read.

31.95.5.16 getUserdataPath()

```
char Fl_Preferences::getUserdataPath (
    char * path,
    int pathlen )
```

Creates a path that is related to the preferences 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. `getUserDataPath()` will remove any extension and end the path with a /. If the file name has no extension, `getUserDataPath()` will append .data/ to the path name.

Example:

```
Fl_Preferences prefs( USER, "matthiasm.com", "test" );
char path[FL_PATH_MAX];
prefs.getUserdataPath( 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

<code>out</code>	<code>path</code>	buffer for user data path
<code>in</code>	<code>pathlen</code>	size of path buffer (should be at least <code>FL_PATH_MAX</code>)

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\(Root, const char*, const char*\)](#)

31.95.5.17 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	<i>num_group</i>	number indexing the requested group
----	------------------	-------------------------------------

Returns

'C' string pointer to the group name

31.95.5.18 groupExists()

```
char Fl_Preferences::groupExists (
    const char * key )
```

Returns non-zero if a group with this name exists.

Group names are relative to the 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	<i>key</i>	name of group that is searched for
----	------------	------------------------------------

Returns

0 if no group by that name was found

31.95.5.19 groups()

```
int Fl_Preferences::groups ( )
```

Returns the number of groups that are contained within a group.

Returns

0 for no groups at all

31.95.5.20 newUUID()

```
const char * Fl_Preferences::newUUID ( ) [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!

31.95.5.21 set() [1/7]

```
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 preferences file.

Parameters

in	key	name of entry
in	value	set this entry to value

Returns

0 if setting the value failed

31.95.5.22 set() [2/7]

```
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 preferences file.

Parameters

in	<i>key</i>	name of entry
in	<i>value</i>	set this entry to <i>value</i>

Returns

0 if setting the value failed

31.95.5.23 set() [3/7]

```
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 preferences file.

Parameters

in	<i>key</i>	name of entry
in	<i>value</i>	set this entry to <i>value</i>
in	<i>precision</i>	number of decimal digits to represent value

Returns

0 if setting the value failed

31.95.5.24 set() [4/7]

```
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 preferences file.

Parameters

in	<i>key</i>	name of entry
in	<i>value</i>	set this entry to <i>value</i>

Returns

0 if setting the value failed

31.95.5.25 set() [5/7]

```
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 preferences file.

Parameters

in	<i>key</i>	name of entry
in	<i>value</i>	set this entry to <i>value</i>
in	<i>precision</i>	number of decimal digits to represent value

Returns

0 if setting the value failed

31.95.5.26 set() [6/7]

```
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 preferences file.

Parameters

in	<i>key</i>	name of entry
in	<i>text</i>	set this entry to <i>value</i>

Returns

0 if setting the value failed

31.95.5.27 set() [7/7]

```
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 preferences file.

Parameters

in	<i>key</i>	name of entry
in	<i>data</i>	set this entry to <i>value</i>
in	<i>dsize</i>	size of data array

Returns

0 if setting the value failed

31.95.5.28 size()

```
int Fl_Preferences::size (
    const char * key )
```

Returns the size of the value part of an entry.

Parameters

in	<i>key</i>	name of entry
----	------------	---------------

Returns

size of value

31.95.6 Member Data Documentation

31.95.6.1 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.

31.95.6.2 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.

31.95.6.3 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:

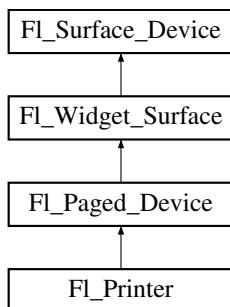
- [Fl_Preferences.H](#)
- [Fl_Preferences.cxx](#)

31.96 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)
Begins a print job.
- int **begin_page** (void)
Begins a new printed page.
- 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 **end_job** (void)
To be called at the end of a print job.
- int **end_page** (void)
To be called at the end of each page.
- **Fl_Printer** (void)
The constructor.
- virtual bool **is_current** ()
Is this surface the current drawing surface?
- 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 **origin** (int *x, int *y)
Computes the coordinates of the current origin of graphics functions.
- void **origin** (int x, int y)
Sets the position of the origin of graphics in the drawable part of the drawing surface.
- int **printable_rect** (int *w, int *h)
Computes the width and height of the drawable area of the drawing surface.
- void **rotate** (float angle)
Rotates the graphics operations relatively to paper.
- void **scale** (float scale_x, float scale_y=0.)
Changes the scaling of page coordinates.
- void **set_current** (void)
Make this surface the current drawing surface.
- void **translate** (int x, int y)
Translates the current graphics origin accounting for the current rotation.
- void **untranslate** (void)
*Undoes the effect of a previous **translate()** call.*
- **~Fl_Printer** (void)
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"

- 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]

Additional Inherited Members

31.96.1 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;
```

Platform specifics

- Unix/Linux 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 appears in print.
 - 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.
 - Class [Fl_RGB_Image](#) prints but loses its transparency if it has one.
 - 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.

31.96.2 Member Function Documentation

31.96.2.1 begin_job()

```
int Fl_Printer::begin_job (
    int pagecount = 0,
    int * frompage = NULL,
    int * topage = NULL,
    char ** perr_message = NULL ) [virtual]
```

Begins a print job.

Parameters

in	<i>pagecount</i>	the total number of pages of the job (or 0 if you don't know the number of pages)
out	<i>frompage</i>	if non-null, <i>*frompage</i> is set to the first page the user wants printed
out	<i>topage</i>	if non-null, <i>*topage</i> is set to the last page the user wants printed
out	<i>perr_message</i>	if non-null and if the returned value is 2, <i>*perr_message</i> 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](#).

31.96.2.2 begin_page()

```
int Fl_Printer::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.

Returns

0 if OK, non-zero if any error

Reimplemented from [Fl_Paged_Device](#).

31.96.2.3 draw_decorated_window()

```
void Fl_Printer::draw_decorated_window (
    Fl_Window * win,
    int win_offset_x = 0,
    int win_offset_y = 0 ) [virtual]
```

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.

Reimplemented from [Fl_Widget_Surface](#).

31.96.2.4 end_page()

```
int Fl_Printer::end_page (
    void ) [virtual]
```

To be called at the end of each page.

Returns

0 if OK, non-zero if any error.

Reimplemented from [Fl_Paged_Device](#).

31.96.2.5 margins()

```
void Fl_Printer::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

<code>out</code>	<code>left</code>	If non-null, <code>*left</code> is set to the left margin size.
<code>out</code>	<code>top</code>	If non-null, <code>*top</code> is set to the top margin size.
<code>out</code>	<code>right</code>	If non-null, <code>*right</code> is set to the right margin size.
<code>out</code>	<code>bottom</code>	If non-null, <code>*bottom</code> is set to the bottom margin size.

Reimplemented from [Fl_Paged_Device](#).

31.96.2.6 origin() [1/2]

```
void Fl_Printer::origin (
    int * x,
    int * y )  [virtual]
```

Computes the coordinates of the current origin of graphics functions.

Parameters

<code>out</code>	<code>x,y</code>	If non-null, <code>*x</code> and <code>*y</code> are set to the horizontal and vertical coordinates of the graphics origin.
------------------	------------------	---

Reimplemented from [Fl_Widget_Surface](#).

31.96.2.7 origin() [2/2]

```
void Fl_Printer::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

<code>in</code>	<code>x,y</code>	Horizontal and vertical positions in the drawing surface of the desired origin of graphics.
-----------------	------------------	---

Reimplemented from [Fl_Widget_Surface](#).

31.96.2.8 printable_rect()

```
int Fl_Printer::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 from [Fl_Widget_Surface](#).

31.96.2.9 rotate()

```
void Fl_Printer::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

<i>angle</i>	Rotation angle in counter-clockwise degrees.
--------------	--

Reimplemented from [Fl_Paged_Device](#).

31.96.2.10 scale()

```
void Fl_Printer::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

<code>scale_x</code>	Horizontal dimensions of plot are multiplied by this quantity.
<code>scale_y</code>	Same as above, vertically. The value 0. is equivalent to setting <code>scale_y = scale_x</code> . Thus, <code>scale(factor)</code> ; is equivalent to <code>scale(factor, factor)</code> ;

Reimplemented from [Fl_Paged_Device](#).

31.96.2.11 set_current()

```
void Fl_Printer::set_current (
    void ) [virtual]
```

Make this surface the current drawing surface.

This surface will receive all future graphics requests. Starting from FLTK 1.4.0, another convenient API to set/unset the current drawing surface is [Fl_Surface_Device::push_current\(\)](#) / [Fl_Surface_Device::pop_current\(\)](#).

Reimplemented from [Fl_Surface_Device](#).

31.96.2.12 translate()

```
void Fl_Printer::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 from [Fl_Widget_Surface](#).

The documentation for this class was generated from the following files:

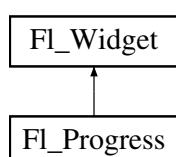
- [Fl_Printer.H](#)
- [Fl_Printer.cxx](#)

31.97 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.
- `void maximum (float v)`
Sets the maximum value in the progress widget.
- `float maximum () const`
Gets the maximum value in the progress widget.
- `void minimum (float v)`
Sets the minimum value in the progress widget.
- `float minimum () const`
Gets the minimum value in the progress widget.
- `void value (float v)`
Sets the current value in the progress widget.
- `float value () const`
Gets the current value in the progress widget.

Protected Member Functions

- `virtual void draw ()`
Draws the progress bar.

Additional Inherited Members

31.97.1 Detailed Description

Displays a progress bar for the user.

31.97.2 Constructor & Destructor Documentation

31.97.2.1 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.

31.97.3 Member Function Documentation

31.97.3.1 draw()

```
void Fl_Progress::draw (
    void ) [protected], [virtual]
```

Draws the progress bar.

Implements [Fl_Widget](#).

31.97.3.2 maximum() [1/2]

```
void Fl_Progress::maximum (
    float v ) [inline]
```

Sets the maximum value in the progress widget.

31.97.3.3 maximum() [2/2]

```
float Fl_Progress::maximum ( ) const [inline]
```

Gets the maximum value in the progress widget.

31.97.3.4 minimum() [1/2]

```
void Fl_Progress::minimum (
    float v ) [inline]
```

Sets the minimum value in the progress widget.

31.97.3.5 minimum() [2/2]

```
float Fl_Progress::minimum ( ) const [inline]
```

Gets the minimum value in the progress widget.

31.97.3.6 value() [1/2]

```
void Fl_Progress::value (
    float v ) [inline]
```

Sets the current value in the progress widget.

31.97.3.7 value() [2/2]

```
float Fl_Progress::value () const [inline]
```

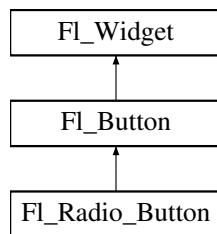
Gets the current value in the progress widget.

The documentation for this class was generated from the following files:

- [Fl_Progress.H](#)
- [Fl_Progress.cxx](#)

31.98 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.

Additional Inherited Members

31.98.1 Constructor & Destructor Documentation

31.98.1.1 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

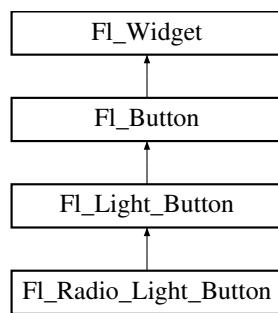
in	<i>X,Y,W,H</i>	position and size of the widget
in	<i>L</i>	widget label, default is no label

The documentation for this class was generated from the following files:

- Fl_Radio_Button.H
- Fl_Button.cxx

31.99 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)

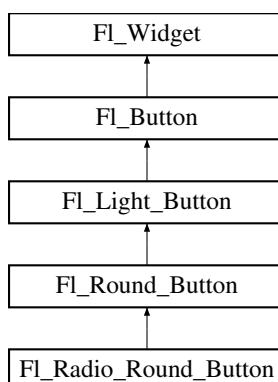
Additional Inherited Members

The documentation for this class was generated from the following files:

- Fl_Radio_Light_Button.H
- Fl_Light_Button.cxx

31.100 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.

Additional Inherited Members

31.100.1 Constructor & Destructor Documentation

31.100.1.1 [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](#)

31.101 [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).
- [Fl_Rect \(\)](#)

The default constructor creates an empty rectangle (x = y = w = h = 0).

- [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 \(const Fl_Widget &widget\)](#)
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.
- int [h \(\) const](#)
gets the height
- void [h \(int H\)](#)
sets the height
- int [r \(\) const](#)
gets the right edge (x + w).
- 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)

31.101.1 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.

31.101.2 Constructor & Destructor Documentation

31.101.2.1 Fl_Rect() [1/5]

```
Fl_Rect::Fl_Rect ( ) [inline]
```

The default constructor creates an empty rectangle (x = y = w = h = 0).

31.101.2.2 Fl_Rect() [2/5]

```
Fl_Rect::Fl_Rect (
    int W,
    int H )  [inline]
```

This constructor creates a rectangle with $x = y = 0$ and the given width and height.

31.101.2.3 Fl_Rect() [3/5]

```
Fl_Rect::Fl_Rect (
    int X,
    int Y,
    int W,
    int H )  [inline]
```

This constructor creates a rectangle with the given x,y coordinates and the given width and height.

31.101.2.4 Fl_Rect() [4/5]

```
Fl_Rect::Fl_Rect (
    const Fl_Widget & widget )  [inline]
```

This constructor creates a rectangle based on a widget's position and size.

31.101.2.5 Fl_Rect() [5/5]

```
Fl_Rect::Fl_Rect (
    const Fl_Widget *const widget )  [inline]
```

This constructor creates a rectangle based on a widget's position and size.

31.101.3 Member Function Documentation**31.101.3.1 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.

31.101.3.2 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

31.102 Fl_Scroll::ScrollInfo::Fl_Region_LRTB Struct Reference

A local struct to manage a region defined by left/right/top/bottom.

```
#include <Fl_Scroll.H>
```

Public Attributes

- int **b**
(b)ottom "y" position, aka y2
- int **l**
(l)eft "x" position, aka x1
- int **r**
(r)ight "x" position, aka x2
- int **t**
(t)op "y" position, aka y1

31.102.1 Detailed Description

A local struct to manage a region defined by left/right/top/bottom.

The documentation for this struct was generated from the following file:

- Fl_Scroll.H

31.103 Fl_Scroll::ScrollInfo::Fl_Region_XYWH Struct Reference

A local struct to manage a region defined by xywh.

```
#include <Fl_Scroll.H>
```

Public Attributes

- int **h**
- int **w**
- int **x**
- int **y**

31.103.1 Detailed Description

A local struct to manage a region defined by xywh.

The documentation for this struct was generated from the following file:

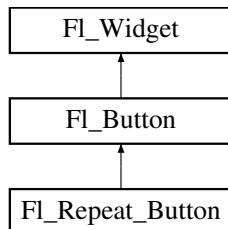
- Fl_Scroll.H

31.104 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)

Handles the specified event.

Additional Inherited Members

31.104.1 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.

31.104.2 Constructor & Destructor Documentation

31.104.2.1 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 boxtyle is FL_UP_BOX. Deletes the button.

31.104.3 Member Function Documentation

31.104.3.1 handle()

```
int Fl_Repeat_Button::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.

Parameters

in	<i>event</i>	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:

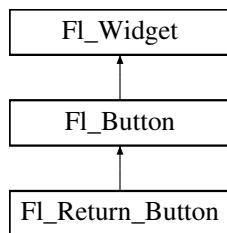
- Fl_Repeat_Button.H
- Fl_Repeat_Button.cxx

31.105 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)
Handles the specified event.

Protected Member Functions

- void [draw](#) ()
Draws the widget.

Additional Inherited Members

31.105.1 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.

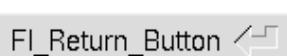


Figure 31.30 Fl_Return_Button

31.105.2 Constructor & Destructor Documentation

31.105.2.1 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 boxtyle is `FL_UP_BOX`.

The inherited destructor deletes the button.

31.105.3 Member Function Documentation

31.105.3.1 draw()

```
void Fl_Return_Button::draw ( ) [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](#).

31.105.3.2 handle()

```
int Fl_Return_Button::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.

Parameters

<code>in</code>	<code>event</code>	the kind of event received
-----------------	--------------------	----------------------------

Return values

<code>0</code>	if the event was not used or understood
<code>1</code>	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:

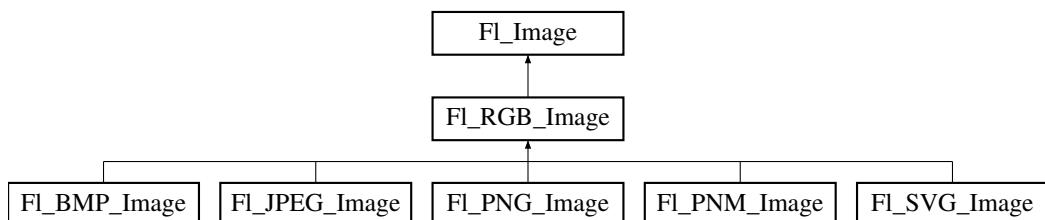
- [Fl_Return_Button.H](#)
- [Fl_Return_Button.cxx](#)

31.106 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 image is an [Fl_SVG_Image](#) or not.
- virtual void [color_average \(Fl_Color c, float i\)](#)
The [color_average\(\)](#) method averages the colors in the image with the FLTK color value `c`.
- virtual [Fl_Image * copy \(int W, int H\)](#)
Creates a resized copy of the specified image.
- [Fl_Image * copy \(\)](#)
- virtual void [desaturate \(\)](#)
The [desaturate\(\)](#) method converts an image to grayscale.

- 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.
- void [draw](#) (int X, int Y)
- [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.
- [FL_RGB_Image](#) (const [FL_Pixmap](#) *pxm, [FL_Color](#) bg=[FL_GRAY](#))
The constructor creates a new RGBA image from the specified [FL_Pixmap](#).
- virtual void [label](#) ([FL_Widget](#) *w)
The [label\(\)](#) methods are an obsolete way to set the image attribute of a widget or menu item.
- virtual void [label](#) ([FL_Menu_Item](#) *m)
The [label\(\)](#) methods are an obsolete way to set the image attribute of a widget or menu item.
- virtual void [normalize](#) ()
Makes sure the object is fully initialized.
- virtual void [uncache](#) ()
If the image has been cached for display, delete the cache data.
- virtual [~FL_RGB_Image](#) ()
The destructor frees all memory and server resources that are used by the image.

Static Public Member Functions

- static void [max_size](#) (size_t size)
Sets the maximum allowed image size in bytes when creating an [FL_RGB_Image](#) object.
- static size_t [max_size](#) ()
Returns the maximum allowed image size in bytes when creating an [FL_RGB_Image](#) object.

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

31.106.1 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.

[FL_RGB_Image](#) is defined in <[FL/FL_Image.H](#)>, however for compatibility reasons <[FL/FL_RGB_Image.H](#)> should be included.

31.106.2 Constructor & Destructor Documentation

31.106.2.1 `Fl_RGB_Image()` [1/2]

```
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	<code>bits</code>	The image data array.
in	<code>W</code>	The width of the image in pixels.
in	<code>H</code>	The height of the image in pixels.
in	<code>D</code>	The image depth, or 'number of channels' (default=3).
in	<code>LD</code>	Line data size (default=0).

See also

[Fl_Image::data\(\)](#), [Fl_Image::w\(\)](#), [Fl_Image::h\(\)](#), [Fl_Image::d\(\)](#), [Fl_Image::ld\(int\)](#)

31.106.2.2 Fl_RGB_Image() [2/2]

```
Fl_RGB_Image::Fl_RGB_Image (
    const Fl_Pixmap * ppxm,
    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.

31.106.3 Member Function Documentation

31.106.3.1 as_svg_image()

```
virtual Fl_SVG_Image* Fl_RGB_Image::as_svg_image ( ) [inline], [virtual]
```

Returns whether an 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](#).

31.106.3.2 color_average()

```
void Fl_RGB_Image::color_average (
    Fl_Color c,
    float i ) [virtual]
```

The `color_average()` method averages the colors in the image with the FLTK color value `c`.

The `i` 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 before changes are applied, to avoid modifying the original image.

Reimplemented from [Fl_Image](#).

Reimplemented in [Fl_SVG_Image](#).

31.106.3.3 copy()

```
Fl_Image* Fl_RGB_Image::copy (
    int W,
    int H ) [virtual]
```

Creates a resized copy of the specified image.

The image should be deleted (or in the case of [Fl_Shared_Image](#), released) when you are done with it.

Parameters

<i>W,H</i>	width and height of the returned copied image
------------	---

Reimplemented from [Fl_Image](#).

Reimplemented in [Fl_SVG_Image](#).

31.106.3.4 desaturate()

```
void Fl_RGB_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 before changes are applied, to avoid modifying the original image.

Reimplemented from [Fl_Image](#).

Reimplemented in [Fl_SVG_Image](#).

31.106.3.5 draw()

```
void Fl_RGB_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.

In other words: `fl_push_clip(X, Y, W, H)` is applied, the image is drawn with its upper-left corner at *X-cx, Y-cy* and its own width and height, `fl_pop_clip()` is applied.

Reimplemented from [Fl_Image](#).

Reimplemented in [Fl_SVG_Image](#).

31.106.3.6 label() [1/2]

```
void Fl_RGB_Image::label (
    Fl_Widget * widget ) [virtual]
```

The [label\(\)](#) methods are an obsolete way to set the image attribute of a widget or menu item.

Use the [image\(\)](#) or [deimage\(\)](#) methods of the [Fl_Widget](#) and [Fl_Menu_Item](#) classes instead.

Reimplemented from [Fl_Image](#).

31.106.3.7 label() [2/2]

```
void Fl_RGB_Image::label (
    Fl_Menu_Item * m ) [virtual]
```

The [label\(\)](#) methods are an obsolete way to set the image attribute of a widget or menu item.

Use the [image\(\)](#) or [deimage\(\)](#) methods of the [Fl_Widget](#) and [Fl_Menu_Item](#) classes instead.

Reimplemented from [Fl_Image](#).

31.106.3.8 max_size() [1/2]

```
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.

31.106.3.9 max_size() [2/2]

```
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\(size_t\)](#)

31.106.3.10 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](#).

31.106.3.11 uncache()

```
void Fl_RGB_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 from [Fl_Image](#).

31.106.4 Member Data Documentation

31.106.4.1 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:

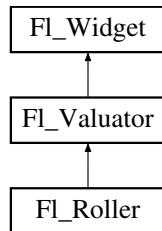
- [Fl_Image.H](#)
- [Fl_Image.cxx](#)

31.107 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)
Handles the specified event.

Protected Member Functions

- void [draw](#) ()
Draws the widget.

Additional Inherited Members

31.107.1 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 mousewheels when the mouse pointer is positioned over the roller widget.

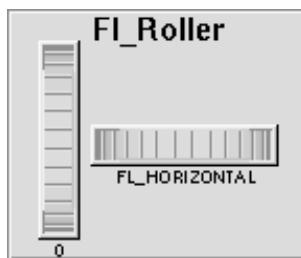


Figure 31.31 Fl_Roller

31.107.2 Constructor & Destructor Documentation

31.107.2.1 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 boxtyle is `FL_NO_BOX`.

Inherited destructor destroys the valuator.

31.107.3 Member Function Documentation

31.107.3.1 draw()

```
void Fl_Roller::draw ( ) [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](#).

31.107.3.2 handle()

```
int Fl_Roller::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.

Parameters

<code>in</code>	<code>event</code>	the kind of event received
-----------------	--------------------	----------------------------

Return values

<code>0</code>	if the event was not used or understood
<code>1</code>	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:

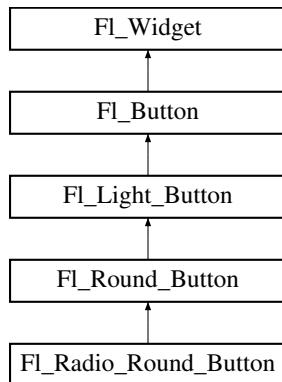
- [Fl_Roller.H](#)
- [Fl_Roller.cxx](#)

31.108 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.

Additional Inherited Members

31.108.1 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 31.32 F1_Round_Button

The `F1_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`.

31.108.2 Constructor & Destructor Documentation

31.108.2.1 F1_Round_Button()

```
F1_Round_Button::F1_Round_Button (
    int X,
    int Y,
    int W,
    int H,
    const char * L = 0 )
```

Creates a new `F1_Round_Button` widget using the given position, size, and label string.



Figure 31.33 F1_Round_Button

The `F1_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:

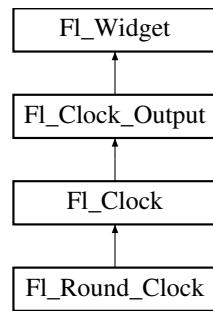
- Fl_Round_Button.H
- Fl_Round_Button.cxx

31.109 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.

Additional Inherited Members

31.109.1 Detailed Description

A clock widget of type FL_ROUND_CLOCK.

Has no box.

31.109.2 Constructor & Destructor Documentation

31.109.2.1 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 boxtyle is `FL_NO_BOX`.

This constructor 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:

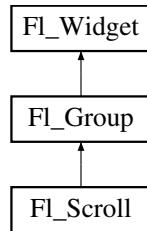
- [Fl_Round_Clock.H](#)
- [Fl_Clock.cxx](#)

31.110 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 [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...
- [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 \(int\)](#)
Handles the specified event.
- void [resize \(int X, int Y, int W, int H\)](#)
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.

Public Attributes

- [Fl_Scrollbar hscrollbar](#)
- [Fl_Scrollbar scrollbar](#)

Protected Member Functions

- void [bbox \(int &, int &, int &, int &\)](#)
Returns the bounding box for the interior of the scrolling area, inside the scrollbars.
- void [draw \(\)](#)
Draws the widget.
- void [recalc_scrollbars \(ScrollInfo &si\)](#)
Calculate visibility/size/position of scrollbars, find children's bounding box.

Additional Inherited Members

31.110.1 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:

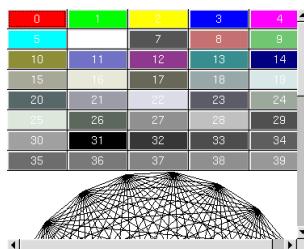


Figure 31.34 `FL_Scroll`

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 `FI_Double_Window`.

By default you can scroll in both directions, and the scrollbars disappear if the data will fit in the area of the scroll.

Use `FI_Scroll::type()` to change this as follows :

- 0 - No scrollbars
- `FI_Scroll::HORIZONTAL` - Only a horizontal scrollbar.
- `FI_Scroll::VERTICAL` - Only a vertical scrollbar.
- `FI_Scroll::BOTH` - The default is both scrollbars.
- `FI_Scroll::HORIZONTAL_ALWAYS` - Horizontal scrollbar always on, vertical always off.
- `FI_Scroll::VERTICAL_ALWAYS` - Vertical scrollbar always on, horizontal always off.
- `FI_Scroll::BOTH_ALWAYS` - Both always on.

Use `scrollbar.align(int)` (see void `FI_Widget::align(FI_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 `FI_Pack`, which is itself a group that packs its children together and changes size to surround them. Filling the `FI_Pack` with `FI_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 `FI_Scroll` without scrolling. Be sure to leave space for the scrollbars, as Fluid won't show these either.

You cannot use `FI_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.

31.110.2 Constructor & Destructor Documentation

31.110.2.1 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 boxtyle 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.

31.110.3 Member Function Documentation

31.110.3.1 bbox()

```
void Fl_Scroll::bbox (
    int & X,
    int & Y,
    int & W,
    int & H ) [protected]
```

Returns the bounding box for the interior of the scrolling area, inside the scrollbars.

Currently this is only reliable after [draw\(\)](#), and before any resizing of the [Fl_Scroll](#) or any child widgets occur.

Todo The visibility of the scrollbars ought to be checked/calculated outside of the [draw\(\)](#) method (STR #1895).

31.110.3.2 clear()

```
void Fl_Scroll::clear ( )
```

Clear all but the scrollbars...

31.110.3.3 draw()

```
void Fl_Scroll::draw ( )  [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](#).

31.110.3.4 handle()

```
int Fl_Scroll::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.

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](#).

31.110.3.5 recalc_scrollbars()

```
void Fl_Scroll::recalc_scrollbars (
    ScrollInfo & si ) [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.

Parameters

in	<i>si</i>	– ScrollInfo structure
----	-----------	--

Returns

Structure containing the calculated info.

31.110.3.6 resize()

```
void Fl_Scroll::resize (
    int X,
    int Y,
    int W,
    int H ) [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](#).

31.110.3.7 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.

31.110.3.8 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\(int\)](#)

31.110.3.9 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	<i>newSize</i>	Sets the scrollbar size in pixels. If 0 (default), scrollbar size tracks the global Fl::scrollbar_size()
----	----------------	---

See also

[Fl::scrollbar_size\(\)](#)

31.110.3.10 xposition()

```
int Fl_Scroll::xposition ( ) const [inline]
```

Gets the current horizontal scrolling position.

31.110.3.11 yposition()

```
int Fl_Scroll::yposition ( ) const [inline]
```

Gets the current vertical scrolling position.

The documentation for this class was generated from the following files:

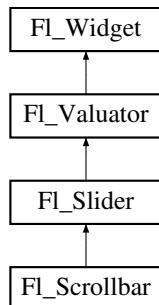
- [Fl_Scroll.H](#)
- [Fl_Scroll.cxx](#)

31.111 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

- [FI_Scrollbar](#) (int X, int Y, int W, int H, const char *L=0)
Creates a new FI_Scrollbar widget with given position, size, and label.
- int [handle](#) (int)
Handles the specified event.
- int [linesize](#) () const
Get the size of step, in lines, that the arrow 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 windowHeight, int first_line, int total_lines)
Sets the position, size and range of the slider in the scrollbar.
- [~FI_Scrollbar](#) ()
Destroys the Scrollbar.

Protected Member Functions

- void [draw](#) ()
Draws the widget.

Additional Inherited Members

31.111.1 Detailed Description

The [FI_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 [FI_Slider](#).

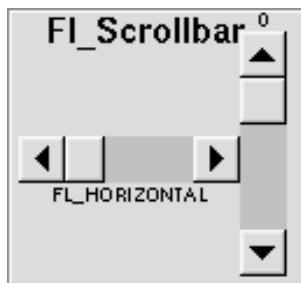


Figure 31.35 FI_Scrollbar

31.111.2 Constructor & Destructor Documentation

31.111.2.1 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.

31.111.2.2 ~Fl_Scrollbar()

```
Fl_Scrollbar::~Fl_Scrollbar ()
```

Destroys the Scrollbar.

31.111.3 Member Function Documentation

31.111.3.1 draw()

```
void Fl_Scrollbar::draw () [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](#).

31.111.3.2 handle()

```
int Fl_Scrollbar::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.

Parameters

in	<i>event</i>	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](#).

31.111.3.3 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.

31.111.3.4 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\(int pos, int size, int first, int total\)](#)

31.111.3.5 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\(int pos, int size, int first, int total\)](#)

31.111.3.6 value() [3/3]

```
int Fl_Scrollbar::value (
    int pos,
    int windowHeight,
    int first_line,
    int total_lines ) [inline]
```

Sets the position, size and range of the slider in the scrollbar.

Parameters

in	<i>pos</i>	position, first line displayed
in	<i>windowSize</i>	number of lines displayed
in	<i>first_line</i>	number of first line
in	<i>total_lines</i>	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](#)

31.112 Fl_Scroll::ScrollbarInfo::Fl_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**
- 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**

31.112.1 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:

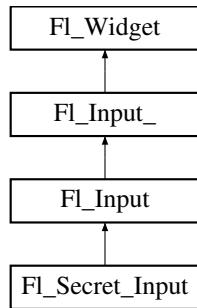
- Fl_Scroll.H

31.113 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)
Handles the specified event.

Additional Inherited Members

31.113.1 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.

31.113.2 Constructor & Destructor Documentation

31.113.2.1 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 boxtyle is FL_DOWN_BOX.

Inherited destructor destroys the widget and any value associated with it.

31.113.3 Member Function Documentation

31.113.3.1 handle()

```
int Fl_Secret_Input::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.

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:

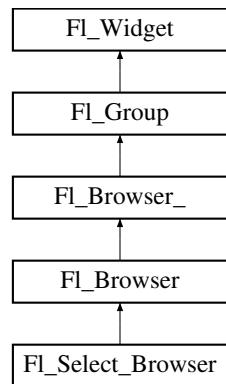
- [Fl_Secret_Input.H](#)
- [Fl_Input.cxx](#)

31.114 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.

Additional Inherited Members

31.114.1 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.

31.114.2 Constructor & Destructor Documentation

31.114.2.1 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 boxtyle 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:

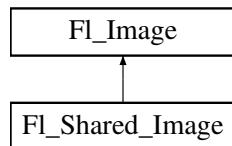
- [Fl_Select_Browser.H](#)
- [Fl_Browser.cxx](#)

31.115 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`:



Public Member Functions

- virtual [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 FLTK color value `c`.
- virtual [Fl_Image * copy \(int W, int H\)](#)
Creates a resized copy of the specified image.
- [Fl_Image * copy \(\)](#)
- virtual void [desaturate \(\)](#)
The `desaturate()` method converts an image to grayscale.
- 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.
- void [draw \(int X, int Y\)](#)
- const char * [name \(\)](#)
Returns the filename of the shared image.
- int [original \(\)](#)

- `int refcount ()`
Returns the number of references of this shared image.
- `void release ()`
Releases and possibly destroys (if refcount <= 0) a shared image.
- `void reload ()`
Reloads the shared image from disk.
- `virtual void uncache ()`
If the image has been cached for display, delete the cache data.

Static Public Member Functions

- `static void add_handler (FI_Shared_Handler f)`
Adds a shared image handler, which is basically a test function for adding new formats.
- `static FI_Shared_Image * find (const char *name, int W=0, int H=0)`
Finds a shared image from its name and size specifications.
- `static FI_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 FI_Shared_Image * get (FI_RGB_Image *rgb, int own_it=1)`
Builds a shared image from a pre-existing FI_RGB_Image.
- `static FI_Shared_Image ** images ()`
Returns the FI_Shared_Image array.*
- `static int num_images ()`
Returns the total number of shared images in the array.
- `static void remove_handler (FI_Shared_Handler f)`
Removes a shared image handler.

Protected Member Functions

- `void add ()`
Adds a shared image to the image cache.
- `FI_Shared_Image ()`
Creates an empty shared image.
- `FI_Shared_Image (const char *n, FI_Image *img=0)`
Creates a shared image from its filename and its corresponding FI_Image img.*
- `void update ()`
- `virtual ~FI_Shared_Image ()`
The destructor frees all memory and server resources that are used by the image.

Static Protected Member Functions

- `static int compare (FI_Shared_Image **i0, FI_Shared_Image **i1)`
Compares two shared images.

Protected Attributes

- `int alloc_image_`
- `FI_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**

Additional Inherited Members

31.115.1 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_Shared_Image::get\(\)](#)
- [Fl_Shared_Image::find\(\)](#)
- [Fl_Shared_Image::release\(\)](#)

31.115.2 Constructor & Destructor Documentation

31.115.2.1 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.

31.115.2.2 `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.

31.115.2.3 `~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.

31.115.3 Member Function Documentation

31.115.3.1 `add()`

```
void Fl_Shared_Image::add () [protected]
```

Adds a shared image to the image cache.

This **protected** method adds an image to the cache, an ordered list of shared images. The cache is searched for a matching image whenever one is requested, for instance with `Fl_Shared_Image::get()` or `Fl_Shared_Image::find()`.

31.115.3.2 `as_shared_image()`

```
virtual Fl_Shared_Image* Fl_Shared_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 from `Fl_Image`.

31.115.3.3 color_average()

```
void Fl_Shared_Image::color_average (
    Fl_Color c,
    float i ) [virtual]
```

The [color_average\(\)](#) method averages the colors in the image with the FLTK color value c.

The i 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 before changes are applied, to avoid modifying the original image.

Reimplemented from [Fl_Image](#).

31.115.3.4 compare()

```
int Fl_Shared_Image::compare (
    Fl_Shared_Image ** i0,
    Fl_Shared_Image ** il ) [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

A special case is considered if the width of one of the images is zero and the other image is marked `original`. In this case the images match, i.e. the comparison returns success (0).

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 usually used in two steps:

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.

Returns

Whether the images match or their relative sort order (see text).

Return values

<i>O</i>	the images match
< <i>O</i>	Image <i>i0</i> is <i>less</i> than image <i>i1</i>
> <i>O</i>	Image <i>i0</i> is <i>greater</i> than image <i>i1</i>

31.115.3.5 copy()

```
F1_Image * F1_Shared_Image::copy (
    int W,
    int H ) [virtual]
```

Creates a resized copy of the specified image.

The image should be deleted (or in the case of [F1_Shared_Image](#), released) when you are done with it.

Parameters

<i>W,H</i>	width and height of the returned copied image
------------	---

Reimplemented from [F1_Image](#).

31.115.3.6 desaturate()

```
void F1_Shared_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 before changes are applied, to avoid modifying the original image.

Reimplemented from [F1_Image](#).

31.115.3.7 draw()

```
void F1_Shared_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.

In other words: `fl_push_clip(X, Y, W, H)` is applied, the image is drawn with its upper-left corner at `X-cx, Y-cy` and its own width and height, `fl_pop_clip()` is applied.

Reimplemented from [Fl_Image](#).

31.115.3.8 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.

31.115.3.9 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

<i>name</i>	name of the image
<i>W,H</i>	desired size

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 \(const char *name_png, const unsigned char *buffer, int maxsize\)](#)

31.115.3.10 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

<i>in</i>	<i>rgb</i>	an Fl_RGB_Image used to build a new shared image.
<i>in</i>	<i>own_it</i>	1 if the shared image should delete <i>rgb</i> when it is itself deleted, 0 otherwise

Version

1.3.4

31.115.3.11 num_images()

```
int Fl_Shared_Image::num_images () [static]
```

Returns the total number of shared images in the array.

31.115.3.12 original()

```
int Fl_Shared_Image::original () [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

31.115.3.13 refcount()

```
int Fl_Shared_Image::refcount ( ) [inline]
```

Returns the number of references of this shared image.

When reference is below 1, the image is deleted.

31.115.3.14 release()

```
void Fl_Shared_Image::release ( ) [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](#).

31.115.3.15 reload()

```
void Fl_Shared_Image::reload ( )
```

Reloads the shared image from disk.

31.115.3.16 remove_handler()

```
void Fl_Shared_Image::remove_handler (
    Fl_Shared_Handler f ) [static]
```

Removes a shared image handler.

31.115.3.17 uncache()

```
void Fl_Shared_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 from [Fl_Image](#).

The documentation for this class was generated from the following files:

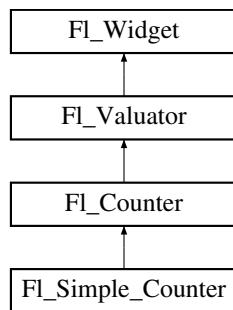
- [Fl_Shared_Image.H](#)
- [Fl_Shared_Image.cxx](#)

31.116 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 \(int X, int Y, int W, int H, const char *L=0\)](#)

Additional Inherited Members

31.116.1 Detailed Description

This widget creates a counter with only 2 arrow buttons.

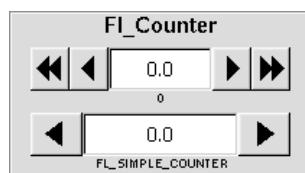


Figure 31.36 Fl_Simple_Counter

The documentation for this class was generated from the following files:

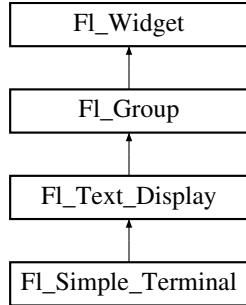
- [Fl_Simple_Counter.H](#)
- [Fl_Counter.cxx](#)

31.117 Fl_Simple_Terminal Class Reference

This is a continuous text scroll widget for logging and debugging output, much like a terminal.

```
#include <Fl_Simple_Terminal.H>
```

Inheritance diagram for Fl_Simple_Terminal:



Public Member Functions

- void [ansi](#) (bool val)

Enable/disable support of ANSI sequences like "033[31m", which sets the color/font/weight/size of any text that follows.
- bool [ansi](#) () const

Get the state of the ANSI flag which enables/disables the handling of ANSI sequences in text.
- void [append](#) (const char *s, int len=-1)

Appends new string 's' to terminal.
- void [clear](#) ()

Clears the terminal's screen and history.
- void [current_style_index](#) (int)

Set the style table index used as the current drawing color/font/weight/size for new text.
- int [current_style_index](#) () const

Get the style table index used as the current drawing color/font/weight/size for new text.
- [Fl_Simple_Terminal](#) (int X, int Y, int W, int H, const char *l=0)

Creates a new [Fl_Simple_Terminal](#) widget that can be a child of other FLTK widgets.
- void [history_lines](#) (int)

Sets the maximum number of lines for the terminal history.
- int [history_lines](#) () const

Get the maximum number of terminal history lines last set by [history_lines\(int\)](#).
- void [normal_style_index](#) (int)

Sets the style table index used by the ANSI terminal reset sequence "\033[0m", which resets the current drawing color/font/weight/size to "normal".
- int [normal_style_index](#) () const

Gets the style table index used by the ANSI terminal reset sequence "\033[0m".
- void [printf](#) (const char *fmt,...)

Appends printf formatted messages to the terminal.
- void [remove_lines](#) (int start, int count)

Remove the specified range of lines from the terminal, starting with line 'start' and removing 'count' lines.
- void [stay_at_bottom](#) (bool)

Configure the terminal to remain scrolled to the bottom when possible, chasing the end of the buffer whenever new text is added.

- `bool stay_at_bottom () const`
Gets the current value of the `stay_at_bottom(bool)` flag.
- `void style_table (FI_Text_Display::Style_Table_Entry *stable, int stable_size, int normal_style_index=0)`
Set a user defined style table, which controls the font colors, faces, weights and sizes available for the terminal's text content.
- `const FI_Text_Display::Style_Table_Entry * style_table () const`
Return the current style table being used.
- `int style_table_size () const`
Return the current style table's size (in bytes).
- `void text (const char *s, int len=-1)`
Replaces the terminal with new text content in string 's'.
- `const char * text () const`
Returns entire text content of the terminal as a single string.
- `void vprintf (const char *fmt, va_list ap)`
Appends printf formatted messages to the terminal.
- `~FI_Simple_Terminal ()`
Destructor for this widget; removes any internal allocations for the terminal, including text buffer, style buffer, etc.

Protected Member Functions

- `virtual void draw ()`
Draws the widget, including a cursor at the end of the buffer.
- `void enforce_history_lines ()`
Enforce 'history_lines' limit on the history buffer by trimming off lines from the top of the buffer.
- `void enforce_stay_at_bottom ()`
Scroll to last line unless someone has manually scrolled the vertical scrollbar away from the bottom.
- `void vscroll_cb2 (FI_Widget *, void *)`

Static Protected Member Functions

- `static void vs(scroll_cb (FI_Widget *, void *)`

Protected Attributes

- `FI_Text_Buffer * buf`
- `FI_Text_Buffer * sbuf`

Additional Inherited Members

31.117.1 Detailed Description

This is a continuous text scroll widget for logging and debugging output, much like a terminal.

Includes `printf()` for appending messages, a line limit for the screen history size, ANSI sequences to control text color, font face, font weight and font size.

This is useful in place of using `stdout/stderr` for logging messages when no terminal is available, such as when an application is invoked from a desktop shortcut, dock, or file browser.

Like a regular console terminal, the vertical scrollbar 'tracks' the bottom of the buffer as new output is added. If the user scrolls away from the bottom, this 'tracking' feature is temporarily suspended, so the user can browse the terminal history without fighting the scrollbar when new text is added asynchronously. When the user returns the scroller to the bottom of the display, the scrollbar's tracking resumes.

Features include:

- `history_lines(int)` can define a maximum size for the terminal screen history
- `stay_at_bottom(bool)` can be used to cause the terminal to keep scrolled to the bottom
- `ansi(bool)` enables ANSI sequences within the text to control text colors
- `style_table()` can be used to define custom color/font/weight/size combinations

What this widget is NOT is a full terminal emulator; it does NOT handle stdio redirection, pipes, pseudo ttys, termio character cooking, keyboard input processing, screen addressing, random cursor positioning, curses(3) compatibility, or VT100/xterm emulation.

It is a simple text display widget that leverages the features of the [Fl_Text_Display](#) base class to handle terminal-like behavior, such as logging events or debug information.

Example use:

```
#include <FL/Fl_Simple_Terminal.H>
:
tty = new Fl_Simple_Terminal(...);
tty->ansi(true);           // enable use of "\033[#m"
:
tty->printf("The time is now: \033[32m%s\033[0m", date_time_str);
```

Example application:

```
#include <time.h>           //START
#include <FL/Fl_Double_Window.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Simple_Terminal.H>

#define TERMINAL_HEIGHT 120

// Globals
Fl_Double_Window *G_win = 0;
Fl_Box *G_box = 0;
Fl_Simple_Terminal *G_tty = 0;

// Append a date/time message to the terminal every 2 seconds
void tick_cb(void *data) {
    time_t lt = time(NULL);
    G_tty->printf("Timer tick: \033[32m%s\033[0m\n", ctime(&lt));
    Fl::repeat_timeout(2.0, tick_cb, data);
}

int main(int argc, char **argv) {
    G_win = new Fl_Double_Window(500, 200+TERMINAL_HEIGHT, "Your App");
    G_win->begin();

    G_box = new Fl_Box(0, 0, G_win->w(), 200,
                      "Your app GUI in this area.\n\n"
                      "Your app's debugging output in tty below");

    // Add simple terminal to bottom of app window for scrolling history of status messages.
    G_tty = new Fl_Simple_Terminal(0,200,G_win->w(),TERMINAL_HEIGHT);
    G_tty->ansi(true); // enable use of "\033[32m"

    G_win->end();
    G_win->resizable(G_win);
    G_win->show();
    Fl::add_timeout(0.5, tick_cb);
    return Fl::run();
}
```

Style Tables For Color/Font/Fontsize Control

Internally this widget derives from [Fl_Text_Display](#), and therefore inherits some of its idiosyncracies. In particular, when colors are used, the base class's concept of a 'style table' is used.

The 'style table' is similar to a color mapped image; where each pixel is a single value that is an index into a table of colors to minimize per-pixel memory use.

The style table has a similar goal; since every character in the terminal can potentially be a different color, instead of managing several integer attribute values per-character, a single character for each character is used as an index into the style table, choosing one of the available color/font/weight/size values available. This saves on as much as 3 to 4 times the memory use, useful when there's a large amount of text.

When [ansi\(\)](#) is set to 'true', ANSI sequences of the form "\033[#m" can be used to select different colors, font faces, font weights (bold,italic..), and font sizes, where '#' is the index number into the style table. Example:

```
"\033[0mThis text uses the 1st entry in the style table\n"
"\033[1mThis text uses the 2nd entry in the style table\n"
"\033[2mThis text uses the 3rd entry in the style table\n"
etc..
```

There is a built-in style table that provides some commonly used ANSI colors for "\033[30m" through "\033[37m" (blk,red,grn,yel,blu,mag,cyn,wht), and a brighter version of those colors for "\033[40" through "\033[47m". See [ansi\(bool\)](#) for more info.

You can also supply a custom style table using [style_table\(Style_Table_Entry*,int,int\)](#), allowing you to define your own color/font/weight/size combinations. See that method's docs for more info.

All style index numbers are rounded to the size of the style table (via modulus) to protect the style array from overruns.

31.117.2 Member Function Documentation

31.117.2.1 [ansi\(\)](#) [1/2]

```
void Fl_Simple_Terminal::ansi (
    bool val )
```

Enable/disable support of ANSI sequences like "\033[31m", which sets the color/font/weight/size of any text that follows.

If enabled, ANSI sequences of the form "\033[#m" can be used to change font color, face, and size, where '#' is an index number into the current style table. These "escape sequences" are hidden from view.

If disabled, the [textcolor\(\)](#) / [textfont\(\)](#) / [textsize\(\)](#) methods define the color and font for all text in the terminal. ANSI sequences are not handled specially, and rendered as raw text.

A built-in style table is provided, but you can configure a custom style table using [style_table\(Style_Table_Entry*,int,int\)](#) for your own colors and fonts.

The built-in style table supports these ANSI sequences:

ANSI Sequence	Color Name	Font Face + Size	Remarks
"\033[0m"	"Normal"	FL_COURIER, 14	Resets to default color/font/weight/size
"\033[30m"	Medium Black	FL_COURIER, 14	
"\033[31m"	Medium Red	FL_COURIER, 14	
"\033[32m"	Medium Green	FL_COURIER, 14	
"\033[33m"	Medium Yellow	FL_COURIER, 14	
"\033[34m"	Medium Blue	FL_COURIER, 14	
"\033[35m"	Medium Magenta	FL_COURIER, 14	
"\033[36m"	Medium Cyan	FL_COURIER, 14	
"\033[37m"	Medium White	FL_COURIER, 14	The color when "\033[0m" reset is used
"\033[40m"	Bright Black	FL_COURIER, 14	
"\033[41m"	Bright Red	FL_COURIER, 14	
"\033[42m"	Bright Green	FL_COURIER, 14	
"\033[43m"	Bright Yellow	FL_COURIER, 14	
"\033[44m"	Bright Blue	FL_COURIER, 14	
"\033[45m"	Bright Magenta	FL_COURIER, 14	
"\033[46m"	Bright Cyan	FL_COURIER, 14	
"\033[47m"	Bright White	FL_COURIER, 14	

Here's example code demonstrating the use of ANSI codes to select the built-in colors, and how it looks in the terminal:



Figure 31.37 Fl_Simple_Terminal built-in ANSI sequences

Note

Changing the [ansi\(bool\)](#) value clears the buffer and forces a [redraw\(\)](#).

Enabling ANSI mode overrides [textfont\(\)](#), [textsize\(\)](#), [textcolor\(\)](#) completely, which are controlled instead by [current_style_index\(\)](#) and the current [style_table\(\)](#).

See also

[style_table\(Style_Table_Entry*,int,int\)](#), [current_style_index\(\)](#), [normal_style_index\(\)](#)

31.117.2.2 ansi() [2/2]

```
bool Fl_Simple_Terminal::ansi ( ) const
```

Get the state of the ANSI flag which enables/disables the handling of ANSI sequences in text.

When true, ANSI sequences in the text stream control color, font and font sizes of text (e.g. "\033[41mThis is Red\033[0m"). For more info, see [ansi\(bool\)](#).

See also

[ansi\(bool\)](#)

31.117.2.3 append()

```
void Fl_Simple_Terminal::append (
    const char * s,
    int len = -1 )
```

Appends new string 's' to terminal.

The string can contain UTF-8, crlf's, and ANSI sequences are also supported when [ansi\(bool\)](#) is set to 'true'.

Parameters

<i>s</i>	string to append.
<i>len</i>	optional length of string can be specified if known to save the internals from having to call strlen()

See also

[printf\(\)](#), [vprintf\(\)](#), [text\(\)](#), [clear\(\)](#)

31.117.2.4 clear()

```
void Fl_Simple_Terminal::clear ( )
```

Clears the terminal's screen and history.

Cursor moves to top of window.

31.117.2.5 current_style_index() [1/2]

```
void Fl_Simple_Terminal::current_style_index (
    int val )
```

Set the style table index used as the current drawing color/font/weight/size for new text.

For example:

```
:
tty->ansi(true);
tty->append("Some normal text.\n");
tty->current_style_index(2); // same as "\033[2m"
tty->append("This text will be green.\n");
tty->current_style_index(tty->normal_style_index()); // same as "\033[0m"
tty->append("Back to normal text.\n");
:
```

This value can also be changed by an ANSI sequence like "\033[#m", where # would be a new style index value. So if the application executes: `term->append("\033[4mTesting")`, then [current_style_index\(\)](#) will be left set to 4.

The index number specified should be within the number of items in the current style table. Values larger than the table will be clamped to the size of the table with a modulus operation.

Effective only when [ansi\(bool\)](#) is 'true'.

31.117.2.6 current_style_index() [2/2]

```
int Fl_Simple_Terminal::current_style_index ( ) const
```

Get the style table index used as the current drawing color/font/weight/size for new text.

This value is also controlled by the ANSI sequence "\033[#m", where # would be a new style index value. So if the application executes: term->append ("\033[4mTesting"), then [current_style_index\(\)](#) returns 4.

See also

[current_style_index\(int\)](#)

31.117.2.7 draw()

```
void Fl_Simple_Terminal::draw (
    void ) [protected], [virtual]
```

Draws the widget, including a cursor at the end of the buffer.

This is needed since currently [Fl_Text_Display](#) doesn't provide a reliable way to always do this.

Reimplemented from [Fl_Text_Display](#).

31.117.2.8 enforce_history_lines()

```
void Fl_Simple_Terminal::enforce_history_lines ( ) [protected]
```

Enforce 'history_lines' limit on the history buffer by trimming off lines from the top of the buffer.

This is a protected member called automatically by the public API functions. Only internal methods or subclasses adjusting the internal buffer directly should need to call this.

31.117.2.9 enforce_stay_at_bottom()

```
void Fl_Simple_Terminal::enforce_stay_at_bottom ( ) [protected]
```

Scroll to last line unless someone has manually scrolled the vertical scrollbar away from the bottom.

This is a protected member called automatically by the public API functions. Only internal methods or subclasses adjusting the internal buffer directly should need to call this.

31.117.2.10 history_lines() [1/2]

```
void Fl_Simple_Terminal::history_lines (
    int maxlines )
```

Sets the maximum number of lines for the terminal history.

The new limit value is automatically enforced on the current screen history, truncating off any lines that exceed the new limit.

When a limit is set, the buffer is trimmed as new text is appended, ensuring the buffer never displays more than the specified number of lines.

The default maximum is 500 lines.

Parameters

<code>maxlines</code>	Maximum number of lines kept on the terminal buffer history. Use -1 for an unlimited scroll history. A value of 0 is not recommended.
-----------------------	--

31.117.2.11 `history_lines()` [2/2]

```
int Fl_Simple_Terminal::history_lines ( ) const
```

Get the maximum number of terminal history lines last set by [history_lines\(int\)](#).

-1 indicates an unlimited scroll history.

See also

[history_lines\(int\)](#)

31.117.2.12 `normal_style_index()` [1/2]

```
void Fl_Simple_Terminal::normal_style_index (
    int val )
```

Sets the style table index used by the ANSI terminal reset sequence "\033[0m", which resets the current drawing color/font/weight/size to "normal".

Effective only when [ansi\(bool\)](#) is 'true'.

See also

[ansi\(bool\)](#), [style_table\(Style_Table_Entry*,int,int\)](#)

Note

Changing this value does *not* change the current drawing color. To change that, use [current_style_index\(int\)](#).

31.117.2.13 `normal_style_index()` [2/2]

```
int Fl_Simple_Terminal::normal_style_index ( ) const
```

Gets the style table index used by the ANSI terminal reset sequence "\033[0m".

This is the value last set by [normal_style_index\(int\)](#), or as set by the 3rd argument to [style_table\(Style_Table_Entry*,int,int\)](#).

See also

[normal_style_index\(int\)](#), [ansi\(bool\)](#), [style_table\(Style_Table_Entry*,int,int\)](#)

31.117.2.14 printf()

```
void Fl_Simple_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 when [ansi\(bool\)](#) is set to 'true'.

Example:

```
#include <FL/Fl_Simple_Terminal.H>
int main(..) {
:
// Create a simple terminal, and append some messages to it
Fl_Simple_Terminal *tty = new Fl_Simple_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

See [Fl_Text_Buffer::vprintf\(\)](#) for limitations.

Parameters

in	<i>fmt</i>	is a printf format string for the message text.
----	------------	---

31.117.2.15 remove_lines()

```
void Fl_Simple_Terminal::remove_lines (
    int start,
    int count )
```

Remove the specified range of lines from the terminal, starting with line 'start' and removing 'count' lines.

This method is used to enforce the history limit.

Parameters

<i>start</i>	– starting line to remove
<i>count</i>	– number of lines to remove

31.117.2.16 stay_at_bottom() [1/2]

```
void Fl_Simple_Terminal::stay_at_bottom (
```

```
    bool val )
```

Configure the terminal to remain scrolled to the bottom when possible, chasing the end of the buffer whenever new text is added.

If disabled, the terminal behaves more like a text display widget; the scrollbar does not chase the bottom of the buffer.

If the user scrolls away from the bottom, this 'chasing' feature is temporarily disabled. This prevents the user from having to fight the scrollbar chasing the end of the buffer while browsing when new text is also being added asynchronously. When the user returns the scroller to the bottom of the display, the chasing behavior resumes.

The default is 'true'.

31.117.2.17 stay_at_bottom() [2/2]

```
bool Fl_Simple_Terminal::stay_at_bottom ( ) const
```

Gets the current value of the [stay_at_bottom\(bool\)](#) flag.

When true, the terminal tries to keep the scrollbar scrolled to the bottom when new text is added.

See also

[stay_at_bottom\(bool\)](#)

31.117.2.18 style_table() [1/2]

```
void Fl_Simple_Terminal::style_table (
    Fl_Text_Display::Style_Table_Entry * stable,
    int stable_size,
    int normal_style_index = 0 )
```

Set a user defined style table, which controls the font colors, faces, weights and sizes available for the terminal's text content.

[ansi\(bool\)](#) must be set to 'true' for the defined style table to be used at all.

If 'stable' is NULL, then the "built in" style table is used. For info about the built-in colors, see [ansi\(bool\)](#).

Which style table entry used for drawing depends on the value last set by [current_style_index\(\)](#), or by the ANSI sequence "\033[#m", where '#' is the index into the style table array, the index limited to the size of the array via modulus.

If the index# passed via "\033[#m" is larger than the number of elements in the table, the value is clamped via modulus. So for a 10 element table, the following ANSI codes would all be equivalent, selecting the 5th element in the table: "\033[5m", "\033[15m", "\033[25m", etc. This is because 5==(15%10)==(25%10), etc.

A special exception is made for "\033[0m", which is supposed to "reset" the current style table to default color/font/weight/size, as last set by [normal_style_index](#), or by the API method [normal_style_index\(int\)](#).

In cases like the built-in style table, where the 17th item is the "normal" color, the 'normal_style_index' is set to 17 so that "\033[0m" resets to that color, instead of the first element in the table.

If you want "\033[0m" to simply pick the first element in the table, then set 'normal_style_index' to 0.

An example of defining a custom style table (white courier 14, red courier 14, and white helvetica 14):

```

int main() {
:
// Our custom style table
Fl_Text_Display::Style_Table_Entry mystyle[] = {
// Font Color Font Face      Font Size     Index  ANSI Sequence
// ----- ----- ----- -----
{ FL_WHITE,    FL_COURIER_BOLD, 14 },      // 0      "\033[0m"  ("default")
{ FL_RED,      FL_COURIER_BOLD, 14 },      // 1      "\033[1m"
{ FL_WHITE,    FL_HELVETICA,    14 }       // 2      "\033[2m"
};
// Create terminal, enable ANSI and our style table
tty = new Fl_Simple_Terminal(...);
tty->ansi(true);                         // enable ANSI codes
tty->style_table(&mystyle[0], sizeof(mystyle), 0); // use our custom style table
:
// Now write to terminal, with ANSI that uses our style table
tty->printf("\033[0mNormal Text\033[1mRed Courier Text\n");
tty->append("\033[2mWhite Helvetica\033[0mBack to normal.\n");
:

```

Note

Changing the style table [clear\(\)](#)s the terminal.

You currently can't control /background/ color of text, a limitation of [Fl_Text_Display](#)'s current implementation.
The caller is responsible for managing the memory of the style table.

Until STR#3412 is repaired, [Fl_Text_Display](#) has scrolling bug if the style table's font size != [textsize\(\)](#)

Parameters

<i>stable</i>	- the style table, an array of structs of the type Fl_Text_Display::Style_Table_Entry . Can be NULL to use the default style table (see ansi(bool)).
<i>stable_size</i>	- the sizeof() the style table (in bytes). Set this to 0 if 'stable' is NULL.
<i>normal_style_index</i>	- the style table index# used when the special ANSI sequence "\033[0m" is encountered. Normally use 0 so that sequence selects the first item in the table. Only use different values if a different entry in the table should be the default. This value should not be larger than the number of items in the table, or it will be clamped with a modulus operation. This value is ignored if stable is NULL.

31.117.2.19 style_table() [2/2]

```
const Fl_Text_Display::Style_Table_Entry * Fl_Simple_Terminal::style_table( ) const
```

Return the current style table being used.

This is the value last passed as the 1st argument to [style_table\(Style_Table_Entry*,int,int\)](#). If no style table was defined, the built-in style table is returned.

[ansi\(bool\)](#) must be set to 'true' for the style table to be used at all.

See also

[style_table\(Style_Table_Entry*,int,int\)](#)

31.117.2.20 style_table_size()

```
int Fl_Simple_Terminal::style_table_size ( ) const
```

Return the current style table's size (in bytes).

This is the value last passed as the 2nd argument to [style_table\(Style_Table_Entry*,int,int\)](#).

31.117.2.21 text() [1/2]

```
void Fl_Simple_Terminal::text (
    const char * s,
    int len = -1 )
```

Replaces the terminal with new text content in string 's'.

The string can contain UTF-8, crlf's, and ANSI sequences are also supported when [ansi\(bool\)](#) is set to 'true'.

Old terminal content is completely cleared.

Parameters

<i>s</i>	string to append.
<i>len</i>	optional length of string can be specified if known to save the internals from having to call strlen()

See also

[append\(\)](#), [printf\(\)](#), [vprintf\(\)](#), [clear\(\)](#)

31.117.2.22 text() [2/2]

```
const char * Fl_Simple_Terminal::text ( ) const
```

Returns entire text content of the terminal as a single string.

This includes the screen history, as well as the visible onscreen content.

31.117.2.23 vprintf()

```
void Fl_Simple_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.

Parameters

<i>fmt</i>	is a printf format string for the message text.
<i>ap</i>	is a va_list created by va_start() and closed with va_end(), which the caller is responsible for handling.

The documentation for this class was generated from the following files:

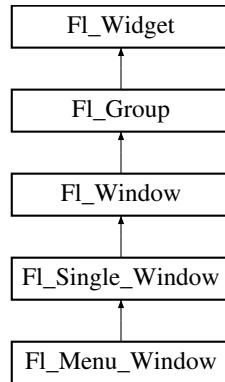
- Fl_Simple_Terminal.H
- Fl_Simple_Terminal.cxx

31.118 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.
- int [make_current](#) ()

Puts the window on the screen.
- void [show](#) ()

Puts the window on the screen.
- void [show](#) (int a, char **b)

Additional Inherited Members

31.118.1 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.

31.118.2 Member Function Documentation

31.118.2.1 show()

```
void Fl_Single_Window::show ( ) [virtual]
```

Puts the window on the screen.

Usually (on X) this has the side effect of opening the display.

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_Window](#).

The documentation for this class was generated from the following files:

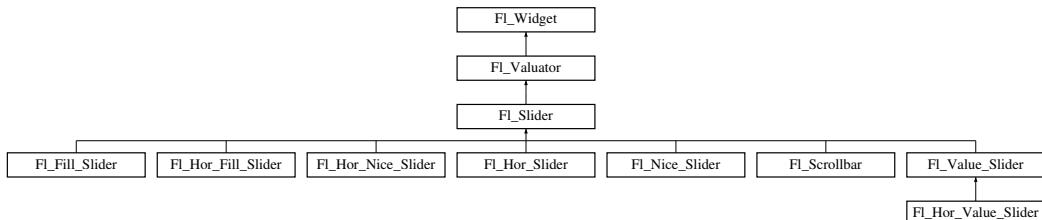
- [Fl_Single_Window.H](#)
- [Fl_Single_Window.cxx](#)

31.119 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 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)
Handles the specified event.
- 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.

Protected Member Functions

- void **draw** (int, int, int, int)
- void **draw ()**
Draws the widget.
- int **handle** (int, int, int, int, int)

Additional Inherited Members

31.119.1 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.

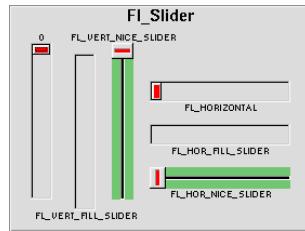


Figure 31.38 Fl_Slider

31.119.2 Constructor & Destructor Documentation

31.119.2.1 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 boxtyle is `FL_DOWN_BOX`.

31.119.3 Member Function Documentation

31.119.3.1 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.

31.119.3.2 draw()

```
void Fl_Slider::draw ( )  [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](#).

31.119.3.3 handle()

```
int Fl_Slider::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.

Parameters

in	<code>event</code>	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](#).

31.119.3.4 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	<i>pos</i>	position of first line displayed
in	<i>size</i>	size of window in lines
in	<i>first</i>	number of first line
in	<i>total</i>	total number of lines Returns Fl_Valuator::value(p)

31.119.3.5 slider() [1/2]

```
Fl_Boxtype Fl_Slider::slider ( ) const [inline]
```

Gets the slider box type.

31.119.3.6 slider() [2/2]

```
void Fl_Slider::slider (
    Fl_Boxtype c ) [inline]
```

Sets the slider box type.

31.119.3.7 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.

The documentation for this class was generated from the following files:

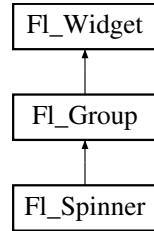
- [Fl_Slider.H](#)
- [Fl_Slider.cxx](#)

31.120 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

- void [color \(Fl_Color v\)](#)
Sets the background color of the spinner widget's input field.
- [Fl_Color color \(\) const](#)
Returns 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 \(int event\)](#)
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.
- void [maximum_size \(int m\)](#)
Sets the maximum width of the input field.
- int [maximum_size \(\) const](#)
Returns 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 \(int X, int Y, int W, int H\)](#)

- **`void selection_color (Fl_Color val)`**

Resizes the `Fl_Group` widget and all of its children.
- **`Fl_Color selection_color () const`**

Sets the selection color of the spinner widget's input field.
- **`void step (double s)`**

Returns the selection color of the spinner widget's input field.
- **`double step () const`**

Sets or returns the amount to change the value when the user clicks a button.
- **`Fl_Color textcolor () const`**

Gets the amount to change the value when the user clicks a button.
- **`void textcolor (Fl_Color c)`**

Gets the color of the text in the input field.
- **`Fl_Font textfont () const`**

Sets the color of the text in the input field.
- **`void textfont (Fl_Font f)`**

Gets the font of the text in the input field.
- **`void fontsize (Fl_Fontsize s)`**

Sets the font of the text in the input field.
- **`uchar type (uchar v)`**

Sets the size of the text in the input field.
- **`uchar type () const`**

Gets the size of the text in the input field.
- **`double value () const`**

Sets the current value of the widget.
- **`void value (double v)`**

Gets the current value of the input widget.
- **`void wrap (int set)`**

Sets whether the spinner wraps around at upper and lower bounds.
- **`int wrap () const`**

Gets the wrap mode of the `Fl_Spinner` widget.

Protected Attributes

- **`Fl_Repeat_Button down_button_`**
- **`Fl_Spinner_Input input_`**
- **`Fl_Repeat_Button up_button_`**

Additional Inherited Members

31.120.1 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.

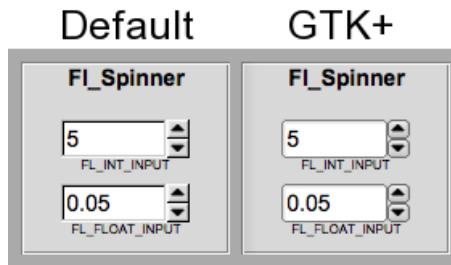


Figure 31.39 Fl_Spinner widget

31.120.2 Constructor & Destructor Documentation

31.120.2.1 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.

31.120.3 Member Function Documentation

31.120.3.1 format() [1/2]

```
const char* Fl_Spinner::format ( ) const [inline]
```

Returns the format string for the value.

31.120.3.2 format() [2/2]

```
void Fl_Spinner::format (
    const char * f ) [inline]
```

Sets the format string for the value.

31.120.3.3 handle()

```
int Fl_Spinner::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.

Parameters

in	<i>event</i>	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](#).

31.120.3.4 maximum() [1/2]

```
double Fl_Spinner::maximum ( ) const [inline]
```

Gets the maximum value of the widget.

31.120.3.5 maximum() [2/2]

```
void Fl_Spinner::maximum (
    double m ) [inline]
```

Sets the maximum value of the widget.

31.120.3.6 minimum() [1/2]

```
double Fl_Spinner::minimum ( ) const [inline]
```

Gets the minimum value of the widget.

31.120.3.7 minimum() [2/2]

```
void Fl_Spinner::minimum (
    double m ) [inline]
```

Sets the minimum value of the widget.

31.120.3.8 range()

```
void Fl_Spinner::range (
    double a,
    double b ) [inline]
```

Sets the minimum and maximum values for the widget.

31.120.3.9 resize()

```
void Fl_Spinner::resize (
    int X,
    int Y,
    int W,
    int H ) [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](#).

31.120.3.10 step() [1/2]

```
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 [Fl_Spinner::step\(\)](#) const

31.120.3.11 step() [2/2]

```
double Fl_Spinner::step ( ) const [inline]
```

Gets the amount to change the value when the user clicks a button.

See also

[Fl_Spinner::step\(double\)](#)

31.120.3.12 textcolor() [1/2]

```
Fl_Color Fl_Spinner::textcolor ( ) const [inline]
```

Gets the color of the text in the input field.

31.120.3.13 textcolor() [2/2]

```
void Fl_Spinner::textcolor (
    Fl_Color c ) [inline]
```

Sets the color of the text in the input field.

31.120.3.14 textfont() [1/2]

```
Fl_Font Fl_Spinner::textfont ( ) const [inline]
```

Gets the font of the text in the input field.

31.120.3.15 textfont() [2/2]

```
void Fl_Spinner::textfont (
    Fl_Font f ) [inline]
```

Sets the font of the text in the input field.

31.120.3.16 textszie() [1/2]

```
Fl_Fontsize Fl_Spinner::textszie ( ) const [inline]
```

Gets the size of the text in the input field.

31.120.3.17 textszie() [2/2]

```
void Fl_Spinner::textszie (
    Fl_Fontsize s ) [inline]
```

Sets the size of the text in the input field.

31.120.3.18 type() [1/2]

```
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.

31.120.3.19 type() [2/2]

```
uchar Fl_Spinner::type ( ) const [inline]
```

Gets the numeric representation in the input field.

See also

[Fl_Spinner::type\(uchar\)](#)

31.120.3.20 value() [1/2]

```
double Fl_Spinner::value ( ) const [inline]
```

Gets the current value of the widget.

31.120.3.21 value() [2/2]

```
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.

31.120.3.22 wrap() [1/2]

```
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

31.120.3.23 wrap() [2/2]

```
int Fl_Spinner::wrap ( ) const [inline]
```

Gets the wrap mode of the [Fl_Spinner](#) widget.

See also

void [wrap\(int\)](#)

Since

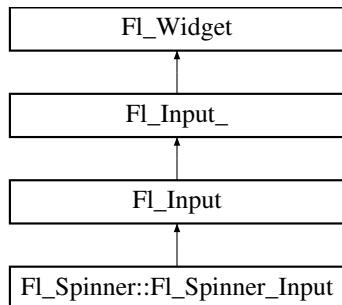
1.4.0

The documentation for this class was generated from the following files:

- [Fl_Spinner.H](#)
- [Fl_Spinner.cxx](#)

31.121 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)

Handles events of [Fl_Spinner](#)'s embedded input widget.

Additional Inherited Members

31.121.1 Member Function Documentation

31.121.1.1 handle()

```
int Fl_Spinner::Fl_Spinner_Input::handle (
    int event ) [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:

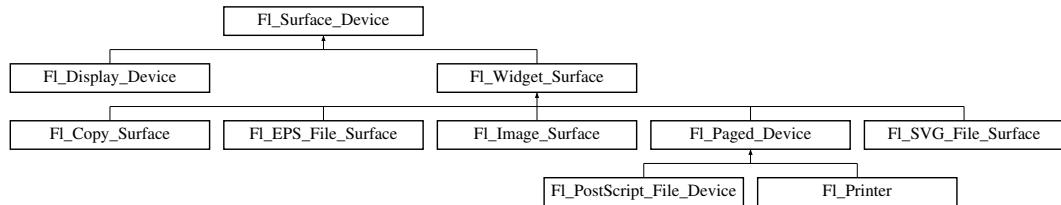
- [Fl_Spinner.H](#)
- [Fl_Spinner.cxx](#)

31.122 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** ()
- **Fl_Surface_Device** (Fl_Graphics_Driver *graphics_driver)
Constructor that sets the graphics driver to use for the created surface.

31.122.1 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 a particular [Fl_Surface_Device](#) derived class (e.g., [Fl_Copy_Surface](#), [Fl_Printer](#)).
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 class [Fl_Printer](#) because it is done by [Fl_Printer::begin_job\(\)](#)).
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. Drawing surfaces from [Fl_Widget_Surface](#) derived classes allow additional ways to draw to them (e.g., [Fl_Printer::print_widget\(\)](#), [Fl_Image_Surface::draw\(\)](#)).
4. After all drawing requests have been performed, redirect graphics requests back to their previous destination with [Fl_Surface_Device::pop_current\(\)](#);
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.

Class [Fl_Surface_Device](#) can also be derived to define new kinds of graphical output usable with FLTK drawing functions. An example would be to draw to a PDF file. This would require to create a new class, say [PDF_File_Surface](#), derived from class [Fl_Surface_Device](#), and another new class, say [PDF_Graphics_Driver](#), derived from class [Fl_Graphics_Driver](#). Class [PDF_Graphics_Driver](#) should implement all virtual methods of the [Fl_Graphics_Driver](#) class to support all FLTK drawing functions and have them draw into PDF files. Alternatively, class [PDF_Graphics_Driver](#) could implement only some virtual methods, and only part of the FLTK drawing API would be usable when drawing to PDF files.

31.122.2 Constructor & Destructor Documentation

31.122.2.1 Fl_Surface_Device()

```
Fl_Surface_Device::Fl_Surface_Device (
    Fl_Graphics_Driver * graphics_driver ) [inline], [protected]
```

Constructor that sets the graphics driver to use for the created surface.

31.122.2.2 ~Fl_Surface_Device()

```
Fl_Surface_Device::~Fl_Surface_Device ( ) [virtual]
```

The destructor.

31.122.3 Member Function Documentation

31.122.3.1 driver() [1/2]

```
void Fl_Surface_Device::driver (
    Fl_Graphics_Driver * graphics_driver ) [inline], [protected]
```

Sets the graphics driver of this drawing surface.

31.122.3.2 driver() [2/2]

```
Fl_Graphics_Driver* Fl_Surface_Device::driver ( ) [inline]
```

Returns the graphics driver of this drawing surface.

31.122.3.3 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.

Version

1.4.0

31.122.3.4 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.

Version

1.4.0

31.122.3.5 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. Starting from FLTK 1.4.0, another convenient API to set/unset the current drawing surface is [Fl_Surface_Device::push_current\(\)](#) / [Fl_Surface_Device::pop_current\(\)](#).

Reimplemented in [Fl_Printer](#), [Fl_Image_Surface](#), and [Fl_Copy_Surface](#).

31.122.3.6 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

The documentation for this class was generated from the following files:

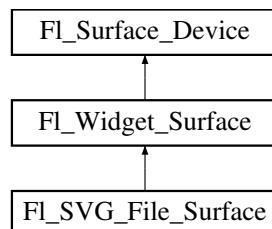
- [Fl_Device.H](#)
- [Fl_Device.cxx](#)

31.123 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 with function fclose() the FILE pointer where SVG data is output.
- FILE * [file \(\)](#)
Returns the underlying FILE pointer.
- [Fl_SVG_File_Surface \(int width, int height, FILE *svg\)](#)
Constructor of the SVG drawing surface.
- virtual void [origin \(int x, int y\)](#)
Sets the position of the origin of graphics in the drawable part of the drawing surface.
- 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.
- [~Fl_SVG_File_Surface \(\)](#)
Destructor.

Additional Inherited Members

31.123.1 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.

31.123.2 Constructor & Destructor Documentation

31.123.2.1 Fl_SVG_File_Surface()

```
Fl_SVG_File_Surface::Fl_SVG_File_Surface (
    int width,
    int height,
    FILE * svg )
```

Constructor of the SVG drawing surface.

Parameters

<i>width,height</i>	Width and height of the graphics area in FLTK drawing units
<i>svg</i>	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() .

31.123.2.2 ~Fl_SVG_File_Surface()

```
Fl_SVG_File_Surface::~Fl_SVG_File_Surface ( )
```

Destructor.

The underlying FILE pointer remains open after destruction of the [Fl_SVG_File_Surface](#) object unless [close\(\)](#) was called.

31.123.3 Member Function Documentation**31.123.3.1 close()**

```
int Fl_SVG_File_Surface::close ( )
```

Closes with function fclose() the FILE pointer where SVG data is output.

The only operation possible after this on the [Fl_SVG_File_Surface](#) object is its destruction.

Returns

The value returned by fclose().

31.123.3.2 origin()

```
virtual void Fl_SVG_File_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 from [Fl_Widget_Surface](#).

31.123.3.3 printable_rect()

```
virtual int Fl_SVG_File_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 from [Fl_Widget_Surface](#).

31.123.3.4 translate()

```
virtual void Fl_SVG_File_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 from [Fl_Widget_Surface](#).

The documentation for this class was generated from the following file:

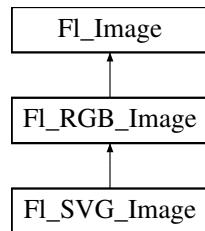
- Fl_SVG_File_Surface.H

31.124 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

- virtual [Fl_SVG_Image * as_svg_image \(\)](#)
Returns whether an image is an [Fl_SVG_Image](#) or not.
- virtual void [color_average \(Fl_Color c, float i\)](#)
The [color_average\(\)](#) method averages the colors in the image with the FLTK color value `c`.
- virtual [Fl_Image * copy \(int W, int H\)](#)
Creates a resized copy of the specified image.
- [Fl_Image * copy \(\)](#)
- virtual void [desaturate \(\)](#)
The [desaturate\(\)](#) method converts an image to grayscale.
- 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.
- void [draw \(int X, int Y\)](#)
- [Fl_SVG_Image \(const char *filename, const char *svg_data=NULL\)](#)
The constructor loads the SVG image from the given .svg/.svgz filename or in-memory data.
- virtual void [normalize \(\)](#)
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.
- virtual [~Fl_SVG_Image \(\)](#)
The destructor frees all memory and server resources that are used by the SVG image.

Public Attributes

- bool [proportional](#)
Set this to `false` to allow image re-scaling that alters the image aspect ratio.

Additional Inherited Members

31.124.1 Detailed Description

The [FI_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 image ratio while performing rasterization.

Use [FI_Image::fail\(\)](#) to check if the [FI_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.

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 [FI_SVG_Image](#) is always rasterized to the exact screen resolution at which it is drawn.

The [FI_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 [FI_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();
}
```

31.124.2 Constructor & Destructor Documentation

31.124.2.1 [Fl_SVG_Image\(\)](#)

```
Fl_SVG_Image::Fl_SVG_Image (
    const char * filename,
    const char * svg_data = NULL )
```

The constructor loads the SVG image from the given .svg/.svgz filename or in-memory data.

Parameters

<i>filename</i>	Name of a .svg or .svgz file, or NULL.
<i>svg_data</i>	A pointer to the memory location of the SVG image data. This parameter allows to load an SVG image from in-memory data, and is used when <i>filename</i> is NULL.

Note

In-memory SVG data is parsed by the object constructor and is not used after construction.

31.124.2.2 ~Fl_SVG_Image()

```
Fl_SVG_Image::~Fl_SVG_Image ( ) [virtual]
```

The destructor frees all memory and server resources that are used by the SVG image.

31.124.3 Member Function Documentation**31.124.3.1 as_svg_image()**

```
virtual Fl_SVG_Image* Fl_SVG_Image::as_svg_image ( ) [inline], [virtual]
```

Returns whether an 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](#).

31.124.3.2 color_average()

```
void Fl_SVG_Image::color_average (
    Fl_Color c,
    float i ) [virtual]
```

The [color_average\(\)](#) method averages the colors in the image with the FLTK color value *c*.

The *i* 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 before changes are applied, to avoid modifying the original image.

Reimplemented from [Fl_RGB_Image](#).

31.124.3.3 copy()

```
Fl_Image * Fl_SVG_Image::copy (
    int W,
    int H ) [virtual]
```

Creates a resized copy of the specified image.

The image should be deleted (or in the case of [Fl_Shared_Image](#), released) when you are done with it.

Parameters

<i>W,H</i>	width and height of the returned copied image
------------	---

Reimplemented from [Fl_RGB_Image](#).

31.124.3.4 desaturate()

```
void Fl_SVG_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 before changes are applied, to avoid modifying the original image.

Reimplemented from [Fl_RGB_Image](#).

31.124.3.5 draw()

```
void Fl_SVG_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.

In other words: `fl_push_clip(X, Y, W, H)` is applied, the image is drawn with its upper-left corner at `X-cx, Y-cy` and its own width and height, `fl_pop_clip()` is applied.

Reimplemented from [Fl_RGB_Image](#).

31.124.3.6 normalize()

```
void Fl_SVG_Image::normalize ( ) [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](#).

31.124.3.7 `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.

31.124.4 Member Data Documentation

31.124.4.1 `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:

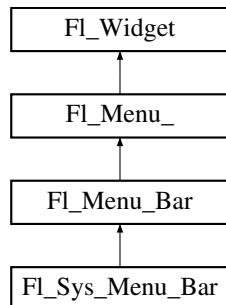
- Fl_SVG_Image.H
- Fl_SVG_Image.cxx

31.125 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`, 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 *`label`, const char *`shortcut`, `Fl_Callback` *`cb`, void *`user_data`=0, int `flags`=0)
Adds a new menu item.
- 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`, int `shortcut`, `Fl_Callback` *`cb`, void *`user_data`=0, int `flags`=0)
insert in the system menu bar a new menu item
- 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.
- 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
- void `mode` (int `i`, int `f`)
Sets the flags of item i.
- int `mode` (int `i`) const
Gets the flags of item i.
- 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.
- virtual void `update` ()
Updates the menu bar after any change to its items.
- virtual ~`Fl_Sys_Menu_Bar` ()
The destructor.

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.

Protected Member Functions

- virtual void `draw ()`
Draws the widget.

Additional Inherited Members

31.125.1 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 than 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 31.40 Mac Application submenu

31.125.2 Member Enumeration Documentation

31.125.2.1 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.

31.125.3 Constructor & Destructor Documentation

31.125.3.1 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\(\)](#).

31.125.4 Member Function Documentation

31.125.4.1 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

<i>cb</i>	a callback that will be called by "About myprog" menu item with NULL 1st argument.
<i>data</i>	a pointer transmitted as 2nd argument to the callback.

31.125.4.2 add() [1/3]

```
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

<i>label</i>	- new menu item's label
<i>shortcut</i>	- new menu item's integer shortcut (can be 0 for none, or e.g. FL_ALT+'x')
<i>cb</i>	- callback to be invoked when item selected (can be 0 for none, in which case the menubar's callback() can be used instead)
<i>user_data</i>	- argument to the callback
<i>flags</i>	- 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\(const char* label, int shortcut, Fl_Callback *cb, void *user_data, int flags\)](#)

31.125.4.3 add() [2/3]

```
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\(const char* label, int shortcut, Fl_Callback*, void *user_data=0, int flags=0\)](#)

31.125.4.4 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\(const char* str\)](#)

31.125.4.5 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\(\)](#)

31.125.4.6 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\(int index\)](#)

31.125.4.7 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

31.125.4.8 draw()

```
void Fl_Sys_Menu_Bar::draw ( ) [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](#).

31.125.4.9 insert() [1/2]

```
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\(int index, const char* label, int shortcut, Fl_Callback *cb, void *user_data, int flags\)](#)

31.125.4.10 insert() [2/2]

```
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\(int index, const char* label, const char* shortcut, Fl_Callback *cb, void *user_data=0, int flags=0\)](#)

31.125.4.11 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

<i>m</i>	Zero-ending list of Fl_Menu_Item 's
----------	---

31.125.4.12 mode()

```
void Fl_Sys_Menu_Bar::mode (
    int i,
    int fl )
```

Sets the flags of item i.

See also

[Fl_Menu_::mode\(int i, int fl\)](#)

31.125.4.13 remove()

```
void Fl_Sys_Menu_Bar::remove (
    int index )
```

remove an item from the system menu bar

Parameters

<i>index</i>	the index of the item to remove
--------------	---------------------------------

31.125.4.14 replace()

```
void Fl_Sys_Menu_Bar::replace (
    int index,
    const char * name )
```

rename an item from the system menu bar

Parameters

<i>index</i>	the index of the item to rename
<i>name</i>	the new item name as a UTF8 string

31.125.4.15 setonly()

```
void Fl_Sys_Menu_Bar::setonly (
    Fl_Menu_Item * item )
```

Turns the radio item "on" for the menu item and turns "off" adjacent radio items of the same group.

31.125.4.16 update()

```
void Fl_Sys_Menu_Bar::update ( ) [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](#).

31.125.4.17 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
- `tapping_mode_none` : add a simple Window menu to the system menu bar
- `tapping_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.
- `tapping_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:

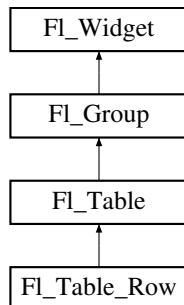
- [Fl_Sys_Menu_Bar.H](#)
- [Fl_Sys_Menu_Bar.cxx](#)

31.126 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_RO←
 W_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 ()`
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 ()`
Returns the current column the event occurred on.
- `TableContext callback_context ()`
Returns the current 'table context'.
- int `callback_row ()`
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 ()`
Returns if column headers are enabled or not.
- void `col_header (int flag)`
Enable or disable column headers.
- void `col_header_color (Fl_Color val)`
Sets the color for column headers and redraws the table.
- `Fl_Color col_header_color ()`
Gets the color for column headers.
- void `col_header_height (int height)`
Sets the height in pixels for column headers and redraws the table.
- int `col_header_height ()`
Gets the column header height.
- void `col_position (int col)`
Sets the horizontal scroll position so 'col' is at the left, and causes the screen to redraw.
- int `col_position ()`
Returns the current column scroll position as a column number.
- int `col_resize ()`
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` ()

Returns the current column minimum resize value.
- void `col_resize_min` (int val)

Sets the current column minimum resize value.
- void `col_width` (int col, int width)

Sets the width of the specified column in pixels, and the table is redrawn.
- int `col_width` (int col)

Returns the current width of the specified column in pixels.
- void `col_width_all` (int width)

Convenience method to set the width of all columns to the same value, in pixels.
- virtual void `cols` (int val)

Set the number of columns in the table and redraw.
- int `cols` ()

Get the number of columns in the table.
- void `do_callback` (TableContext context, int row, int col)

Calls the widget callback.
- void `end` ()
- int `find` (const `FI_Widget` *wgt) const
- int `find` (const `FI_Widget` &wgt) const
- `FI_Table` (int X, int Y, int W, int H, const char *l=0)

The constructor for `FI_Table`.
- void `get_selection` (int &row_top, int &col_left, int &row_bot, int &col_right)

Gets the region of cells selected (highlighted).
- void `init_sizes` ()

Resets the internal array of widget sizes and positions.
- void `insert` (`FI_Widget` &wgt, int n)

The specified widget is removed from its current group (if any) and inserted into the `FI_Table`'s group at position 'n'.
- void `insert` (`FI_Widget` &wgt, `FI_Widget` *w2)

The specified widget is removed from its current group (if any) and inserted into `FI_Table`'s group before widget 'w2'.
- int `is_interactive_resize` ()

Returns 1 if someone is interactively resizing a row or column.
- int `is_selected` (int r, int c)

See if the cell at row `r` and column `c` is selected.
- int `move_cursor` (int R, int C, int shiftselect)

Moves the selection cursor a relative number of rows/columns specified by R/C.
- int `move_cursor` (int R, int C)

Same as `move_cursor(R,C,1)`;
- void `remove` (`FI_Widget` &wgt)

The specified widget is removed from `FI_Table`'s group.
- void `resize` (int X, int Y, int W, int H)

Handle resize events if user resizes parent window.
- int `row_header` ()

Returns if row headers are enabled or not.
- void `row_header` (int flag)

Enables/disables showing the row headers.
- void `row_header_color` (`FI_Color` val)

Sets the row header color and causes the screen to redraw.
- `FI_Color` `row_header_color` ()

Returns the current row header color.

- void [row_header_width](#) (int width)
Sets the row header width to n and causes the screen to redraw.
- int [row_header_width](#) ()
Returns the current row header width (in pixels).
- void [row_height](#) (int row, int height)
Sets the height of the specified row in pixels, and the table is redrawn.
- int [row_height](#) (int row)
Returns the current height of the specified row as a value in pixels.
- void [row_height_all](#) (int height)
Convenience method to set the height of all rows to the same value, in pixels.
- void [row_position](#) (int row)
Sets the vertical scroll position so 'row' is at the top, and causes the screen to redraw.
- int [row_position](#) ()
Returns the current row scroll position as a row number.
- int [row_resize](#) ()
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](#) ()
Returns the current row minimum resize value.
- void [row_resize_min](#) (int val)
Sets the current row minimum resize value.
- virtual void [rows](#) (int val)
Sets the number of rows in the table, and the table is redrawn.
- int [rows](#) ()
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).
- void [tab_cell_nav](#) (int val)
Flag to control if Tab navigates table cells or not.
- int [tab_cell_nav](#) () const
Get state of table's 'Tab' key cell navigation flag.
- 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)
Returns the current box type used for the data 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.
- int [top_row](#) ()
Returns the current top row shown in the table.
- void [visible_cells](#) (int &r1, int &r2, int &c1, int &c2)
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.

Protected Types

- enum **ResizeFlag** {

RESIZE_NONE = 0, RESIZE_COL_LEFT = 1, RESIZE_COL_RIGHT = 2, RESIZE_ROW_ABOVE = 3,

RESIZE_ROW_BELOW = 4 }

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** ()

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)

Handle FLTK events.
- int **is_fltk_container** ()

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.

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)
- **FI_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)
- **FI_Scroll * table**
child [FI_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](#).
- **FI_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

31.126.1 Detailed Description

A table of widgets or other content.

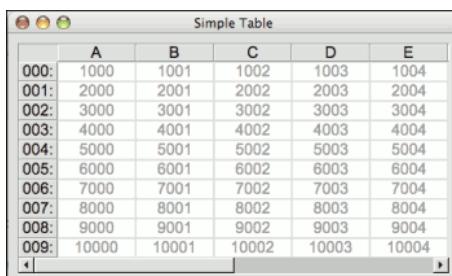
This is the base class for table widgets.

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:

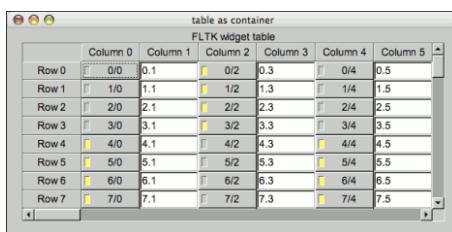
- As a custom widget; see `examples/table-simple.cxx` and `test/table.cxx`. Very optimal for even extremely large tables.
- As a table made up of a single FLTK widget instanced all over the table, simulating a numeric spreadsheet. See `examples/table-spreadsheet.cxx` and `examples/table-spreadsheet-with-keyboard-nav.cxx`. 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.



The screenshot shows a window titled "Simple Table". Inside, there is a 10x5 grid of numerical values. The columns are labeled A, B, C, D, and E. The rows are labeled 000: through 009:. The values range from 1000 to 10004. The window has standard OS X-style window controls (minimize, maximize, close).

	A	B	C	D	E
000:	1000	1001	1002	1003	1004
001:	2000	2001	2002	2003	2004
002:	3000	3001	3002	3003	3004
003:	4000	4001	4002	4003	4004
004:	5000	5001	5002	5003	5004
005:	6000	6001	6002	6003	6004
006:	7000	7001	7002	7003	7004
007:	8000	8001	8002	8003	8004
008:	9000	9001	9002	9003	9004
009:	10000	10001	10002	10003	10004

Figure 31.41 table-simple example



The screenshot shows a window titled "table as container". Inside, there is a 8x6 grid of numerical values. The columns are labeled Column 0 through Column 5. The rows are labeled Row 0 through Row 7. The values range from 0.1 to 7.5. The window has standard OS X-style window controls (minimize, maximize, close).

	Column 0	Column 1	Column 2	Column 3	Column 4	Column 5
Row 0	0/0	0.1	0/2	0.3	0/4	0.5
Row 1	1/0	1.1	1/2	1.3	1/4	1.5
Row 2	2/0	2.1	2/2	2.3	2/4	2.5
Row 3	3/0	3.1	3/2	3.3	3/4	3.5
Row 4	4/0	4.1	4/2	4.3	4/4	4.5
Row 5	5/0	5.1	5/2	5.3	5/4	5.5
Row 6	6/0	6.1	6/2	6.3	6/4	6.5
Row 7	7/0	7.1	7/2	7.3	7/4	7.5

Figure 31.42 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`:

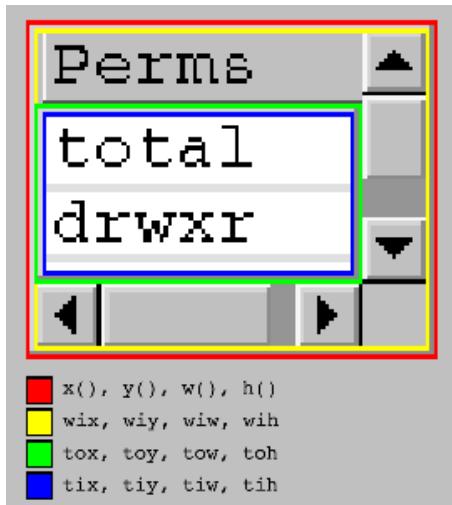


Figure 31.43 `Fl_Table` Dimensions

<code>x()/y()/w()/h()</code>	<code>Fl_Table</code> widget's outer dimension. The outer edge of the border of the <code>Fl_Table</code> . (Red in the diagram above)
<code>wix/wiy/wiw/wih</code>	<code>Fl_Table</code> widget's inner dimension. The inner edge of the border of the <code>Fl_Table</code> . eg. if the <code>Fl_Table</code> 's <code>box()</code> is <code>FL_NO_BOX</code> , these values are the same as <code>x()/y()/w()/h()</code> . (Yellow in the diagram above)
<code>tox/toy/tow/toh</code>	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)
<code>tix/tiy/tiw/tih</code>	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 <code>table_box()</code> is <code>FL_N_O_BOX</code> , these values are the same as <code>tox/toy/tow/toh</code> . (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]

31.126.2 Member Enumeration Documentation

31.126.2.1 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

31.126.3 Constructor & Destructor Documentation

31.126.3.1 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.

31.126.3.2 ~Fl_Table()

```
Fl_Table::~Fl_Table ( )
```

The destructor for [Fl_Table](#).

Destroys the table and its associated widgets.

31.126.4 Member Function Documentation

31.126.4.1 array()

```
Fl_Widget* const* Fl_Table::array ( ) [inline]
```

Returns a pointer to the array of children.

This pointer is only valid until the next time a child is added or removed.

31.126.4.2 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:

<code>Fl_Table::CONTEXT_ROW_HEADER</code>	Someone clicked on a row header. Excludes resizing.
<code>Fl_Table::CONTEXT_COL_HEADER</code>	Someone clicked on a column header. Excludes resizing.
<code>Fl_Table::CONTEXT_CELL</code>	Someone clicked on a cell. To receive callbacks for <code>FL_RELEASE</code> events, you must set <code>when(FL_WHEN_RELEASE)</code> .
<code>Fl_Table::CONTEXT_RC_RESIZE</code>	Someone is resizing rows/columns either interactively, or via the <code>col_width()</code> or <code>row_height()</code> API. Use <code>is_interactive_resize()</code> 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.
Generated by Doxygen	NOTE: To receive resize events, you must set <code>when(FL_WHEN_CHANGED)</code> .

```

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
    }
};

```

31.126.4.3 callback_col()

`int Fl_Table::callback_col () [inline]`

Returns the current column the event occurred on.

This function should only be used from within the user's callback function.

31.126.4.4 callback_context()

`TableContext Fl_Table::callback_context () [inline]`

Returns the current 'table context'.

This function should only be used from within the user's callback function.

31.126.4.5 callback_row()

`int Fl_Table::callback_row () [inline]`

Returns the current row the event occurred on.

This function should only be used from within the user's callback function.

31.126.4.6 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);
    [...]
}
```

31.126.4.7 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\(int\)](#)

31.126.4.8 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](#).

31.126.4.9 col_header()

```
void Fl_Table::col_header (
    int flag ) [inline]
```

Enable or disable column headers.

If changed, the table is redrawn.

31.126.4.10 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.

31.126.4.11 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 .

31.126.4.12 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.

31.126.4.13 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.

31.126.4.14 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.

31.126.4.15 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\(\)](#).

31.126.4.16 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\(\)](#).

31.126.4.17 draw()

```
void Fl_Table::draw (
    void ) [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](#).

31.126.4.18 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:

<code>Fl_Table::CONTEXT_STARTPAGE</code>	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())
<code>Fl_Table::CONTEXT_ENDPAGE</code>	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)
<code>Fl_Table::CONTEXT_ROW_HEADER</code>	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
<code>Fl_Table::CONTEXT_COL_HEADER</code>	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
<code>Fl_Table::CONTEXT_CELL</code>	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
<code>Fl_Table::CONTEXT_RC_RESIZE</code>	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.

`row` and `col` will be set to the row and column number of the cell being drawn. In the case of row headers, `col` will be 0. In the case of column headers, `row` 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( row_selected(R) ? 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
}

```

31.126.4.19 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.

31.126.4.20 get_selection()

```
void Fl_Table::get_selection (
    int & row_top,
    int & col_left,
    int & row_bot,
    int & col_right )
```

Gets the region of cells selected (highlighted).

Parameters

in	<i>row_top</i>	Returns the top row of selection area
in	<i>col_left</i>	Returns the left column of selection area
in	<i>row_bot</i>	Returns the bottom row of selection area
in	<i>col_right</i>	Returns the right column of selection area

31.126.4.21 init_sizes()

```
void Fl_Table::init_sizes ( ) [inline]
```

Resets the internal array of widget sizes and positions.

See also

[Fl_Group::init_sizes\(\)](#)

31.126.4.22 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.

31.126.4.23 is_interactive_resize()

```
int Fl_Table::is_interactive_resize ( ) [inline]
```

Returns 1 if someone is interactively resizing a row or column.

You can currently call this only from within your [callback\(\)](#).

31.126.4.24 is_selected()

```
int Fl_Table::is_selected (
    int r,
    int c )
```

See if the cell at row *r* and column *c* is selected.

Returns

1 if the cell is selected, 0 if not.

31.126.4.25 move_cursor()

```
int Fl_Table::move_cursor (
    int R,
    int C,
    int shiftselect )
```

Moves the selection cursor a relative number of rows/columns specified 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.
```

31.126.4.26 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.

31.126.4.27 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.

31.126.4.28 resize()

```
void Fl_Table::resize (
    int X,
    int Y,
    int W,
    int H ) [virtual]
```

Handle resize events if user resizes parent window.

This changes the size of [Fl_Table](#), causing it to redraw.

Reimplemented from [Fl_Group](#).

31.126.4.29 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.

31.126.4.30 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.

31.126.4.31 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.

31.126.4.32 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.

31.126.4.33 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.

31.126.4.34 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.

31.126.4.35 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\(int\)](#)

31.126.4.36 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	<code>newSize</code>	Sets the scrollbar size in pixels. If 0 (default), scrollbar size tracks the global Fl::scrollbar_size()
----	----------------------	---

See also

[Fl::scrollbar_size\(\)](#)

31.126.4.37 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	<code>row_top</code>	Top row of selection area
in	<code>col_left</code>	Left column of selection area
in	<code>row_bot</code>	Bottom row of selection area
in	<code>col_right</code>	Right column of selection area

31.126.4.38 tab_cell_nav() [1/2]

```
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.
----	-----	--

31.126.4.39 tab_cell_nav() [2/2]

```
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\(int\)](#)

31.126.4.40 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.

31.126.4.41 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.

31.126.4.42 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.

31.126.4.43 top_row() [1/2]

```
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.

31.126.4.44 top_row() [2/2]

```
int Fl_Table::top_row ( ) [inline]
```

Returns the current top row shown in the table.

This row may be partially obscured.

31.126.4.45 visible_cells()

```
void Fl_Table::visible_cells (
    int & r1,
    int & r2,
    int & c1,
    int & c2 ) [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

31.126.4.46 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	<code>callback()</code> will be called when rows or columns are resized (interactively or via <code>col_width()</code> or <code>row_height()</code>), passing CONTEXT_RC_RESIZE via <code>callback_context()</code> .
FL_WHEN_RELEASE	<code>callback()</code> 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:

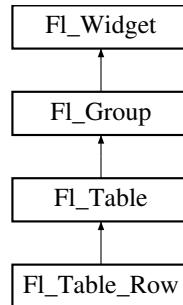
- Fl_Table.H
- Fl_Table.cxx

31.127 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 **TableRowSelectionMode** { **SELECT_NONE**, **SELECT_SINGLE**, **SELECT_MULTI** }

Public Member Functions

- 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.
- **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.
- void **rows** (int val)

Sets the number of rows in the table, and the table is redrawn.

- int **rows** ()
- 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'.
- void **type** (TableRowSelectMode val)

Sets the table selection mode.
- TableRowSelectMode **type** () const
- **~Fl_Table_Row** ()

The destructor for the [Fl_Table_Row](#).

Protected Member Functions

- int **find_cell** ([TableContext](#) context, int R, int C, int &X, int &Y, int &W, int &H)
- int **handle** (int event)

Handle FLTK events.

Additional Inherited Members

31.127.1 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\(\)](#).

31.127.2 Constructor & Destructor Documentation

31.127.2.1 [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.

31.127.2.2 ~Fl_Table_Row()

```
Fl_Table_Row::~Fl_Table_Row ( ) [inline]
```

The destructor for the [Fl_Table_Row](#).

Destroys the table and its associated widgets.

31.127.3 Member Function Documentation

31.127.3.1 clear()

```
void Fl_Table_Row::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 from [Fl_Table](#).

31.127.3.2 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\(row, flag\)](#)

31.127.3.3 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.

31.127.3.4 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'.

0=deselected, 1=select, 2=toggle existing state.

31.127.3.5 type()

```
void Fl_Table_Row::type (
    TableRowSelectionMode 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:

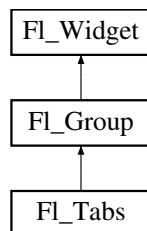
- `Fl_Table_Row.H`
- `Fl_Table_Row.cxx`

31.128 Fl_Tabs Class Reference

The `Fl_Tabs` widget is the "file card tabs" interface that allows you to put lots and lots of buttons and switches in a panel, as popularized by many toolkits.

```
#include <Fl_Tabs.H>
```

Inheritance diagram for `Fl_Tabs`:



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)
Handles the specified event.
- [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 [tab_align](#) ([FL_Align](#) a)
Sets the tab label alignment.
- [FL_Align tab_align](#) () const
Gets the tab label alignment.
- [FL_Widget * value](#) ()
Gets the currently visible widget/tab.
- int [value](#) ([FL_Widget](#) *)
Sets the widget to become the current visible widget/tab.
- virtual [FL_Widget * which](#) (int event_x, int event_y)
Return the widget of the tab the user clicked on at event_x / event_y.

Protected Member Functions

- virtual void [clear_tab_positions](#) ()
- void [draw](#) ()
Draws the widget.
- virtual void [draw_tab](#) (int x1, int x2, int W, int H, [FL_Widget](#) *o, int sel=0)
- virtual void [redraw_tabs](#) ()
- virtual int [tab_height](#) ()
- virtual int [tab_positions](#) ()

Protected Attributes

- [FL_Align tab_align_](#)
- int [tab_count](#)
- int * [tab_pos](#)
- int * [tab_width](#)

Additional Inherited Members

31.128.1 Detailed Description

The [Fl_Tabs](#) widget is the "file card tabs" interface that allows you to put lots and lots of buttons and switches in a panel, as popularized by many toolkits.

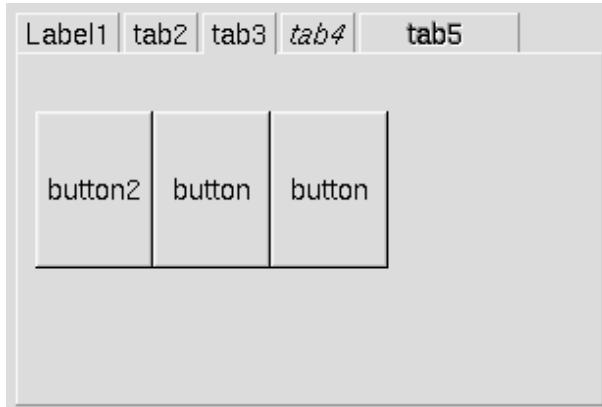


Figure 31.44 Fl_Tabs

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.

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_Tab, i.e. whether there is more space above or below them. The height of the "tabs" depends on how much free space is available.

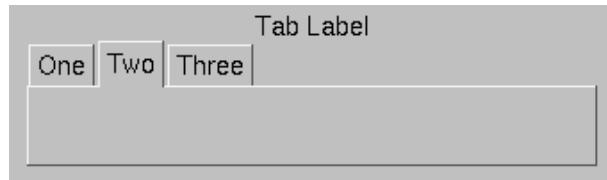


Figure 31.45 Fl_Tabs Default Appearance

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:

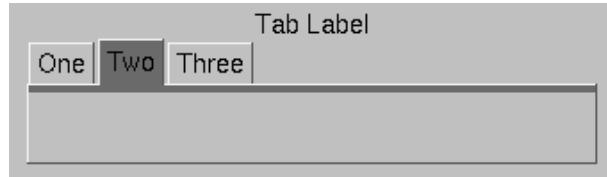


Figure 31.46 Highlighting the selected tab

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( (tab->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:

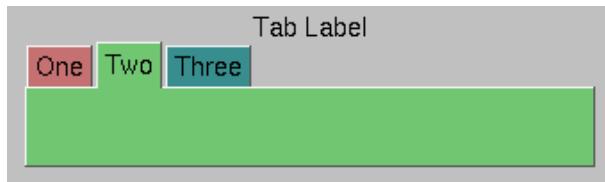


Figure 31.47 Fl_Tabs with uniform colors

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.

31.128.2 Constructor & Destructor Documentation

31.128.2.1 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 boxtyle 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.

31.128.3 Member Function Documentation

31.128.3.1 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.

`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	<code>tabh</code>	position and optional height of tabs (see above)
out	<code>rx,ry,rw,rh</code>	(x,y,w,h) of client area for children

Since

FLTK 1.3.0

31.128.3.2 draw()

```
void Fl_Tabs::draw ( ) [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](#).

31.128.3.3 handle()

```
int Fl_Tabs::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.

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](#).

31.128.3.4 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\(Fl_Widget*\)](#).

31.128.3.5 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\(\)](#).

31.128.3.6 tab_align() [1/2]

```
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_ICON_NEXT_TO_TEXT is the recommended alignment to show the icon left of the text.

31.128.3.7 tab_align() [2/2]

```
Fl_Align Fl_Tabs::tab_align ( ) const [inline]
```

Gets the tab label alignment.

See also

[tab_align\(Fl_Align\)](#)

31.128.3.8 value() [1/2]

```
Fl_Widget * Fl_Tabs::value ( )
```

Gets the currently visible widget/tab.

The [value\(\)](#) is the first visible child (or the last child if none are visible) and this also hides any other children. This allows the tabs to be deleted, moved to other groups, and [show\(\)/hide\(\)](#) called without it screwing up.

31.128.3.9 value() [2/2]

```
int Fl_Tabs::value (
    Fl_Widget * newvalue )
```

Sets the widget to become the current visible widget/tab.

Setting the value hides all other children, and makes this one visible, if it is really a child.

Returns

1 if there was a change (new value different from previous),
0 if there was no change (new value already set)

31.128.3.10 which()

```
Fl_Widget * Fl_Tabs::which (
    int event_x,
    int event_y ) [virtual]
```

Return the widget of the tab the user clicked on at event_x / event_y.

This is used for event handling (clicks) and by fluid to pick tabs.

Returns

The child widget of the tab the user clicked on, or
0 if there are no children or if the event is outside of the tabs area.

The documentation for this class was generated from the following files:

- Fl_Tabs.H
- Fl_Tabs.cxx

31.129 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_pdelete_callback](#) (Fl_Text_Pdelete_Cb bufPdelCB, void *cbArg)

Adds a callback routine to be called before text is deleted from the buffer.
- const char * [address](#) (int pos) const

Convert a byte offset in buffer into a memory address.
- char * [address](#) (int pos)

Convert a byte offset in buffer into a memory address.
- void [append](#) (const char *t)

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_pdelete_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.
- void [canUndo](#) (char flag=1)

Lets the undo system know if we can undo changes.
- 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 [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.
- void [highlight](#) (int start, int end)

Highlights the specified text within the buffer.
- int [highlight](#) ()

Returns a non-zero value if text has been highlighted, 0 otherwise.
- int [highlight_position](#) (int *start, int *end)

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](#) ()

Returns the highlighted text.
- void [insert](#) (int pos, const char *text)

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 ix) const`
Returns the index of the next character.
 - `int next_char_clipped (int ix) 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 ix) const`
Returns the index of the previous character.
 - `int prev_char_clipped (int ix) const`
 - `const FI_Text_Selection * primary_selection () const`
Returns the primary selection.
 - `FI_Text_Selection * primary_selection ()`
Returns the primary selection.
 - `void printf (const char *fmt,...)`
Appends printf formatted messages to the end of the buffer.
 - `void remove (int start, int end)`
Deletes a range of characters in the buffer.
 - `void remove_modify_callback (FI_Text_Modify_Cb bufModifiedCB, void *cbArg)`
Removes a modify callback.
 - `void remove_predelete_callback (FI_Text_Predelete_Cb predelCB, void *cbArg)`
Removes a callback routine `bufPPreDeleteCB` 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)`
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)`
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** ()
Returns a non-zero value if text has been selected in the secondary text selection, 0 otherwise.
 - const **FI_Text_Selection * secondary_selection** () const
Returns the secondary selection.
 - int **secondary_selection_position** (int *`start`, int *`end`)
Returns the current selection in the secondary text selection object.
 - char * **secondary_selection_text** ()
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`)
Gets the selection position.
 - char * **selection_text** ()
Returns the currently selected text.
 - int **skip_displayed_characters** (int `lineStartPos`, int `nChars`)
Count forward from buffer position `startPos` in displayed characters.
 - int **skip_lines** (int `startPos`, int `nLines`)
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.
 - 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

- `int word_start (int pos) const`
Returns the position corresponding to the end of the word.
- `~FI_Text_Buffer ()`
Returns the position corresponding to the start of the word.
- `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)(FI_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

- `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)`
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 (FI_Text_Selection *oldSelection, FI_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_ (FI_Text_Selection *sel)`
Removes the text from the buffer corresponding to `sel`.
- `void replace_selection_ (FI_Text_Selection *sel, const char *text)`
Replaces the text in selection `sel`.
- `char *selection_text_ (FI_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
- `FI_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)
- `FI_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
- `FI_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
- `FI_Text_Selection mPrimary`
highlighted areas
- `FI_Text_Selection mSecondary`
highlighted areas
- `int mTabDist`
equiv.

31.129.1 Detailed Description

This class manages Unicode text displayed in one or more `FI_Text_Display` widgets.

All text in `FI_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 `FI_Text_Buffer` class is used by the `FI_Text_Display` and `FI_Text_Editor` to manage complex text data and is based upon the excellent NEdit text editor engine - see <https://sourceforge.net/projects/nedit/>.

31.129.2 Constructor & Destructor Documentation

31.129.2.1 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

<i>requestedSize</i>	use this to avoid unnecessary re-allocation if you know exactly how much the buffer will need to hold
<i>preferredGapSize</i>	Initial size for the buffer gap (empty space in the buffer where text might be inserted if the user is typing sequential characters)

31.129.3 Member Function Documentation

31.129.3.1 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);
```

31.129.3.2 address() [1/2]

```
const char* Fl_Text_Buffer::address (
    int pos ) const [inline]
```

Convert a byte offset in buffer into a memory address.

Parameters

<i>pos</i>	byte offset into buffer
------------	-------------------------

Returns

byte offset converted to a memory address

31.129.3.3 address() [2/2]

```
char* Fl_Text_Buffer::address (
    int pos ) [inline]
```

Convert a byte offset in buffer into a memory address.

Parameters

<i>pos</i>	byte offset into buffer
------------	-------------------------

Returns

byte offset converted to a memory address

31.129.3.4 append()

```
void Fl_Text_Buffer::append (
    const char * t ) [inline]
```

Appends the text string to the end of the buffer.

Parameters

<i>t</i>	UTF-8 encoded and nul terminated text
----------	---------------------------------------

31.129.3.5 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\(\)](#).

31.129.3.6 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

<i>pos</i>	byte offset into buffer
------------	-------------------------

Returns

unencoded raw byte

31.129.3.7 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

<i>pos</i>	byte offset into buffer, <i>pos</i> must be at a UTF-8 character boundary
------------	---

Returns

Unicode UCS-4 encoded character

31.129.3.8 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

<i>fromBuf</i>	source text buffer, may be the same as this
<i>fromStart</i>	byte offset into buffer
<i>fromEnd</i>	byte offset into buffer
<i>toPos</i>	destination byte offset into buffer

31.129.3.9 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.

31.129.3.10 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.

31.129.3.11 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

<i>startPos</i>	byte offset to start position
<i>searchChar</i>	UCS-4 character that we want to find
<i>foundPos</i>	byte offset where the character was found

Returns

1 if found, 0 if not

31.129.3.12 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

<code>startPos</code>	byte offset to start position
<code>searchChar</code>	UCS-4 character that we want to find
<code>foundPos</code>	byte offset where the character was found

Returns

1 if found, 0 if not

31.129.3.13 highlight_text()

```
char * Fl_Text_Buffer::highlight_text ( )
```

Returns the highlighted text.

When you are done with the text, free it using the `free()` function.

31.129.3.14 insert()

```
void Fl_Text_Buffer::insert (
    int pos,
    const char * text )
```

Inserts null-terminated string `text` at position `pos`.

Parameters

<i>pos</i>	insertion position as byte offset (must be UTF-8 character aligned)
<i>text</i>	UTF-8 encoded and nul terminated text

31.129.3.15 insert_()

```
int Fl_Text_Buffer::insert_ (
    int pos,
    const char * text ) [protected]
```

Internal (non-redisplaying) version of [insert\(\)](#).

Returns the length of text inserted (this is just `strlen(text)`, 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

31.129.3.16 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](#).

31.129.3.17 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.

31.129.3.18 length()

```
int Fl_Text_Buffer::length ( ) const [inline]
```

Returns the number of bytes in the buffer.

Returns

size of text in bytes

31.129.3.19 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

<i>pos</i>	byte index into buffer
------------	------------------------

Returns

byte offset to line end

31.129.3.20 line_start()

```
int Fl_Text_Buffer::line_start (
    int pos ) const
```

Returns the position of the start of the line containing position *pos*.

Parameters

<i>pos</i>	byte index into buffer
------------	------------------------

Returns

byte offset to line start

31.129.3.21 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

<i>pos</i>	byte index into buffer
------------	------------------------

Returns

copy of UTF-8 text, must be free'd

31.129.3.22 loadfile()

```
int Fl_Text_Buffer::loadfile (
    const char * file,
    int buflen = 128*1024 ) [inline]
```

Loads a text file into the buffer.

See also [insertfile\(\)](#).

31.129.3.23 next_char()

```
int Fl_Text_Buffer::next_char (
    int ix ) const
```

Returns the index of the next character.

Parameters

<i>ix</i>	index to the current character
-----------	--------------------------------

31.129.3.24 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\(const char *file, int buflen\)](#)

31.129.3.25 prev_char()

```
int Fl_Text_Buffer::prev_char (
    int ix ) const
```

Returns the index of the previous character.

Parameters

<i>ix</i>	index to the current character
-----------	--------------------------------

31.129.3.26 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

<code>in</code>	<code>fmt</code>	is a printf format string for the message text.
-----------------	------------------	---

31.129.3.27 remove()

```
void Fl_Text_Buffer::remove (
    int start,
    int end )
```

Deletes a range of characters in the buffer.

Parameters

<code>start</code>	byte offset to first character to be removed
<code>end</code>	byte offset to character after last character to be removed

31.129.3.28 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).

31.129.3.29 replace()

```
void Fl_Text_Buffer::replace (
    int start,
    int end,
    const char * text )
```

Deletes the characters between `start` and `end`, and inserts the null-terminated string `text` in their place in the buffer.

Parameters

<i>start</i>	byte offset to first character to be removed and new insert position
<i>end</i>	byte offset to character after last character to be removed
<i>text</i>	UTF-8 encoded and nul terminated text

31.129.3.30 rewind_lines()

```
int Fl_Text_Buffer::rewind_lines (
    int startPos,
    int nLines )
```

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.

31.129.3.31 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\(const char *file, int start, int end, int buflen\)](#)

31.129.3.32 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

<i>startPos</i>	byte offset to start position
<i>searchString</i>	UTF-8 string that we want to find
<i>foundPos</i>	byte offset where the string was found
<i>matchCase</i>	if set, match character case

Returns

1 if found, 0 if not

31.129.3.33 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

<i>startPos</i>	byte offset to start position
<i>searchString</i>	UTF-8 string that we want to find
<i>foundPos</i>	byte offset where the string was found
<i>matchCase</i>	if set, match character case

Returns

1 if found, 0 if not

31.129.3.34 secondary_selection_text()

```
char * Fl_Text_Buffer::secondary_selection_text ( )
```

Returns the text in the secondary selection.

When you are done with the text, free it using the free() function.

31.129.3.35 selection_text()

```
char * Fl_Text_Buffer::selection_text ( )
```

Returns the currently selected text.

When you are done with the text, free it using the free() function.

31.129.3.36 skip_displayed_characters()

```
int Fl_Text_Buffer::skip_displayed_characters (
    int lineStartPos,
    int nChars )
```

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

<code>lineStartPos</code>	byte offset into buffer
<code>nChars</code>	number of bytes that are sent to the display

Returns

byte offset in input after all output bytes are sent

31.129.3.37 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.

31.129.3.38 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

31.129.3.39 `text()` [2/2]

```
void Fl_Text_Buffer::text (
    const char * text )
```

Replaces the entire contents of the text buffer.

Parameters

<i>text</i>	Text must be valid UTF-8. If null, an empty string is substituted.
-------------	--

31.129.3.40 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

<i>start</i>	byte offset to first character
<i>end</i>	byte offset after last character in range

Returns

newly allocated text buffer - must be free'd, text is UTF-8

31.129.3.41 vprintf()

```
void Fl_Text_Buffer::vprintf (
    const char * fmt,
    va_list ap )
```

Can be used by subclasses that need their own [printf\(\)](#) style functionality.

e.g. [Fl_Simple_Terminal::printf\(\)](#) would wrap around this method.

Note

The expanded string is currently limited to 1024 characters.

Parameters

<i>in</i>	<i>fmt</i>	is a printf format string for the message text.
<i>in</i>	<i>ap</i>	is a va_list created by va_start() and closed with va_end(), which the caller is responsible for handling.

31.129.3.42 word_end()

```
int Fl_Text_Buffer::word_end (
    int pos ) const
```

Returns the position corresponding to the end of the word.

Parameters

<i>pos</i>	byte index into buffer
------------	------------------------

Returns

byte offset to word end

31.129.3.43 word_start()

```
int Fl_Text_Buffer::word_start (
    int pos ) const
```

Returns the position corresponding to the start of the word.

Parameters

<i>pos</i>	byte index into buffer
------------	------------------------

Returns

byte offset to word start

31.129.4 Member Data Documentation

31.129.4.1 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.

31.129.4.2 mPredeleteProcs

```
Fl_Text_Predelete_Cb* Fl_Text_Buffer::mPredeleteProcs [protected]
```

procedure to call before text is deleted from the buffer; at most one is supported.

31.129.4.3 mTabDist

```
int Fl_Text_Buffer::mTabDist [protected]
```

equiv.

number of characters in a tab

31.129.4.4 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:

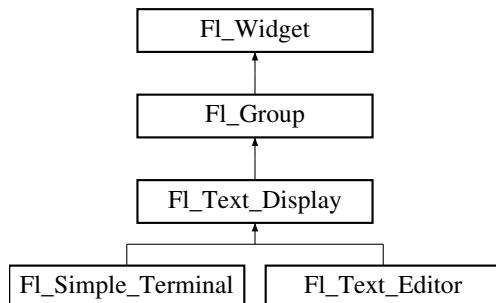
- Fl_Text_Buffer.H
- Fl_Text_Buffer.cxx

31.130 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()`
- `typedef void(*) Unfinished_Style_Cb (int, void *)`

Public Member Functions

- `void buffer (Fl_Text_Buffer *buf)`

Attach a text buffer to display, replacing the current buffer (if any).
- `void buffer (Fl_Text_Buffer &buf)`

Sets the current text buffer associated with the text widget.
- `Fl_Text_Buffer * buffer () const`

Gets the current text buffer associated with the text widget.
- `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.
- `void cursor_style (int style)`

Sets the text cursor style.
- `int cursor_style () const`
- `Fl_Text_Display (int X, int Y, int W, int H, const char *l=0)`

Creates a new text display widget.
- `virtual int handle (int e)`

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.
- void `insert_position` (int newPos)
Sets the position of the text insertion cursor for text display.
- int `insert_position` () const
Gets 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.
- void `linenumber_align` (FI_Align val)
Set alignment for line numbers (if enabled).
- FI_Align `linenumber_align` () const
Returns the alignment used for line numbers (if enabled).
- void `linenumber bgcolor` (FI_Color val)
Set the background color used for line numbers (if enabled).
- FI_Color `linenumber bgcolor` () const
Returns the background color used for line numbers (if enabled).
- void `linenumber fgcolor` (FI_Color val)
Set the foreground color used for line numbers (if enabled).
- FI_Color `linenumber fgcolor` () const
Return the foreground color used for line numbers (if enabled).
- void `linenumber font` (FI_Font val)
Set the font used for line numbers (if enabled).
- FI_Font `linenumber font` () const
Return the font used for line numbers (if enabled).
- void `linenumber format` (const char *val)
Sets the printf() style format string used for line numbers.
- const char * `linenumber format` () const
Returns the line number printf() format string.
- void `linenumber size` (FI_Fontsize val)
Set the font size used for line numbers (if enabled).
- FI_Fontsize `linenumber size` () const
Return the font size used for line numbers (if enabled).
- void `linenumber width` (int width)
Set width of screen area for line numbers.
- int `linenumber width` () const
Return the screen area width provided 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.
- `virtual void resize (int X, int Y, int W, int H)`
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.
- `FI_Align scrollbar_align () const`
Gets the scrollbar alignment type.
- `void scrollbar_align (FI_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 `FI::scrollbar_size()` unless a specific `scrollbar_width_` has been set.
- `void scrollbar_width (int width)`
Sets the global `FI::scrollbar_size()`, and forces this instance of the widget to use it.
- `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.
- `FI_Color textcolor () const`
Gets the default color of text in the widget.
- `void textcolor (FI_Color n)`
Sets the default color of text in the widget.
- `FI_Font textfont () const`
Gets the default font used when drawing text in the widget.
- `void textfont (FI_Font s)`
Sets the default font used when drawing text in the widget.
- `FI_Fontsize textsize () const`
Gets the default size of text in the widget.
- `void textsize (FI_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.

Protected Types

- enum {
DRAW_LINE, FIND_INDEX, FIND_INDEX_FROM_ZERO, GET_WIDTH,
FIND_CURSOR_INDEX }

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.
- virtual void `draw ()`
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 [get_absolute_top_line_number](#) () const
Returns the absolute (non-wrapped) line number of the first line displayed.
 - 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](#) (FI_Text_Buffer *buf, int startPos, int maxPos, int maxLines, bool startPosIsLineStart, int styleBufOffset, int *retPos, int *retLines, int *retLineStart, int *retLineEnd, bool countLastIsLineMissingNewLine=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.

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`
- int `dragging`
- int `dragPos`
- int `dragType`
- `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_**
 - int **shortcut_**
 -
- ```
struct {
 int h
 int w
 int x
 int y
} text_area
```
- [Fl\\_Color](#) **textcolor\_**
  - [Fl\\_Font](#) **textfont\_**
  - [Fl\\_Fontsize](#) **textsize\_**

## Friends

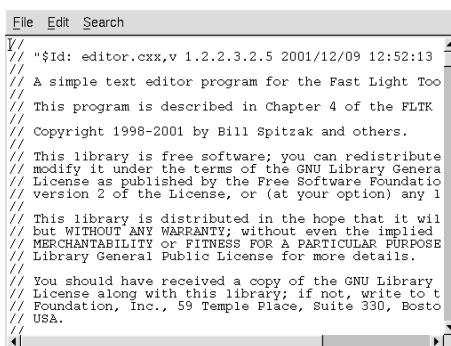
- void [fl\\_text\\_drag\\_me](#) (int pos, [Fl\\_Text\\_Display](#) \*d)

## Additional Inherited Members

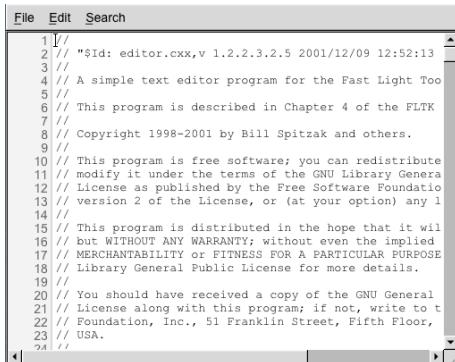
### 31.130.1 Detailed Description

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 31.48 Fl\_Text\_Display widget**



**Figure 31.49** 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\(int\)](#)

## 31.130.2 Member Enumeration Documentation

### 31.130.2.1 anonymous enum

anonymous enum

text display cursor shapes enumeration

## Enumerator

|               |                                              |
|---------------|----------------------------------------------|
| NORMAL_CURSOR | I-beam.                                      |
| 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 <a href="#">Fl_Input</a> cursor |

## 31.130.2.2 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 |

## 31.130.3 Constructor &amp; Destructor Documentation

31.130.3.1 [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 |

### 31.130.3.2 ~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\(Fl\\_Text\\_Buffer\\* buf\)](#)

## 31.130.4 Member Function Documentation

### 31.130.4.1 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.

### 31.130.4.2 buffer() [1/3]

```
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

|            |                         |
|------------|-------------------------|
| <i>buf</i> | attach this text buffer |
|------------|-------------------------|

## 31.130.4.3 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**

|            |                 |
|------------|-----------------|
| <i>buf</i> | new text buffer |
|------------|-----------------|

**See also**

[Fl\\_Text\\_Display::buffer\(Fl\\_Text\\_Buffer\\* buf\)](#)

## 31.130.4.4 buffer() [3/3]

```
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\(Fl\\_Text\\_Buffer& buf\)](#)

## 31.130.4.5 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**

|                    |                                                               |
|--------------------|---------------------------------------------------------------|
| <i>pos</i>         | starting index of modification                                |
| <i>nInserted</i>   | number of bytes we inserted (must be UTF-8 aligned!)          |
| <i>nDeleted</i>    | number of bytes deleted (must be UTF-8 aligned!)              |
| <i>nRestyled</i>   | ??                                                            |
| <i>deletedText</i> | this is what was removed, must not be NULL if nDeleted is set |
| <i>cbArg</i>       | "this" pointer for static callback function                   |

**31.130.4.6 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**

|                 |                                                         |
|-----------------|---------------------------------------------------------|
| <i>pos</i>      | starting index of deletion                              |
| <i>nDeleted</i> | number of bytes we will delete (must be UTF-8 aligned!) |
| <i>cbArg</i>    | "this" pointer for static callback function             |

**31.130.4.7 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.

**31.130.4.8 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**

|                          |                                        |
|--------------------------|----------------------------------------|
| <i>startLine,endLine</i> | range of lines to scan as line numbers |
|--------------------------|----------------------------------------|

**31.130.4.9 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**

|                         |                                      |
|-------------------------|--------------------------------------|
| <i>style</i>            | index into style table               |
| <i>X,Y,width,height</i> | size and position of background area |

**31.130.4.10 col\_to\_x()**

```
double Fl_Text_Display::col_to_x (
 double col) const
```

Convert a column number into an x pixel position.

**Parameters**

|            |                                                     |
|------------|-----------------------------------------------------|
| <i>col</i> | 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

**31.130.4.11 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

|                                  |                                       |
|----------------------------------|---------------------------------------|
| <code>startPos</code>            | index to first character              |
| <code>endPos</code>              | index after last character            |
| <code>startPosIsLineStart</code> | avoid scanning back to the line start |

#### Returns

number of lines

### 31.130.4.12 cursor\_color() [1/2]

```
Fl_Color Fl_Text_Display::cursor_color () const [inline]
```

Gets the text cursor color.

#### Returns

cursor color

### 31.130.4.13 cursor\_color() [2/2]

```
void Fl_Text_Display::cursor_color (
 Fl_Color n) [inline]
```

Sets the text cursor color.

#### Parameters

|                |                  |
|----------------|------------------|
| <code>n</code> | new cursor color |
|----------------|------------------|

### 31.130.4.14 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

|                    |                  |
|--------------------|------------------|
| <code>style</code> | new cursor style |
|--------------------|------------------|

#### 31.130.4.15 `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.

**Todo** Unicode?

#### 31.130.4.16 `draw()`

```
void Fl_Text_Display::draw (
 void) [protected], [virtual]
```

Draw the widget.

This function tries to limit drawing to smaller areas if possible.

Reimplemented from [Fl\\_Group](#).

Reimplemented in [Fl\\_Simple\\_Terminal](#).

#### 31.130.4.17 `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 |
|-----|---------------------------|

**31.130.4.18 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. |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**31.130.4.19 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**

|                       |                                    |
|-----------------------|------------------------------------|
| <code>startpos</code> | index of first character to draw   |
| <code>endpos</code>   | index after last character to draw |

**31.130.4.20 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

|                     |                                                |
|---------------------|------------------------------------------------|
| <code>style</code>  | index into style lookup table                  |
| <code>X,Y</code>    | drawing origin                                 |
| <code>toX</code>    | rightmost position if this is a fill operation |
| <code>string</code> | text if this is a drawing operation            |
| <code>nChars</code> | number of characters to draw                   |

#### 31.130.4.21 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

|                           |                                                |
|---------------------------|------------------------------------------------|
| <code>left,top</code>     | are in coordinates of the text drawing window. |
| <code>width,height</code> | size in pixels                                 |

#### 31.130.4.22 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**

|                                     |                                                 |
|-------------------------------------|-------------------------------------------------|
| <i>visLineNum</i>                   | index of line in the visible line number lookup |
| <i>leftClip,rightClip</i>           | pixel position of clipped area                  |
| <i>leftCharIndex,rightCharIndex</i> | index into line of segment that we want to draw |

**31.130.4.23 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

**31.130.4.24 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**

|                 |    |
|-----------------|----|
| <i>startpos</i> | ?? |
| <i>endpos</i>   | ?? |

**Todo** Unicode?

**31.130.4.25 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

|     |                            |  |
|-----|----------------------------|--|
|     | <i>startPos</i>            |  |
|     | <i>startPosIsLineStart</i> |  |
| out | <i>lineEnd</i>             |  |
| out | <i>nextLineStart</i>       |  |

#### 31.130.4.26 `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

|                      |  |
|----------------------|--|
| <i>deletedText</i>   |  |
| <i>pos</i>           |  |
| <i>nInserted</i>     |  |
| <i>nDeleted</i>      |  |
| <i>modRangeStart</i> |  |
| <i>modRangeEnd</i>   |  |
| <i>linesInserted</i> |  |
| <i>linesDeleted</i>  |  |

#### 31.130.4.27 `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

|              |                                                               |
|--------------|---------------------------------------------------------------|
| <i>s</i>     | UTF-8 text string                                             |
| <i>len</i>   | length of string                                              |
| <i>style</i> | index into style lookup table                                 |
| <i>x</i>     | position in pixels - negative returns closest cursor position |

#### Returns

index into buffer

### 31.130.4.28 get\_absolute\_top\_line\_number()

```
int Fl_Text_Display::get_absolute_top_line_number() const [protected]
```

Returns the absolute (non-wrapped) line number of the first line displayed.

Returns 0 if the absolute top line number is not being maintained.

### 31.130.4.29 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 | <i>mode</i>                          | DRAW_LINE, GET_WIDTH, FIND_INDEX, FIND_INDEX_FROM_ZERO, or FIND_CURSOR_INDEX                        |
| in | <i>lineStartPos</i>                  | index of first character                                                                            |
| in | <i>lineLen</i>                       | size of string in bytes                                                                             |
| in | <i>leftChar,rightChar</i>            |                                                                                                     |
| in | <i>Y</i>                             | drawing position                                                                                    |
| in | <i>bottomClip,leftClip,rightClip</i> | stop work when we reach the clipped area. rightClip is the X position that we search in FIND_INDEX. |

**Return values**

|                             |                                                                             |
|-----------------------------|-----------------------------------------------------------------------------|
| <i>DRAW_LINE</i>            | index of last drawn character                                               |
| <i>GET_WIDTH</i>            | width in pixels of text segment if we would draw it                         |
| <i>FIND_INDEX</i>           | index of character at given x position in window coordinates                |
| <i>FIND_INDEX_FROM_ZERO</i> | 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

**31.130.4.30 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 [extendRangeForStyleMods](#) 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.

**Parameters**

|                              |                                                                                                                                                                                                   |
|------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>styleBuffer</i>           | 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. |
| <i>styleTable</i>            | a list of styles indexed by the style buffer                                                                                                                                                      |
| <i>nStyles</i>               | number of styles in the style table                                                                                                                                                               |
| <i>unfinishedStyle</i>       | if this style is found, the callback below is called                                                                                                                                              |
| <i>unfinishedHighlightCB</i> | if a character with an unfinished style is found, this callback will be called                                                                                                                    |
| <i>cbArg</i>                 | an optional argument for the callback above, usually a pointer to the Text Display.                                                                                                               |

**Todo** "extendRangeForStyleMods" does not exist (might be a hangover from the port from nedit). Find the correct function.

**31.130.4.31 in\_selection()**

```
int Fl_Text_Display::in_selection (
 int X,
 int Y) const
```

Check if a pixel position is within the primary selection.

**Parameters**

|            |                        |
|------------|------------------------|
| <i>X,Y</i> | pixel position to test |
|------------|------------------------|

**Returns**

1 if position (X, Y) is inside of the primary [Fl\\_Text\\_Selection](#)

**31.130.4.32 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**

|             |                             |
|-------------|-----------------------------|
| <i>text</i> | new text in UTF-8 encoding. |
|-------------|-----------------------------|

### 31.130.4.33 insert\_position() [1/2]

```
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

|                     |                    |
|---------------------|--------------------|
| <code>newPos</code> | new caret position |
|---------------------|--------------------|

### 31.130.4.34 insert\_position() [2/2]

```
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\(int\)](#)

### 31.130.4.35 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**

|                            |                                       |
|----------------------------|---------------------------------------|
| <i>startPos</i>            | index to starting character           |
| <i>startPosIsLineStart</i> | avoid scanning back to the line start |

**Returns**

new position as index

**31.130.4.36 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**

|            |                             |
|------------|-----------------------------|
| <i>pos</i> | index to starting character |
|------------|-----------------------------|

**Returns**

new position as index

**31.130.4.37 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

**31.130.4.38 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

**31.130.4.39 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

**31.130.4.40 linenumber\_font()**

```
void Fl_Text_Display::linenumber_font (
 Fl_Font val)
```

Set the font used for line numbers (if enabled).

**Version**

1.3.3

**31.130.4.41 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

**31.130.4.42 linenumbersize()**

```
void Fl_Text_Display::linenumbersize (
 Fl_Fontsize val)
```

Set the font size used for line numbers (if enabled).

**Version**

1.3.3

**31.130.4.43 linumber\_width()**

```
void Fl_Text_Display::linumber_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**

|                    |                                                                                                    |
|--------------------|----------------------------------------------------------------------------------------------------|
| <code>width</code> | The new width of the area for line numbers to appear, in pixels. 0 disables line numbers (default) |
|--------------------|----------------------------------------------------------------------------------------------------|

**31.130.4.44 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

**31.130.4.45 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 TextDPosToLineAndCol). More specifically, this allows the line number reported in the statistics line to be calibrated in absolute lines, rather than post-wrapped lines.

**Todo** TextDPosToLineAndCol does not exist (nedit port?)

#### 31.130.4.46 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).

#### 31.130.4.47 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 |  |

#### 31.130.4.48 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**

|             |                                                    |
|-------------|----------------------------------------------------|
| <i>s</i>    | text string                                        |
| <i>xPix</i> | x pixel position needed for calculating tab widths |
| <i>pos</i>  | offset within string                               |

**Returns**

width of character in pixels

**31.130.4.49 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**

|                   |                                |
|-------------------|--------------------------------|
| <i>visLineNum</i> | index into visible lines array |
|-------------------|--------------------------------|

**Returns**

width of line in pixels

**31.130.4.50 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

**31.130.4.51 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

**31.130.4.52 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

**31.130.4.53 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

**31.130.4.54 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 newTopLineNum. 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**

|                      |                   |
|----------------------|-------------------|
| <i>newTopLineNum</i> | index into buffer |
|----------------------|-------------------|

**31.130.4.55 overstrike()**

```
void Fl_Text_Display::overstrike (
 const char * text)
```

Replaces text at the current insert position.

## Parameters

|             |                            |
|-------------|----------------------------|
| <i>text</i> | new text in UTF-8 encoding |
|-------------|----------------------------|

**Todo** Unicode? Find out exactly what we do here and simplify.

## 31.130.4.56 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.

## Parameters

|                     |                                   |
|---------------------|-----------------------------------|
| <i>lineStartPos</i> | beginning of this line            |
| <i>lineLen</i>      | number of bytes in line           |
| <i>lineIndex</i>    | position of character within line |

## Returns

style for the given character

## 31.130.4.57 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**

|            |                |                                                   |
|------------|----------------|---------------------------------------------------|
| <i>in</i>  | <i>pos</i>     | byte position in buffer                           |
| <i>out</i> | <i>lineNum</i> | relative line number of byte <i>pos</i> in buffer |

**Returns**

whether the character at byte position *pos* is currently displayed

**Return values**

|          |                                                               |
|----------|---------------------------------------------------------------|
| <i>0</i> | <i>pos</i> is not displayed; <i>lineNum</i> is invalid (zero) |
| <i>1</i> | <i>pos</i> is displayed; <i>lineNum</i> is valid              |

**31.130.4.58 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**

|            |                |                                               |
|------------|----------------|-----------------------------------------------|
|            | <i>pos</i>     | character index                               |
| <i>out</i> | <i>lineNum</i> | absolute (unwrapped) line number              |
| <i>out</i> | <i>column</i>  | 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.

**31.130.4.59 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

|     |            |                                       |
|-----|------------|---------------------------------------|
|     | <i>pos</i> | character index                       |
| out | <i>X,Y</i> | pixel position of character on screen |

#### Returns

0 if character vertically out of view, X & Y positions otherwise

### 31.130.4.60 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

|                 |                                           |
|-----------------|-------------------------------------------|
| <i>startpos</i> | index of first character needing redraw   |
| <i>endpos</i>   | index after last character needing redraw |

### 31.130.4.61 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.

**31.130.4.62 resize()**

```
void Fl_Text_Display::resize (
 int X,
 int Y,
 int W,
 int H) [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](#).

**31.130.4.63 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

**31.130.4.64 scroll()**

```
void Fl_Text_Display::scroll (
 int topLineNum,
 int horizOffset)
```

Scrolls the current buffer to start at the specified line and column.

**Parameters**

|                    |                 |
|--------------------|-----------------|
| <i>topLineNum</i>  | top line number |
| <i>horizOffset</i> | column number   |

**Todo** Column numbers make little sense here.

**31.130.4.65 scroll\_()**

```
int Fl_Text_Display::scroll_
 int topLineNum,
 int horizOffset) [protected]
```

Scrolls the current buffer to start at the specified line and column.

**Parameters**

|                    |                 |
|--------------------|-----------------|
| <i>topLineNum</i>  | top line number |
| <i>horizOffset</i> | in pixels       |

**Returns**

0 if nothing changed, 1 if we scrolled

**31.130.4.66 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.

**31.130.4.67 scrollbar\_align() [1/2]**

[Fl\\_Align](#) `Fl_Text_Display::scrollbar_align () const [inline]`

Gets the scrollbar alignment type.

**Returns**

scrollbar alignment

**31.130.4.68 scrollbar\_align() [2/2]**

```
void Fl_Text_Display::scrollbar_align (
 Fl_Align a) [inline]
```

Sets the scrollbar alignment type.

**Parameters**

|   |                         |
|---|-------------------------|
| a | new scrollbar alignment |
|---|-------------------------|

**31.130.4.69 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\(int\)](#)

**31.130.4.70 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 | <i>newSize</i> | Sets the scrollbar size in pixels.<br>If 0 (default), scrollbar size tracks the global <a href="#">Fl::scrollbar_size()</a> |
|----|----------------|-----------------------------------------------------------------------------------------------------------------------------|

**See also**[Fl::scrollbar\\_size\(\)](#)**31.130.4.71 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.

**31.130.4.72 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

**31.130.4.73 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

**31.130.4.74 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**

|          |                      |
|----------|----------------------|
| <i>s</i> | the new shortcut key |
|----------|----------------------|

**31.130.4.75 show\_cursor()**

```
void Fl_Text_Display::show_cursor (
 int b = 1)
```

Shows the text cursor.

This function may trigger a redraw.

**Parameters**

|          |                                             |
|----------|---------------------------------------------|
| <i>b</i> | show(1) or hide(0) the text cursor (caret). |
|----------|---------------------------------------------|

**31.130.4.76 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\(\)](#)

**31.130.4.77 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**

|                            |                                       |
|----------------------------|---------------------------------------|
| <i>startPos</i>            | index to starting character           |
| <i>nLines</i>              | number of lines to skip ahead         |
| <i>startPosIsLineStart</i> | avoid scanning back to the line start |

**Returns**

new position as index

**31.130.4.78 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**

|               |                           |
|---------------|---------------------------|
| <i>string</i> | the text                  |
| <i>length</i> | number of bytes in string |
| <i>style</i>  | index into style table    |

**Returns**

width of text segment in pixels

**31.130.4.79 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

**31.130.4.80 textcolor() [2/2]**

```
void Fl_Text_Display::textcolor (
 Fl_Color n) [inline]
```

Sets the default color of text in the widget.

**Parameters**

|          |                |
|----------|----------------|
| <i>n</i> | new text color |
|----------|----------------|

**31.130.4.81 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

**31.130.4.82 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 |
|---|------------------------|

**31.130.4.83 textszie() [1/2]**

```
Fl_Fontsize Fl_Text_Display::textszie () const [inline]
```

Gets the default size of text in the widget.

**Returns**

current text height unless overridden by a style

**31.130.4.84 textszie() [2/2]**

```
void Fl_Text_Display::textszie (
 Fl_Fontsize s) [inline]
```

Sets the default size of text in the widget.

**Parameters**

|          |               |
|----------|---------------|
| <i>s</i> | new text size |
|----------|---------------|

**31.130.4.85 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.

**31.130.4.86 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**

|     |                      |                                                   |
|-----|----------------------|---------------------------------------------------|
|     | <i>pos</i>           | index into buffer of recent changes               |
|     | <i>charsInserted</i> | number of bytes(!) inserted                       |
|     | <i>charsDeleted</i>  | number of bytes(!) deleted                        |
|     | <i>linesInserted</i> | number of lines                                   |
|     | <i>linesDeleted</i>  | number of lines                                   |
| out | <i>scrolled</i>      | set to 1 if the text display needs to be scrolled |

**31.130.4.87 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.

**31.130.4.88 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**

|                   |                                     |
|-------------------|-------------------------------------|
| <i>visLineNum</i> | index of line in visible line array |
|-------------------|-------------------------------------|

**Returns**

number of bytes in this line

**31.130.4.89 word\_end()**

```
int Fl_Text_Display::word_end (
 int pos) const [inline]
```

Moves the insert position to the end of the current word.

**Parameters**

|            |                                 |
|------------|---------------------------------|
| <i>pos</i> | start calculation at this index |
|------------|---------------------------------|

**Returns**

index of first character after the end of the word

**31.130.4.90 word\_start()**

```
int Fl_Text_Display::word_start (
 int pos) const [inline]
```

Moves the insert position to the beginning of the current word.

**Parameters**

|            |                                 |
|------------|---------------------------------|
| <i>pos</i> | start calculation at this index |
|------------|---------------------------------|

**Returns**

beginning of the words

**31.130.4.91 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**

|                         |                                                                                                                                                                                                                                                                                                                                                                                         |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>wrap</code>       | new wrap mode (see above)                                                                                                                                                                                                                                                                                                                                                               |
| <code>wrapMargin</code> | in <code>WRAP_AT_COLUMN</code> 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 <code>WRAP_AT_PIXEL</code> mode, this is the pixel position. |

**31.130.4.92 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 `TextDEndOfLine` or `BufEndOfLine`, 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**

|                   |                                         |
|-------------------|-----------------------------------------|
| <i>lineEndPos</i> | index of character where the line wraps |
|-------------------|-----------------------------------------|

**Returns**

1 if a \n character causes the line wrap

**Todo** TextDEndOfLine and BufEndOfLine functions don't exist (nedit port?)

**31.130.4.93 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 involves character re-counting.

**Parameters**

|               |  |
|---------------|--|
| <i>row</i>    |  |
| <i>column</i> |  |

**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?

**31.130.4.94 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  | <i>buf</i>                         | The text buffer to operate on                                                                                                          |
| in  | <i>startPos</i>                    | Starting index position into the buffer                                                                                                |
| in  | <i>maxPos</i>                      | Maximum index position into the buffer we'll reach                                                                                     |
| in  | <i>maxLines</i>                    | Maximum number of lines we'll reach                                                                                                    |
| in  | <i>startPosIsLineStart</i>         | Flag indicating if startPos is start of line. (If set, prevents our having to find the line start)                                     |
| in  | <i>styleBufOffset</i>              | Offset index position into style buffer.                                                                                               |
| out | <i>retPos</i>                      | Position where counting ended. When counting lines, the position returned is the start of the line "maxLines" lines beyond "startPos". |
| out | <i>retLines</i>                    | Number of line breaks counted                                                                                                          |
| out | <i>retLineStart</i>                | Start of the line where counting ended                                                                                                 |
| out | <i>retLineEnd</i>                  | End position of the last line traversed                                                                                                |
| out | <i>countLastLineMissingNewLine</i> |                                                                                                                                        |

#### 31.130.4.95 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

|            |  |
|------------|--|
| <i>row</i> |  |
|------------|--|

**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?)

**31.130.4.96 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

**31.130.4.97 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              |
| <code>posType</code> | CURSOR_POS or CHARACTER_POS |

**Returns**

index into text buffer

### 31.130.4.98 xy\_to\_rowcol()

```
void Fl_Text_Display::xy_to_rowcol (
 int X,
 int Y,
 int * row,
 int * column,
 int postType = 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

|            |                   |                             |
|------------|-------------------|-----------------------------|
|            | <i>X,Y</i>        | pixel coordinates           |
| <i>out</i> | <i>row,column</i> | neares row and column       |
|            | <i>postType</i>   | CURSOR_POS or CHARACTER_POS |

The documentation for this class was generated from the following files:

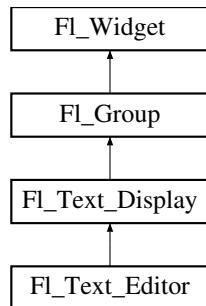
- Fl\_Text\_Display.H
- Fl\_Text\_Display.cxx

## 31.131 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, FI_Text_Editor *editor)`  
*Key function binding callback type.*

## 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, Key_Binding **list)`  
*Adds a key of state state with the function function to an arbitrary key binding list list.*
- `void add_key_binding (int key, int state, Key_Func f)`  
*Adds a key of state state with the function f.*
- `Key_Func bound_key_function (int key, int state, Key_Binding *list) const`  
*Returns the function associated with a key binding.*
- `Key_Func bound_key_function (int key, int state) const`  
*Returns the function associated with a key binding.*
- `void default_key_function (Key_Func f)`  
*Sets the default key function for unassigned keys.*
- `FI_Text_Editor (int X, int Y, int W, int H, const char *l=0)`  
*The constructor creates a new text editor widget.*
- `virtual int handle (int e)`  
*Event handling.*
- `void insert_mode (int b)`  
*Sets the current insert mode; if non-zero, new text is inserted before the current cursor position.*
- `int insert_mode ()`  
*Gets the current insert mode; if non-zero, new text is inserted before the current cursor position.*
- `void remove_all_key_bindings (Key_Binding **list)`  
*Removes all of the key bindings associated with the text editor or list.*
- `void remove_all_key_bindings ()`  
*Removes all of the key bindings associated with the text editor or list.*
- `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.*
- `void remove_key_binding (int key, int state)`  
*Removes the key binding associated with the key "key" of state "state".*
- `void tab_nav (int val)`  
*Enables or disables Tab key focus navigation.*
- `int tab_nav () const`  
*Check if Tab focus navigation is enabled.*

## Static Public Member Functions

- `static int kf_backspace (int c, FI_Text_Editor *e)`  
*Does a backspace for key 'c' in the current buffer of editor 'e'.*
- `static int kf_c_s_move (int c, FI_Text_Editor *e)`  
*Extends the current selection in the direction indicated by control key 'c' in editor 'e'.*
- `static int kf_copy (int c, FI_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, FI_Text_Editor *e)`

- static int `kf_cut` (int c, `FI_Text_Editor` \*e)
 

*Moves the current text cursor in the direction indicated by control key 'c' in editor 'e'.*
- static int `kf_default` (int c, `FI_Text_Editor` \*e)
 

*Does a cut of selected text in the current buffer of editor 'e'.*
- static int `kf_delete` (int c, `FI_Text_Editor` \*e)
 

*Inserts the text associated with key 'c' in editor 'e'.*
- static int `kf_down` (int c, `FI_Text_Editor` \*e)
 

*Does a delete of selected text or the current character in the current buffer of editor 'e'.*
- static int `kf_end` (int c, `FI_Text_Editor` \*e)
 

*Moves the text cursor one line down for editor 'e'.*
- static int `kf_enter` (int c, `FI_Text_Editor` \*e)
 

*Moves the text cursor to the end of the current line in editor 'e'.*
- static int `kf_home` (int c, `FI_Text_Editor` \*e)
 

*Inserts a newline for key 'c' at the current cursor position in editor 'e'.*
- static int `kf_ignore` (int c, `FI_Text_Editor` \*e)
 

*Moves the text cursor to the beginning of the current line in editor 'e'.*
- static int `kf_insert` (int c, `FI_Text_Editor` \*e)
 

*Ignores the key 'c' in editor 'e'.*
- static int `kf_left` (int c, `FI_Text_Editor` \*e)
 

*Toggles the insert mode for editor 'e'.*
- static int `kf_m_s_move` (int c, `FI_Text_Editor` \*e)
 

*Moves the text cursor one character to the left in editor 'e'.*
- static int `kf_meta_move` (int c, `FI_Text_Editor` \*e)
 

*Extends the current selection in the direction indicated by meta key 'c' in editor 'e'.*
- static int `kf_move` (int c, `FI_Text_Editor` \*e)
 

*Moves the current text cursor in the direction indicated by meta key 'c' in editor 'e'.*
- static int `kf_page_down` (int c, `FI_Text_Editor` \*e)
 

*Moves the text cursor down one page for editor 'e'.*
- static int `kf_page_up` (int c, `FI_Text_Editor` \*e)
 

*Moves the text cursor up one page for editor 'e'.*
- static int `kf_paste` (int c, `FI_Text_Editor` \*e)
 

*Does a paste of selected text in the current buffer of editor 'e'.*
- static int `kf_right` (int c, `FI_Text_Editor` \*e)
 

*Moves the text cursor one character to the right for editor 'e'.*
- static int `kf_select_all` (int c, `FI_Text_Editor` \*e)
 

*Selects all text in the current buffer in editor 'e'.*
- static int `kf_shift_move` (int c, `FI_Text_Editor` \*e)
 

*Extends the current selection in the direction of key 'c' in editor 'e'.*
- static int `kf_undo` (int c, `FI_Text_Editor` \*e)
 

*Undo last edit in the current buffer of editor 'e'.*
- static int `kf_up` (int c, `FI_Text_Editor` \*e)
 

*Moves the text cursor one line up for editor 'e'.*

## Protected Member Functions

- int `handle_key` ()
 

*Handles a key press in the editor.*
- void `maybe_do_callback` ()
 

*does or does not a callback according to `changed()` and `when()` settings*

## Static Protected Attributes

- static `Key_Binding * global_key_bindings`

*Global key binding list.*

## Additional Inherited Members

### 31.131.1 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.

### 31.131.2 Member Typedef Documentation

#### 31.131.2.1 Key\_Func

```
typedef int(* Fl_Text_Editor::Key_Func) (int key, Fl_Text_Editor *editor)
```

Key function binding callback type.

### 31.131.3 Constructor & Destructor Documentation

#### 31.131.3.1 Fl\_Text\_Editor()

```
Fl_Text_Editor::Fl_Text_Editor (
 int X,
 int Y,
 int W,
 int H,
 const char * l = 0)
```

The constructor creates a new text editor widget.

### 31.131.4 Member Function Documentation

#### 31.131.4.1 add\_default\_key\_bindings()

```
void Fl_Text_Editor::add_default_key_bindings (
 Key_Binding ** list)
```

Adds all of the default editor key bindings to the specified key binding list.

#### 31.131.4.2 add\_key\_binding() [1/2]

```
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](#).

#### 31.131.4.3 add\_key\_binding() [2/2]

```
void Fl_Text_Editor::add_key_binding (
 int key,
 int state,
 Key_Func f) [inline]
```

Adds a key of state state with the function f.

#### 31.131.4.4 bound\_key\_function() [1/2]

```
Fl_Text_Editor::Key_Func Fl_Text_Editor::bound_key_function (
 int key,
 int state,
 Key_Binding * list) const
```

Returns the function associated with a key binding.

#### 31.131.4.5 bound\_key\_function() [2/2]

```
Key_Func Fl_Text_Editor::bound_key_function (
 int key,
 int state) const [inline]
```

Returns the function associated with a key binding.

#### 31.131.4.6 default\_key\_function()

```
void Fl_Text_Editor::default_key_function (
 Key_Func f) [inline]
```

Sets the default key function for unassigned keys.

#### 31.131.4.7 insert\_mode() [1/2]

```
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.

#### 31.131.4.8 insert\_mode() [2/2]

```
int Fl_Text_Editor::insert_mode () [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.

#### 31.131.4.9 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.

#### 31.131.4.10 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\(\)](#).

### 31.131.4.11 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.

### 31.131.4.12 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<br>current page  |
| FL_Page_Down | -- moves the cursor to the beginning of the last line on the<br>current page |

### 31.131.4.13 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.

### 31.131.4.14 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.

**31.131.4.15 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.

**31.131.4.16 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.

**31.131.4.17 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.

**31.131.4.18 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.

**31.131.4.19 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.

**31.131.4.20 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.

**31.131.4.21 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.

**31.131.4.22 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.

**31.131.4.23 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\(\)](#).

## 31.131.4.24 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:

|                       |                                                                              |
|-----------------------|------------------------------------------------------------------------------|
| <code>FL_Up</code>    | -- moves cursor to the beginning of the <code>current</code> document        |
| <code>FL_Down</code>  | -- moves cursor to the <code>end</code> of the <code>current</code> document |
| <code>FL_Left</code>  | -- moves the cursor to the beginning of the <code>current</code> line        |
| <code>FL_Right</code> | -- moves the cursor to the <code>end</code> of the <code>current</code> line |

## 31.131.4.25 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:

|                           |                                                                              |
|---------------------------|------------------------------------------------------------------------------|
| <code>FL_Home</code>      | -- moves the cursor to the beginning of the <code>current</code> line        |
| <code>FL_End</code>       | -- moves the cursor to the <code>end</code> of the <code>current</code> line |
| <code>FL_Left</code>      | -- moves the cursor left one character                                       |
| <code>FL_Right</code>     | -- moves the cursor right one character                                      |
| <code>FL_Up</code>        | -- moves the cursor up one line                                              |
| <code>FL_Down</code>      | -- moves the cursor down one line                                            |
| <code>FL_Page_Up</code>   | -- moves the cursor up one page                                              |
| <code>FL_Page_Down</code> | -- moves the cursor down one page                                            |

## 31.131.4.26 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.

## 31.131.4.27 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.

#### 31.131.4.28 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.

#### 31.131.4.29 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.

#### 31.131.4.30 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.

#### 31.131.4.31 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\(\)](#)

#### 31.131.4.32 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.

**31.131.4.33 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.

**31.131.4.34 remove\_all\_key\_bindings() [1/2]**

```
void Fl_Text_Editor::remove_all_key_bindings (
 Key_Binding ** list)
```

Removes all of the key bindings associated with the text editor or list.

**31.131.4.35 remove\_all\_key\_bindings() [2/2]**

```
void Fl_Text_Editor::remove_all_key_bindings () [inline]
```

Removes all of the key bindings associated with the text editor or list.

**31.131.4.36 remove\_key\_binding() [1/2]**

```
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`.

**31.131.4.37 remove\_key\_binding() [2/2]**

```
void Fl_Text_Editor::remove_key_binding (
 int key,
 int state) [inline]
```

Removes the key binding associated with the key "key" of state "state".

**31.131.4.38 tab\_nav() [1/2]**

```
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 | <i>val</i> | If <i>val</i> is 0, Tab inserts a tab character (default).<br>If <i>val</i> is 1, Tab navigates widget focus. |
|----|------------|---------------------------------------------------------------------------------------------------------------|

**See also**

[tab\\_nav\(\)](#), [Fl::OPTION\\_ARROW\\_FOCUS](#).

**Version**

1.3.4 ABI feature

**31.131.4.39 tab\_nav() [2/2]**

```
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

**31.131.5 Member Data Documentation**

### 31.131.5.1 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](#)

## 31.132 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](#)  
*Returns the status and the positions of this selection.*
- bool [selected \(\) const](#)  
*Returns true if any text is selected.*
- void [selected \(bool b\)](#)  
*Modifies the 'selected' flag.*
- 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 `FI_Text_Buffer`

### 31.132.1 Detailed Description

This is an internal class for `FI_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.

### 31.132.2 Member Function Documentation

### 31.132.2.1 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.

### 31.132.2.2 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.

### 31.132.2.3 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

### 31.132.2.4 position()

```
int Fl_Text_Selection::position (
 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**

|                       |                                                           |
|-----------------------|-----------------------------------------------------------|
| <code>startpos</code> | return byte offset to first selected character            |
| <code>endpos</code>   | return byte offset pointing after last selected character |

**Returns**

whether the selection is active ([selected\(\)](#)) or not

**Return values**

|                |                 |
|----------------|-----------------|
| <code>0</code> | if not selected |
| <code>1</code> | if selected     |

**See also**

[selected\(\)](#), [start\(\)](#), [end\(\)](#)

**31.132.2.5 selected() [1/2]**

```
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.

**31.132.2.6 selected() [2/2]**

```
void Fl_Text_Selection::selected (
 bool b) [inline]
```

Modifies the 'selected' flag.

**Parameters**

|                |          |
|----------------|----------|
| <code>b</code> | new flag |
|----------------|----------|

### 31.132.2.7 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 | <code>startpos</code> | byte offset to first selected character            |
| in | <code>endpos</code>   | byte offset pointing after last selected character |

### 31.132.2.8 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.

### 31.132.2.9 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

|                  |                                                           |
|------------------|-----------------------------------------------------------|
| <i>pos</i>       | byte offset into text buffer at which the change occurred |
| <i>nDeleted</i>  | number of bytes deleted from the buffer                   |
| <i>nInserted</i> | number of bytes inserted into the buffer                  |

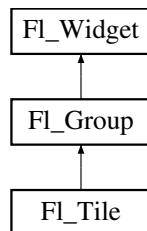
The documentation for this class was generated from the following files:

- Fl\_Text\_Buffer.H
- Fl\_Text\_Buffer.cxx

## 31.133 Fl\_Tile Class Reference

The [Fl\\_Tile](#) class lets you resize its children by dragging the border between them.

Inheritance diagram for Fl\_Tile:



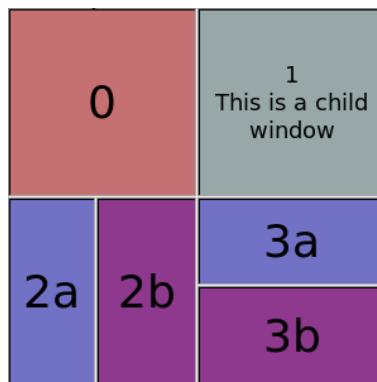
#### Public Member Functions

- [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)  
*Handles the specified event.*
- void [position](#) (int oldx, int oldy, int newx, int newy)  
*Drags the intersection at (oldx,oldy) to (newx,newy).*
- void [resize](#) (int X, int Y, int W, int H)  
*Resizes the Fl\_Tile widget and its children.*

#### Additional Inherited Members

##### 31.133.1 Detailed Description

The [Fl\\_Tile](#) class lets you resize its children by dragging the border between them.

Figure 31.50 `FL_Tile`

For the tiling to work correctly, the children of an `FL_Tile` must cover the entire area of the widget, but 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.

`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`.

### 31.133.2 Constructor & Destructor Documentation

#### 31.133.2.1 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 boxtyle 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](#)

### 31.133.3 Member Function Documentation

#### 31.133.3.1 handle()

```
int Fl_Tile::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.

#### 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\\_Group](#).

### 31.133.3.2 position()

```
void Fl_Tile::position (
 int oldx,
 int oldy,
 int newx,
 int newy)
```

Drags the intersection at (oldx,oldy) to (newx,newy).

This redraws all the necessary children.

Pass zero as oldx or oldy to disable drag in that direction.

### 31.133.3.3 resize()

```
void Fl_Tile::resize (
 int X,
 int Y,
 int W,
 int H) [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.

Enlarging works by just 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](#).

The documentation for this class was generated from the following files:

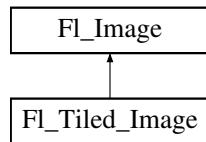
- [Fl\\_Tile.H](#)
- [Fl\\_Tile.cxx](#)

## 31.134 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

- virtual void [color\\_average \(Fl\\_Color c, float i\)](#)  
*The [color\\_average\(\)](#) method averages the colors in the image with the FLTK color value c.*
- virtual [Fl\\_Image \\* copy \(int W, int H\)](#)  
*Creates a resized copy of the specified image.*
- [Fl\\_Image \\* copy \(\)](#)
- virtual void [desaturate \(\)](#)  
*The [desaturate\(\)](#) method converts an image to grayscale.*
- virtual void [draw \(int X, int Y, int W, int H, int cx=0, int cy=0\)](#)  
*Draws a tiled image.*
- void [draw \(int X, int Y\)](#)
- [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 \(\)](#)  
*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.*

### Protected Attributes

- int [alloc\\_image\\_](#)
- [Fl\\_Image \\* image\\_](#)

### Additional Inherited Members

#### 31.134.1 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.

## 31.134.2 Constructor & Destructor Documentation

### 31.134.2.1 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\(const char \\*\)](#).

## 31.134.3 Member Function Documentation

### 31.134.3.1 color\_average()

```
void Fl_Tiled_Image::color_average (
 Fl_Color c,
 float i) [virtual]
```

The [color\\_average\(\)](#) method averages the colors in the image with the FLTK color value c.

The i 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 before changes are applied, to avoid modifying the original image.

Reimplemented from [Fl\\_Image](#).

## 31.134.3.2 copy()

```
Fl_Image * Fl_Tiled_Image::copy (
 int W,
 int H) [virtual]
```

Creates a resized copy of the specified image.

The image should be deleted (or in the case of [Fl\\_Shared\\_Image](#), released) when you are done with it.

#### Parameters

|            |                                               |
|------------|-----------------------------------------------|
| <i>W,H</i> | width and height of the returned copied image |
|------------|-----------------------------------------------|

Reimplemented from [Fl\\_Image](#).

## 31.134.3.3 desaturate()

```
void Fl_Tiled_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 before changes are applied, to avoid modifying the original image.

Reimplemented from [Fl\\_Image](#).

## 31.134.3.4 draw()

```
void Fl_Tiled_Image::draw (
 int X,
 int Y,
 int W,
 int H,
 int cx = 0,
 int cy = 0) [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  $\geq 0$  (negative values are ignored). If one of the values is greater than the image width or height resp. (*cx*  $\geq$  [image\(\)->w\(\)](#) or *cy*  $\geq$  [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:

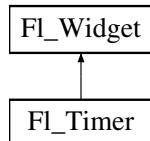
- `Fl_Tiled_Image.H`
- `Fl_Tiled_Image.cxx`

### 31.135 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 *)`  
*Creates a new `Fl_Timer` widget using the given type, position, size, and label string.*
- `int handle (int)`  
*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.*
- `void value (double)`  
*Sets the current timer value.*
- `double value () const`  
*See void `Fl_Timer::value(double)`*
- `~Fl_Timer ()`  
*Destroys the timer and removes the timeout.*

## Protected Member Functions

- void [draw \(\)](#)  
*Draws the widget.*

## Additional Inherited Members

### 31.135.1 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.

### 31.135.2 Constructor & Destructor Documentation

#### 31.135.2.1 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.

### 31.135.3 Member Function Documentation

#### 31.135.3.1 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\(\)](#).

### 31.135.3.2 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\(\)](#).

### 31.135.3.3 draw()

```
void Fl_Timer::draw () [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](#).

### 31.135.3.4 handle()

```
int Fl_Timer::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.

#### Parameters

|    |                    |                            |
|----|--------------------|----------------------------|
| in | <code>event</code> | 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](#).

#### 31.135.3.5 suspended() [1/2]

```
char Fl_Timer::suspended () const [inline]
```

Gets or sets whether the timer is suspended.

#### 31.135.3.6 suspended() [2/2]

```
void Fl_Timer::suspended (char d)
```

Gets or sets whether the timer is suspended.

The documentation for this class was generated from the following files:

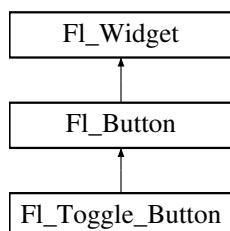
- [Fl\\_Timer.H](#)
- [forms\\_timer.cxx](#)

## 31.136 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.*

## Additional Inherited Members

### 31.136.1 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\(\)](#).

### 31.136.2 Constructor & Destructor Documentation

#### 31.136.2.1 [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](#)

## 31.137 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 [FI\\_Fontsize size \(\)](#)  
*Gets the size of the tooltip text.*
- static void [size \(FI\\_Fontsize s\)](#)  
*Sets the size of the tooltip text.*
- static [FI\\_Color textcolor \(\)](#)  
*Gets the color of the text in the tooltip.*
- static void [textcolor \(FI\\_Color c\)](#)  
*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](#) )([FI\\_Widget](#) \*w) = nothing
- static void(\* [exit](#) )([FI\\_Widget](#) \*w) = nothing

## Friends

- class [FI\\_TooltipBox](#)
- void [FI\\_Widget::copy\\_tooltip](#) (const char \*)
- void [FI\\_Widget::tooltip](#) (const char \*)

### 31.137.1 Detailed Description

The [FI\\_Tooltip](#) class provides tooltip support for all FLTK widgets.

It contains only static methods.

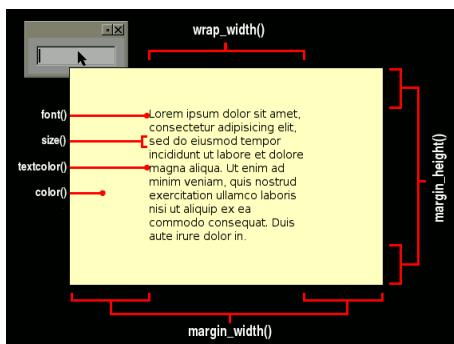


Figure 31.51 FI\_Tooltip Options

### 31.137.2 Member Function Documentation

**31.137.2.1 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.

**31.137.2.2 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.

**31.137.2.3 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.

**31.137.2.4 delay() [1/2]**

```
static float Fl_Tooltip::delay () [inline], [static]
```

Gets the tooltip delay.

The default delay is 1.0 seconds.

**31.137.2.5 delay() [2/2]**

```
static void Fl_Tooltip::delay (
 float f) [inline], [static]
```

Sets the tooltip delay.

The default delay is 1.0 seconds.

**31.137.2.6 disable()**

```
static void Fl_Tooltip::disable () [inline], [static]
```

Same as enable(0), disables tooltips on all widgets.

**31.137.2.7 enable()**

```
static void Fl_Tooltip::enable (
 int b = 1) [inline], [static]
```

Enables tooltips on all widgets (or disables if *b* is false).

**31.137.2.8 enabled()**

```
static int Fl_Tooltip::enabled () [inline], [static]
```

Returns non-zero if tooltips are enabled.

**31.137.2.9 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).

**31.137.2.10 font() [1/2]**

```
static Fl_Font Fl_Tooltip::font () [inline], [static]
```

Gets the typeface for the tooltip text.

**31.137.2.11 font() [2/2]**

```
static void Fl_Tooltip::font (
 Fl_Font i) [inline], [static]
```

Sets the typeface for the tooltip text.

**31.137.2.12 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.

**31.137.2.13 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.

**31.137.2.14 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.

**31.137.2.15 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.

**31.137.2.16 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.

**31.137.2.17 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.

**31.137.2.18 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.

**31.137.2.19 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.

**31.137.2.20 size()** [1/2]

```
static Fl_Fontsize Fl_Tooltip::size () [inline], [static]
```

Gets the size of the tooltip text.

**31.137.2.21 size()** [2/2]

```
static void Fl_Tooltip::size (
 Fl_Fontsize s) [inline], [static]
```

Sets the size of the tooltip text.

**31.137.2.22 textcolor()** [1/2]

```
static Fl_Color Fl_Tooltip::textcolor () [inline], [static]
```

Gets the color of the text in the tooltip.

The default is black.

**31.137.2.23 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.

## 31.137.2.24 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.

## 31.137.2.25 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:

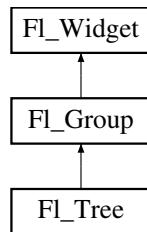
- Fl\_Tooltip.H
- [Fl.hxx](#)
- Fl\_Tooltip.hxx

## 31.138 Fl\_Tree Class Reference

Tree widget.

```
#include <Fl_Tree.h>
```

Inheritance diagram for Fl\_Tree:



## Public Member Functions

- `FI_Tree_Item * add (const char *path, FI_Tree_Item *newitem=0)`  
*Adds a new item, given a menu style 'path'.*
- `FI_Tree_Item * add (FI_Tree_Item *parent_item, const char *name)`  
*Add a new child item labeled 'name' to the specified 'parent\_item'.*
- `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.*
- `void callback_item (FI_Tree_Item *item)`  
*Sets the item that was changed for this callback.*
- `FI_Tree_Item * callback_item ()`  
*Gets the item that caused the callback.*
- `void callback_reason (FI_Tree_Reason reason)`  
*Sets the reason for this callback.*
- `FI_Tree_Reason callback_reason () const`  
*Gets the reason for this callback.*
- `void clear ()`  
*Clear the entire tree's children, including the root.*
- `void clear_children (FI_Tree_Item *item)`  
*Clear all the children for 'item'.*
- `int close (FI_Tree_Item *item, int docallback=1)`  
*Closes the specified 'item'.*
- `int close (const char *path, int docallback=1)`  
*Closes the item specified by 'path'.*
- `FI_Image * closeicon () const`  
*Returns the icon to be used as the 'close' icon.*
- `void closeicon (FI_Image *val)`  
*Sets the icon to be used as the 'close' icon.*
- `FI_Color connectorcolor () const`  
*Get the connector color used for tree connection lines.*
- `void connectorcolor (FI_Color val)`  
*Set the connector color used for tree connection lines.*
- `FI_Tree_Connector connectorstyle () const`  
*Returns the line drawing style for inter-connecting items.*
- `void connectorstyle (FI_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 (FI_Tree_Item *item, int docallback=1)`  
*Deselect the specified item.*
- `int deselect (const char *path, int docallback=1)`  
*Deselect an item specified by 'path'.*
- `int deselect_all (FI_Tree_Item *item=0, int docallback=1)`  
*Deselect 'item' and all its children.*
- `void display (FI_Tree_Item *item)`  
*Displays 'item', scrolling the tree as necessary.*
- `int displayed (FI_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'.*
- const `FL_Tree_Item * find_clicked (int yonly=0) const`  
*Find the item that was last clicked on.*
- `FL_Tree_Item * find_clicked (int yonly=0)`  
*Non-const version of `FL_Tree::find_clicked(int yonly)` const.*
- `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)`  
*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 (FL_Tree_Item *item) const`  
*See if the specified 'item' is closed.*
- int `is_close (const char *path) const`  
*See if item specified by 'path' is closed.*
- int `is_hscroll_visible () const`  
*See if the horizontal scrollbar is currently visible.*
- int `is_open (FL_Tree_Item *item) const`  
*See if 'item' is open.*
- int `is_open (const char *path) const`  
*See if item specified by 'path' is open.*
- int `is_scrollbar (FL_Widget *w)`  
*See if widget 'w' is one of the `FL_Tree` widget's scrollbars.*
- int `is_selected (FL_Tree_Item *item) const`

- `int is_selected (const char *path)`

*See if the specified 'item' is selected.*
- `int is_vscroll_visible () const`

*See if item specified by 'path' is selected.*
- `FI_Tree_Item * item_clicked ()`

*Return the item that was last clicked.*
- `FI_Tree_Item_Draw_Mode item_draw_mode () const`

*Get the 'item draw mode' used for the tree.*
- `void item_draw_mode (FI_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'.*
- `FI_Color item_labelbgcolor (void) const`

*Get the default label background color used for creating new items.*
- `void item_labelbgcolor (FI_Color val)`

*Set the default label background color used for creating new items.*
- `FI_Color item_labelfgcolor (void) const`

*Get the default label foreground color used for creating new items.*
- `void item_labelfgcolor (FI_Color val)`

*Set the default label foreground color used for creating new items.*
- `FI_Font item_labelfont () const`

*Get the default font face used for creating new items.*
- `void item_labelfont (FI_Font val)`

*Set the default font face used for creating new items.*
- `FI_Fontsize item_labelsize () const`

*Get the default label fontsize used for creating new items.*
- `void item_labelsize (FI_Fontsize val)`

*Set the default label font size used for creating new items.*
- `int item.pathname (char *pathname, int pathnameLEN, const FI_Tree_Item *item) const`

*Return 'pathname' of size 'pathnameLEN' for the specified 'item'.*
- `FI_Tree_Item_Reselect_Mode item_reselect_mode () const`

*Returns the current item re-selection mode.*
- `void item_reselect_mode (FI_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.*
- `FI_Tree_Item * last ()`

*Returns the last item in the tree.*
- `FI_Tree_Item * last_selected_item ()`

*Returns the last selected item in the tree.*
- `FI_Tree_Item * last_visible ()`

*Returns the last open(), visible item in the tree.*
- `FI_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 **FI\_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.*
- **FI\_Tree\_Item** \* **next** (**FI\_Tree\_Item** \*item=0)  
*Return the next item after 'item', or 0 if no more items.*
- **FI\_Tree\_Item** \* **next\_item** (**FI\_Tree\_Item** \*item, int dir=**FL\_Down**, bool **visible**=false)  
*Returns next item after 'item' in direction 'dir' depending on 'visible'.*
- **FI\_Tree\_Item** \* **next\_selected\_item** (**FI\_Tree\_Item** \*item=0, int dir=**FL\_Down**)  
*Returns the next selected item above or below 'item', depending on 'dir'.*
- **FI\_Tree\_Item** \* **next\_visible\_item** (**FI\_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** (**FI\_Tree\_Item** \*item, int docallback=1)  
*Open the specified 'item'.*
- int **open** (const char \*path, int docallback=1)  
*Opens the item specified by 'path'.*
- void **open\_toggle** (**FI\_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.*
- **FI\_Image** \* **openicon** () const  
*Returns the icon to be used as the 'open' icon.*
- void **openicon** (**FI\_Image** \*val)  
*Sets the icon to be used as the 'open' icon.*
- const **FI\_Tree\_Prefs** & **prefs** () const
- **FI\_Tree\_Item** \* **prev** (**FI\_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** (**FI\_Tree\_Item** \*item)  
*Remove the specified 'item' from the tree.*
- void **resize** (int, int, int, int)  
*Resizes the **FI\_Group** widget and all of its children.*
- **FI\_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 (FL_Tree_Item *item, int docallback=1)`

*Select the specified 'item'.*
  - int `select (const char *path, int docallback=1)`

*Select the item specified by 'path'.*
  - 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, 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 (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_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.

## 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 \(\)](#)  
Standard FLTK [draw\(\)](#) method, handles drawing the tree widget.
- void [item\\_clicked \(Fl\\_Tree\\_Item \\*val\)](#)  
Set the item that was last clicked.

## 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

### 31.138.1 Detailed Description

Tree widget.



**Figure 31.52 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:

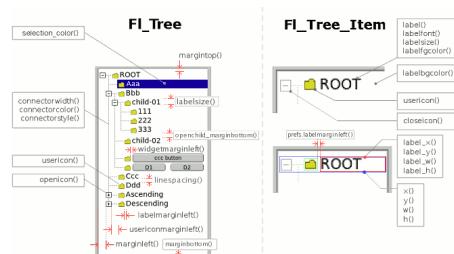


Figure 31.53 Fl\_Tree elements

The following shows the protected dimension variables 'tree inner' (tix..) and 'tree outer' (tox..):

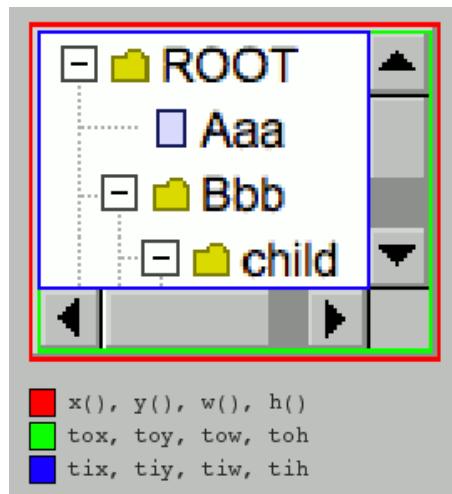


Figure 31.54 FL\_Tree inner/outer dimensions

#### KEYBOARD BINDINGS

The following table lists keyboard bindings for navigating the tree:

| Keyboard               | FL_TREE_SELECT_←<br><b>MULTI</b> | FL_TREE_SELECT_←<br><b>SINGLE</b> | FL_TREE_SELECT_←<br><b>NONE</b> |
|------------------------|----------------------------------|-----------------------------------|---------------------------------|
| 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                    |

#### 31.138.2 Member Function Documentation

## 31.138.2.1 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 | <i>path</i> | The path to the item, e.g. "Flintstone/Fred".                                                            |
| in | <i>item</i> | 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

## 31.138.2.2 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 | <i>parent_item</i> | The parent item the new child item will be added to. Must not be NULL. |
| in | <i>name</i>        | The label for the new item                                             |

#### Returns

The new item added.

**Version**

1.3.0 release

**31.138.2.3 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

**31.138.2.4 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)

## 31.138.2.5 callback\_item() [1/2]

```
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.

## 31.138.2.6 callback\_item() [2/2]

```
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.

## 31.138.2.7 callback\_reason() [1/2]

```
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.

## 31.138.2.8 callback\_reason() [2/2]

```
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

### 31.138.2.9 clear()

```
void Fl_Tree::clear ()
```

Clear the entire tree's children, including the root.

The tree will be left completely empty.

### 31.138.2.10 clear\_children()

```
void Fl_Tree::clear_children (
 Fl_Tree_Item * item)
```

Clear all the children for 'item'.

Item may not be NULL.

### 31.138.2.11 close() [1/2]

```
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 | <i>item</i>       | – the item to be closed. Must not be NULL.                                                                                                                                                                                                                                                                                                             |
| in | <i>docallback</i> | <p>– A flag that determines if the <a href="#">callback()</a> is invoked or not:</p> <ul style="list-style-type: none"> <li>• 0 - <a href="#">callback()</a> is not invoked</li> <li>• 1 - <a href="#">callback()</a> is invoked if item changed (default), <a href="#">callback_reason()</a> will be <a href="#">FL_TREE_REASON_CLOSED</a></li> </ul> |

#### Returns

- 1 – item was closed
- 0 – item was already closed, no change

#### See also

[open\(\)](#), [close\(\)](#), [is\\_open\(\)](#), [is\\_close\(\)](#), [callback\\_item\(\)](#), [callback\\_reason\(\)](#)

## 31.138.2.12 close() [2/2]

```
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\V25\V2010").

The callback can use [callback\\_item\(\)](#) and [callback\\_reason\(\)](#) respectively to determine the item changed and the reason the callback was called.

#### Parameters

|    |                   |                                                                                                                                                                                                                                                                                                                                              |
|----|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| in | <i>path</i>       | – the tree item's pathname (e.g. "Flintstones/Fred")                                                                                                                                                                                                                                                                                         |
| in | <i>docallback</i> | – A flag that determines if the <a href="#">callback()</a> is invoked or not: <ul style="list-style-type: none"> <li>• 0 - <a href="#">callback()</a> is not invoked</li> <li>• 1 - <a href="#">callback()</a> is invoked if item changed (default), <a href="#">callback_reason()</a> will be <code>FL_TREE_REASON_CLOSED</code></li> </ul> |

#### 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\(\)](#)

## 31.138.2.13 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.

## 31.138.2.14 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**

|                 |                  |                                                       |
|-----------------|------------------|-------------------------------------------------------|
| <code>in</code> | <code>val</code> | – The new image, or zero to use the default [-] icon. |
|-----------------|------------------|-------------------------------------------------------|

**31.138.2.15 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.

**31.138.2.16 deselect() [1/2]**

```
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**

|                 |                         |                                                                                                                                                                                                                                                                                                                                                                |
|-----------------|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>in</code> | <code>item</code>       | – the item to be deselected. Must not be NULL.                                                                                                                                                                                                                                                                                                                 |
| <code>in</code> | <code>docallback</code> | – A flag that determines if the <a href="#">callback()</a> is invoked or not: <ul style="list-style-type: none"> <li>• 0 - the <a href="#">callback()</a> is not invoked</li> <li>• 1 - the <a href="#">callback()</a> is invoked if item changed state (default), <a href="#">callback_reason()</a> will be <code>FL_TREE_REASON_DESELECTED</code></li> </ul> |

**Returns**

- 0 - item was already deselected, no change was made
- 1 - item's state was changed

**31.138.2.17 deselect() [2/2]**

```
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 | <i>path</i>       | – the tree item's pathname (e.g. "Flintstones/Fred")                                                                                                                                                                                                                                                                                                                  |
| in | <i>docallback</i> | <p>– A flag that determines if the <a href="#">callback()</a> is invoked or not:</p> <ul style="list-style-type: none"> <li>• 0 - the <a href="#">callback()</a> is not invoked</li> <li>• 1 - the <a href="#">callback()</a> is invoked if item changed state (default), <a href="#">callback_reason()</a> will be <code>FL_TREE_REASON_DESELECTED</code></li> </ul> |

#### Returns

- 1 - OK: item's state was changed
- 0 - OK: item was already deselected, no change was made
- -1 - ERROR: item was not found

### 31.138.2.18 `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 | <i>item</i>       | The item that will be deselected (along with all its children). If NULL, <a href="#">first()</a> is used.                                                                                                                                                                                                                                                                        |
| in | <i>docallback</i> | <p>– A flag that determines if the <a href="#">callback()</a> is invoked or not:</p> <ul style="list-style-type: none"> <li>• 0 - the <a href="#">callback()</a> is not invoked</li> <li>• 1 - the <a href="#">callback()</a> is invoked for each item that changed state (default), <a href="#">callback_reason()</a> will be <code>FL_TREE_REASON_DESELECTED</code></li> </ul> |

**Returns**

Count of how many items were actually changed to the deselected state.

**31.138.2.19 display()**

```
void Fl_Tree::display (
 Fl_Tree_Item * item)
```

Displays 'item', scrolling the tree as necessary.

**Parameters**

|    |             |                                                                     |
|----|-------------|---------------------------------------------------------------------|
| in | <i>item</i> | The item to be displayed. If NULL, <a href="#">first()</a> is used. |
|----|-------------|---------------------------------------------------------------------|

**31.138.2.20 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 | <i>item</i> | The item to be checked. If NULL, <a href="#">first()</a> is used. |
|----|-------------|-------------------------------------------------------------------|

**Returns**

1 if displayed, 0 if scrolled off screen or no items are in tree.

**31.138.2.21 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 | <i>from</i>    | Starting item                                                                                        |
| in | <i>to</i>      | Ending item                                                                                          |
| in | <i>val</i>     | Select or deselect items (0=deselect, 1=select, 2=toggle)                                            |
| in | <i>visible</i> | true=affect only <code>open()</code> , visible items,<br>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

### 31.138.2.22 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*,int,bool)` should be used.

Handles calling `redraw()` if anything changed.

#### Parameters

|    |                |                                                                                                      |
|----|----------------|------------------------------------------------------------------------------------------------------|
| in | <i>from</i>    | Starting item                                                                                        |
| in | <i>to</i>      | Ending item                                                                                          |
| in | <i>dir</i>     | Direction to extend selection (FL_Up or FL_Down)                                                     |
| in | <i>val</i>     | 0=deselect, 1=select, 2=toggle                                                                       |
| in | <i>visible</i> | true=affect only <code>open()</code> , visible items,<br>false=affect open or closed items (default) |

**Returns**

The number of items whose selection states were changed, if any.

**Version**

1.3.3

**31.138.2.23 find\_clicked()**

```
const F1_Tree_Item * F1_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 [F1\\_Tree](#), and are receiving events before [F1\\_Tree](#) has been able to process and update [callback\\_item\(\)](#).

**Parameters**

|    |              |                                                                                              |
|----|--------------|----------------------------------------------------------------------------------------------|
| in | <i>yonly</i> | – 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

**31.138.2.24 find\_item()**

```
const F1_Tree_Item * F1_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 | <i>path</i> | – the tree item's pathname to be found (e.g. "Flintstones/Fred") |
|----|-------------|------------------------------------------------------------------|

**Returns**

The item, or NULL if not found.

**See also**

[item.pathname\(\)](#)

**31.138.2.25 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\(\)](#)

**31.138.2.26 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\(\)](#)

### 31.138.2.27 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.

### 31.138.2.28 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

### 31.138.2.29 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**

|                  |                        |                                       |
|------------------|------------------------|---------------------------------------|
| <code>out</code> | <code>ret_items</code> | 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

**31.138.2.30 handle()**

```
int Fl_Tree::handle (
 int e) [virtual]
```

Standard FLTK event handler for this widget.

**Todo** add [Fl\\_Widget\\_Tracker](#) (see [Fl\\_Browser\\_.cxx::handle\(\)](#))

Reimplemented from [Fl\\_Group](#).

**31.138.2.31 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.

**31.138.2.32 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**

|                 |                  |                                                          |
|-----------------|------------------|----------------------------------------------------------|
| <code>in</code> | <code>pos</code> | 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.

**31.138.2.33 insert()**

```
F1_Tree_Item * F1_Tree::insert (
 F1_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
F1_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**

|                 |                   |                                                               |
|-----------------|-------------------|---------------------------------------------------------------|
| <code>in</code> | <code>item</code> | The existing item to insert new child into. Must not be NULL. |
| <code>in</code> | <code>name</code> | The label for the new item                                    |
| <code>in</code> | <code>pos</code>  | The position of the new item in the child list                |

**Returns**

The new item added.

**See also**

[insert\\_above\(\)](#)

**31.138.2.34 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 | <i>above</i> | – the item above which to insert the new item. Must not be NULL. |
| in | <i>name</i>  | – the name of the new item                                       |

**Returns**

The new item added, or 0 if 'above' could not be found.

**See also**

[insert\(\)](#)

**31.138.2.35 is\_close()** [1/2]

```
int Fl_Tree::is_close (
 Fl_Tree_Item * item) const
```

See if the specified 'item' is closed.

**Parameters**

|    |             |                                            |
|----|-------------|--------------------------------------------|
| in | <i>item</i> | – the item to be tested. Must not be NULL. |
|----|-------------|--------------------------------------------|

**Returns**

- 1 : item is closed
- 0 : item is open

**31.138.2.36 is\_close()** [2/2]

```
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 | <i>path</i> | – 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

**31.138.2.37 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.

**31.138.2.38 is\_open() [1/2]**

```
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 | <i>item</i> | – the item to be tested. Must not be NULL. |
|----|-------------|--------------------------------------------|

**Returns**

- 1 : item is open
- 0 : item is closed

**31.138.2.39 is\_open() [2/2]**

```
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\\V25\\V2010").

Items that are 'open' are themselves not necessarily visible; one of the item's parents might be closed.

**Parameters**

|    |             |                                                      |
|----|-------------|------------------------------------------------------|
| in | <i>path</i> | – 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\(\)](#)

### 31.138.2.40 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

### 31.138.2.41 is\_selected() [1/2]

```
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

### 31.138.2.42 is\_selected() [2/2]

```
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 | <i>path</i> | – the tree item's pathname (e.g. "Flintstones/Fred") |
|----|-------------|------------------------------------------------------|

#### Returns

- 1 : item selected
- 0 : item deselected
- -1 : item was not found

### 31.138.2.43 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.

### 31.138.2.44 item\_clicked() [1/2]

```
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.

**31.138.2.45 item\_clicked() [2/2]**

```
F1_Tree_Item * F1_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.

**31.138.2.46 item\_draw\_mode() [1/3]**

```
F1_Tree_Item_Draw_Mode F1_Tree::item_draw_mode () const
```

Get the 'item draw mode' used for the tree.

**Version**

1.3.1 ABI feature

**31.138.2.47 item\_draw\_mode() [2/3]**

```
void F1_Tree::item_draw_mode (
 F1_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 [F1\\_Tree\\_Item\\_Draw\\_Mode](#) for possible values.

**Version**

1.3.1 ABI feature

**31.138.2.48 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

**31.138.2.49 item\_labelbgcolor() [1/2]**

```
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'.

**31.138.2.50 item\_labelbgcolor() [2/2]**

```
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\(Fl\\_Color\)](#)

**31.138.2.51 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\(Fl\\_Color\)](#)

**31.138.2.52 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\(Fl\\_Font\)](#)

### 31.138.2.53 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\(Fl\\_Fontsize\)](#)

### 31.138.2.54 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 | <i>pathname</i>    | The string to use to return the pathname                             |
| in  | <i>pathnamelen</i> | The maximum length of the string (including NULL). Must not be zero. |
| in  | <i>item</i>        | 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\(\)](#)

### 31.138.2.55 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

**31.138.2.56 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

**31.138.2.57 labelmarginleft() [1/2]**

```
int Fl_Tree::labelmarginleft () const
```

Get the amount of white space (in pixels) that should appear to the left of the label text.

**31.138.2.58 labelmarginleft() [2/2]**

```
void Fl_Tree::labelmarginleft (
 int val)
```

Set the amount of white space (in pixels) that should appear to the left of the label text.

**31.138.2.59 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\(\)](#)

### 31.138.2.60 last\_selected\_item()

```
F1_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

### 31.138.2.61 last\_visible()

```
F1_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.

### 31.138.2.62 last\_visible\_item()

```
F1_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

### 31.138.2.63 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 | <i>prefs</i> | the <a href="#">Fl_Preferences</a> database |
|----|--------------|---------------------------------------------|

### 31.138.2.64 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 | <i>item</i> | 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\(\)](#)

### 31.138.2.65 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:   | <code>last_visible_item()</code>  | <code>last()</code>  |
| dir=FL_Down: | <code>first_visible_item()</code> | <code>first()</code> |

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 | <code>item</code>    | The item to use to find the next item. If NULL, returns 0.                                                          |
| in | <code>dir</code>     | Can be <code>FL_Up</code> or <code>FL_Down</code> (default= <code>FL_Down</code> or 'next')                         |
| in | <code>visible</code> | true= <code>return only open()</code> , visible items,<br>false= <code>return open or closed items (default)</code> |

#### 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

## 31.138.2.66 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 | <i>item</i> | The item above or below which we'll find the next selected item. If NULL, <a href="#">first()</a> is used if FL_Down, <a href="#">last()</a> if FL_Up. (default=NULL) |
| in | <i>dir</i>  | 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

## 31.138.2.67 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 | <i>item</i> | The item above/below which we'll find the next visible item |
| in | <i>dir</i>  | 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

**31.138.2.68 open() [1/2]**

```
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 | <i>item</i>       | – the item to be opened. Must not be NULL.                                                                                                                                                                                                                                                                                      |
| in | <i>docallback</i> | – A flag that determines if the <a href="#">callback()</a> is invoked or not: <ul style="list-style-type: none"> <li>• 0 - <a href="#">callback()</a> is not invoked</li> <li>• 1 - <a href="#">callback()</a> is invoked if item changed (default), <a href="#">callback_reason()</a> will be FL_TREE_REASON_OPENED</li> </ul> |

**Returns**

- 1 – item was opened
- 0 – item was already open, no change

**See also**

[open\(\)](#), [close\(\)](#), [is\\_open\(\)](#), [is\\_close\(\)](#), [callback\\_item\(\)](#), [callback\\_reason\(\)](#)

**31.138.2.69 open() [2/2]**

```
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 | <i>path</i>       | – the tree item's pathname (e.g. "Flintstones/Fred")                                                                                                                                                                                                                                                                                                |
| in | <i>docallback</i> | <p>– A flag that determines if the <a href="#">callback()</a> is invoked or not:</p> <ul style="list-style-type: none"> <li>• 0 - <a href="#">callback()</a> is not invoked</li> <li>• 1 - <a href="#">callback()</a> is invoked if item changed (default), <a href="#">callback_reason()</a> will be <code>FL_TREE_REASON_OPENED</code></li> </ul> |

**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\(\)](#)

### 31.138.2.70 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 | <i>item</i>       | – the item whose open state is to be toggled. Must not be NULL.                                                                                                                                                                                                                                                                                        |
| in | <i>docallback</i> | <p>– A flag that determines if the <a href="#">callback()</a> is invoked or not:</p> <ul style="list-style-type: none"> <li>• 0 - <a href="#">callback()</a> is not invoked</li> <li>• 1 - <a href="#">callback()</a> is invoked (default), <a href="#">callback_reason()</a> will be either FL_TREE_REASON_OPENED or FL_TREE_REASON_CLOSED</li> </ul> |

#### See also

[open\(\)](#), [close\(\)](#), [is\\_open\(\)](#), [is\\_close\(\)](#), [callback\\_item\(\)](#), [callback\\_reason\(\)](#)

### 31.138.2.71 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.

### 31.138.2.72 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 | <i>val</i> | – The new image, or zero to use the default [+] icon. |
|----|------------|-------------------------------------------------------|

### 31.138.2.73 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 | <i>item</i> | 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\(\)](#)

### 31.138.2.74 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.

### 31.138.2.75 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.

### 31.138.2.76 resize()

```
void Fl_Tree::resize (
 int X,
 int Y,
 int W,
 int H) [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](#).

### 31.138.2.77 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

### 31.138.2.78 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'.

**31.138.2.79 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\(int\)](#)

**31.138.2.80 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 | <code>size</code> | Sets the scrollbar size in pixels.<br>If 0 (default), scrollbar size tracks the global <a href="#">Fl::scrollbar_size()</a> |
|----|-------------------|-----------------------------------------------------------------------------------------------------------------------------|

**See also**

[Fl::scrollbar\\_size\(\)](#)

**31.138.2.81 select() [1/2]**

```
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 | <i>item</i>       | – the item to be selected. Must not be NULL.                                                                                                                                                                                                                                                                                                              |
| in | <i>docallback</i> | <p>– A flag that determines if the <a href="#">callback()</a> is invoked or not:</p> <ul style="list-style-type: none"> <li>• 0 - the <a href="#">callback()</a> is not invoked</li> <li>• 1 - the <a href="#">callback()</a> is invoked if item changed state, <a href="#">callback_reason()</a> will be <code>FL_TREE_REASON_SELECTED</code></li> </ul> |

#### Returns

- 1 - item's state was changed
- 0 - item was already selected, no change was made

### 31.138.2.82 [select\(\)](#) [2/2]

```
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 | <i>path</i>       | – the tree item's pathname (e.g. "Flintstones/Fred")                                                                                                                                                                                                                                                                                                                |
| in | <i>docallback</i> | <p>– A flag that determines if the <a href="#">callback()</a> is invoked or not:</p> <ul style="list-style-type: none"> <li>• 0 - the <a href="#">callback()</a> is not invoked</li> <li>• 1 - the <a href="#">callback()</a> is invoked if item changed state (default), <a href="#">callback_reason()</a> will be <code>FL_TREE_REASON_SELECTED</code></li> </ul> |

**Returns**

- 1 : OK: item's state was changed
- 0 : OK: item was already selected, no change was made
- -1 : ERROR: item was not found

**31.138.2.83 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 | <i>item</i>       | The item that will be selected (along with all its children). If NULL, <a href="#">first()</a> is used.                                                                                                                                                                                                                                                                                                                 |
| in | <i>docallback</i> | <ul style="list-style-type: none"> <li>– A flag that determines if the <a href="#">callback()</a> is invoked or not:           <ul style="list-style-type: none"> <li>• 0 - the <a href="#">callback()</a> is not invoked</li> <li>• 1 - the <a href="#">callback()</a> is invoked for each item that changed state (default), <a href="#">callback_reason()</a> will be FL_TREE_REASON_SELECTED</li> </ul> </li> </ul> |

**Returns**

Count of how many items were actually changed to the selected state.

**31.138.2.84 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 | <i>selitem</i>       | The item to be selected. If NULL, <a href="#">first()</a> is used.                                                                                                                                                                                                                                                                                                                                                                                                                          |
| in | <i>docalcallback</i> | <ul style="list-style-type: none"> <li>– A flag that determines if the <a href="#">callback()</a> is invoked or not:           <ul style="list-style-type: none"> <li>• 0 - the <a href="#">callback()</a> is not invoked</li> <li>• 1 - the <a href="#">callback()</a> is invoked for each item that changed state (default), <a href="#">callback_reason()</a> will be either <a href="#">FL_TREE_REASON_SELECTED</a> or <a href="#">FL_TREE_REASON_DESELECTED</a></li> </ul> </li> </ul> |

**Returns**

The number of items whose selection states were changed, if any.

**31.138.2.85 select\_toggle()**

```
void Fl_Tree::select_toggle (
 Fl_Tree_Item * item,
 int docalcallback = 1)
```

Toggle the select state of the specified 'item'.

Invokes the callback depending on the value of optional parameter 'docalcallback'. 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 | <i>item</i>          | – the item to be selected. Must not be NULL.                                                                                                                                                                                                                                                                                                                                                                                                               |
| in | <i>docalcallback</i> | <ul style="list-style-type: none"> <li>– A flag that determines if the <a href="#">callback()</a> is invoked or not:           <ul style="list-style-type: none"> <li>• 0 - the <a href="#">callback()</a> is not invoked</li> <li>• 1 - the <a href="#">callback()</a> is invoked (default), <a href="#">callback_reason()</a> will be either <a href="#">FL_TREE_REASON_SELECTED</a> or <a href="#">FL_TREE_REASON_DESELECTED</a></li> </ul> </li> </ul> |

**31.138.2.86 selectbox() [1/2]**

```
Fl_Botype Fl_Tree::selectbox () const
```

Sets the style of box used to draw selected items.

This is an fltk [Fl\\_Botype](#). The default is influenced by FLTK's current [Fl::scheme\(\)](#)

**31.138.2.87 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\(\)](#)

**31.138.2.88 selectmode() [1/2]**

```
Fl_Tree_Select Fl_Tree::selectmode () const
```

Gets the tree's current selection mode.

See [Fl\\_Tree\\_Select](#) for possible values.

**31.138.2.89 selectmode() [2/2]**

```
void Fl_Tree::selectmode (
 Fl_Tree_Select val)
```

Sets the tree's selection mode.

See [Fl\\_Tree\\_Select](#) for possible values.

**31.138.2.90 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 | <i>item</i> | The item that should take focus. If NULL, none will have focus. |
|----|-------------|-----------------------------------------------------------------|

**31.138.2.91 show\_item() [1/2]**

```
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 | <i>item</i> | The item to be shown. If NULL, <a href="#">first()</a> is used. |
| in | <i>yoff</i> | The pixel offset from the top for the displayed position.       |

#### See also

[show\\_item\\_top\(\)](#), [show\\_item\\_middle\(\)](#), [show\\_item\\_bottom\(\)](#)

### 31.138.2.92 show\_item() [2/2]

```
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 | <i>item</i> | The item to be shown. If NULL, <a href="#">first()</a> is used. |
|----|-------------|-----------------------------------------------------------------|

#### See also

[show\\_item\\_top\(\)](#), [show\\_item\\_middle\(\)](#), [show\\_item\\_bottom\(\)](#)

### 31.138.2.93 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 | <i>item</i> | The item to be shown. If NULL, <a href="#">first()</a> is used. |
|----|-------------|-----------------------------------------------------------------|

**31.138.2.94 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 | <i>item</i> | The item to be shown. If NULL, <a href="#">first()</a> is used. |
|----|-------------|-----------------------------------------------------------------|

**31.138.2.95 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 | <i>item</i> | The item to be shown. If NULL, <a href="#">first()</a> is used. |
|----|-------------|-----------------------------------------------------------------|

**31.138.2.96 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

**31.138.2.97 showcollapse() [1/2]**

```
int Fl_Tree::showcollapse () const
```

Returns 1 if the collapse icon is enabled, 0 if not.

**See also**

[showcollapse\(int\)](#)

### 31.138.2.98 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 | <i>val</i> | 1: shows collapse icons (default),<br>0: hides collapse icons. |
|----|------------|----------------------------------------------------------------|

### 31.138.2.99 showroot()

```
void Fl_Tree::showroot (
 int val)
```

Set if the root item should be shown or not.

#### Parameters

|    |            |                                                             |
|----|------------|-------------------------------------------------------------|
| in | <i>val</i> | 1 – show the root item (default)<br>0 – hide the root item. |
|----|------------|-------------------------------------------------------------|

### 31.138.2.100 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.

### 31.138.2.101 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.

### 31.138.2.102 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. |
|----|-----|------------------------------------------------------------|

**31.138.2.103 usericonmarginleft() [1/2]**

```
int Fl_Tree::usericonmarginleft () const
```

Get the amount of white space (in pixels) that should appear to the left of the usericon.

**31.138.2.104 usericonmarginleft() [2/2]**

```
void Fl_Tree::usericonmarginleft (
 int val)
```

Set the amount of white space (in pixels) that should appear to the left of the usericon.

**31.138.2.105 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\(int\)](#)

**31.138.2.106 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 | <i>pos</i> | The vertical position (in pixels) to scroll the tree to. |
|----|------------|----------------------------------------------------------|

**See also**[vposition\(\)](#), [hposition\(\)](#), [hposition\(int\)](#)**31.138.2.107 widgetmarginleft() [1/2]**

```
int Fl_Tree::widgetmarginleft () const
```

Get the amount of white space (in pixels) that should appear to the left of the child fltk widget (if any).

**31.138.2.108 widgetmarginleft() [2/2]**

```
void Fl_Tree::widgetmarginleft (
 int val)
```

Set the amount of white space (in pixels) that should appear to the left of the child fltk widget (if any).

The documentation for this class was generated from the following files:

- [Fl\\_Tree.H](#)
- [Fl\\_Tree.cxx](#)

**31.139 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'.*
- **FI\_Tree\_Item \* add** (const **FI\_Tree\_Prefs &prefs**, const char \*new\_label, **FI\_Tree\_Item \*newitem**)
 

*Add 'item' as immediate child with 'new\_label' and defaults from 'prefs'.*
- **FI\_Tree\_Item \* add** (const **FI\_Tree\_Prefs &prefs**, const char \*new\_label)
 

*Add a new child to this item with the name 'new\_label' and defaults from 'prefs'.*
- **FI\_Tree\_Item \* add** (const **FI\_Tree\_Prefs &prefs**, char \*\*arr, **FI\_Tree\_Item \*newitem**)
 

*Descend into path specified by 'arr' and add 'newitem' there.*
- **FI\_Tree\_Item \* add** (const **FI\_Tree\_Prefs &prefs**, char \*\*arr)
 

*Descend into the path specified by 'arr', and add a new child there.*
- **FI\_Tree\_Item \* child** (int index)
 

*Return the child item for the given 'index'.*
- const **FI\_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.*
- **FI\_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, **FI\_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\_onCollapse\_icon** (const **FI\_Tree\_Prefs &prefs**) const
 

*Was the event on the 'collapse' button of this item?*
- int **event\_on\_item** (const **FI\_Tree\_Prefs &prefs**) const
 

*Was event anywhere on the item?*
- int **event\_on\_label** (const **FI\_Tree\_Prefs &prefs**) const
 

*Was event on the `label()` of this item?*
- int **event\_onUser\_icon** (const **FI\_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** (**FI\_Tree\_Item \*item**)
 

*Find the index number for the specified 'item' in the current item's list of children.*
- const **FI\_Tree\_Item \* find\_child\_item** (const char \*name) const
 

*Return the /immediate/ child of current item that has the label 'name'.*
- **FI\_Tree\_Item \* find\_child\_item** (const char \*name)

- Non-const version of `FI_Tree_Item::find_child_item(const char *name) const.`
- const `FI_Tree_Item * find_child_item (char **arr) const`  
*Find child item by descending array 'arr' of names.*
- `FI_Tree_Item * find_child_item (char **arr)`  
*Non-const version of `FI_Tree_Item::find_child_item(char **arr) const.`*
- const `FI_Tree_Item * find_clicked (const FI_Tree_Prefs &prefs, int yonly=0) const`  
*Find the item that the last event was over.*
- `FI_Tree_Item * find_clicked (const FI_Tree_Prefs &prefs, int yonly=0)`  
*Non-const version of `FI_Tree_Item::find_clicked(const FI_Tree_Prefs&,int) const.`*
- const `FI_Tree_Item * find_item (char **arr) const`  
*Find item by descending array of 'names'.*
- `FI_Tree_Item * find_item (char **arr)`  
*Non-const version of `FI_Tree_Item::find_item(char **names) const.`*
- `FI_Tree_Item (const FI_Tree_Prefs &prefs)`  
*Constructor.*
- `FI_Tree_Item (FI_Tree *tree)`  
*Constructor.*
- `FI_Tree_Item (const FI_Tree_Item *o)`  
*Copy constructor.*
- int `h () const`  
*The item's height.*
- int `has_children () const`  
*See if this item has children.*
- `FI_Tree_Item * insert (const FI_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'.*
- `FI_Tree_Item * insert_above (const FI_Tree_Prefs &prefs, const char *new_label)`  
*Insert a new item named 'new\_label' above this item.*
- char `isActivated () const`  
*See if the item is activated.*
- char `isActive () const`  
*See if the item is activated. Alias for `isActivated()`.*
- int `isClose () const`  
*See if the item is 'closed'.*
- int `isOpen () const`  
*See if the item is 'open'.*
- int `isRoot () const`  
*Is this item the root of the tree?*
- char `isSelected () const`  
*See if the item is selected.*
- int `isVisible () const`  
*See if the item is visible.*
- void `label (const char *val)`  
*Set the label to 'name'.*
- const char \* `label () const`  
*Return the label.*
- int `label_h () const`  
*The item's label height.*
- int `label_w () const`  
*The item's maximum label width to right edge of `FI_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.*
- void `labelbgcolor (FI_Color val)`  
*Set item's label background color.*
- `FI_Color labelbgcolor () const`  
*Return item's label background text color.*
- void `labelcolor (FI_Color val)`  
*Set item's label text color. Alias for `labelfgcolor(FI_Color)`.*
- `FI_Color labelcolor () const`  
*Return item's label text color. Alias for `labelfgcolor() const`.*
- void `labelfgcolor (FI_Color val)`  
*Set item's label foreground text color.*
- `FI_Color labelfgcolor () const`  
*Return item's label foreground text color.*
- void `labelfont (FI_Font val)`  
*Set item's label font face.*
- `FI_Font labelfont () const`  
*Get item's label font face.*
- void `labelsize (FI_Fontsize val)`  
*Set item's label font size.*
- `FI_Fontsize labelsize () const`  
*Get item's label font size.*
- int `move (int to, int from)`  
*Move the item 'from' to sibling position of 'to'.*
- int `move (FI_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_above (FI_Tree_Item *item)`  
*Move the current item above the specified 'item'.*
- int `move_below (FI_Tree_Item *item)`  
*Move the current item below the specified 'item'.*
- int `move_into (FI_Tree_Item *item, int pos=0)`  
*Parent the current item as a child of the specified 'item'.*
- `FI_Tree_Item * next ()`  
*Return the next item in the tree.*
- `FI_Tree_Item * next_displayed (FI_Tree_Prefs &prefs)`  
*Same as `next_visible()`.*
- `FI_Tree_Item * next_sibling ()`  
*Return this item's next sibling.*
- `FI_Tree_Item * next_visible (FI_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.*
- `FI_Tree_Item * parent ()`  
*Return the parent for this item. Returns NULL if we are the root.*
- const `FI_Tree_Item * parent () const`  
*Return the const parent for this item. Returns NULL if we are the root.*
- void `parent (FI_Tree_Item *val)`  
*Set the parent for this item.*
- const `FI_Tree_Prefs & prefs () const`

- `FI_Tree_Item * prev ()`  
*Return the previous item in the tree.*
- `FI_Tree_Item * prev_displayed (FI_Tree_Prefs &prefs)`  
*Same as `prev_visible()`.*
- `FI_Tree_Item * prev_sibling ()`  
*Return this item's previous sibling.*
- `FI_Tree_Item * prev_visible (FI_Tree_Prefs &prefs)`  
*Return the previous `open()`, `visible()` item.*
- `int remove_child (FI_Tree_Item *item)`  
*Remove 'item' from the current item's children.*
- `int remove_child (const char *new_label)`  
*Remove immediate child (and its children) by its label 'name'.*
- `int reparent (FI_Tree_Item *newchild, int index)`  
*Reparent specified item as a child of ourself at position 'pos'.*
- `FI_Tree_Item * replace (FI_Tree_Item *new_item)`  
*Replace the current item with a new item.*
- `FI_Tree_Item * replace_child (FI_Tree_Item *olditem, FI_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.*
- `void swap_children (int ax, int bx)`  
*Swap two of our children, given two child index values 'ax' and 'bx'.*
- `int swap_children (FI_Tree_Item *a, FI_Tree_Item *b)`  
*Swap two of our immediate children, given item pointers.*
- `const FI_Tree * tree () const`  
*Return the tree for this item.*
- `FI_Tree * tree ()`  
*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 (void *data)`  
*Set a user-data value for the item.*
- `void * user_data () const`  
*Retrieve the user-data value that has been assigned to the item.*
- `void userdeicon (FI_Image *val)`  
*Set the usericon to draw when the item is deactivated.*
- `FI_Image * userdeicon () const`  
*Return the deactivated version of the user icon, if any.*
- `void usericon (FI_Image *val)`  
*Set the item's user icon to an `FI_Image`.*
- `FI_Image * usericon () const`  
*Get the item's user icon as an `FI_Image`. Returns '0' if disabled.*
- `int visible () const`

- `int visible_r () const`  
*See if the item is visible. Alias for `is_visible()`.*
- `int w () const`  
*See if item and all its parents are `open()` and `visible()`.*
- `int x () const`  
*The entire item's width to right edge of `FI_Tree`'s inner width within scrollbars.*
- `void widget (FI_Widget *val)`  
*Assign an FLTK widget to this item.*
- `FI_Widget * widget () const`  
*Return FLTK widget assigned to this item.*
- `int y () const`  
*The item's y position relative to the window.*
- `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 FI_Tree_Prefs &prefs, FI_Tree *tree)`
- `int calc_item_height (const FI_Tree_Prefs &prefs) const`  
*Return the item's 'visible' height.*
- `void draw_horizontal_connector (int x1, int x2, int y, const FI_Tree_Prefs &prefs)`  
*Internal: Horizontal connector line based on preference settings.*
- `void draw_vertical_connector (int x, int y1, int y2, const FI_Tree_Prefs &prefs)`  
*Internal: Vertical connector line based on preference settings.*
- `FI_Color drawbgcolor () const`  
*Returns the recommended background color used for drawing this item.*
- `FI_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.*

### 31.139.1 Detailed Description

Tree widget item.

This class is a single tree item, and manages all of the item's attributes. `FI_Tree_Item` is used by `FI_Tree`, which is comprised of many instances of `FI_Tree_Item`.

`FI_Tree_Item` is hierarchical; it dynamically manages an `FI_Tree_Item_Array` of children that are themselves instances of `FI_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:

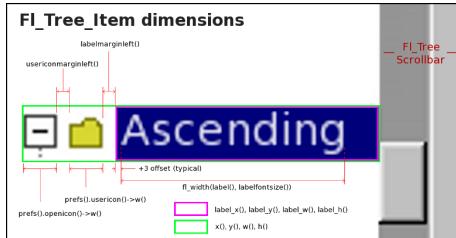


Figure 31.55 `Fl_Tree_Item`'s internal dimensions.

### 31.139.2 Constructor & Destructor Documentation

#### 31.139.2.1 `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.

#### 31.139.2.2 `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`'.

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

### 31.139.3 Member Function Documentation

#### 31.139.3.1 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.

#### 31.139.3.2 add() [1/4]

```
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

#### 31.139.3.3 add() [2/4]

```
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

31.139.3.4 `add()` [3/4]

```
F1_Tree_Item * F1_Tree_Item::add (
 const F1_Tree_Prefs & prefs,
 char ** arr,
 F1_Tree_Item * newitem)
```

Descend into path specified by 'arr' and add 'newitem' there.

Should be used only by `F1_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

31.139.3.5 `add()` [4/4]

```
F1_Tree_Item * F1_Tree_Item::add (
 const F1_Tree_Prefs & prefs,
 char ** arr)
```

Descend into the path specified by 'arr', and add a new child there.

Should be used only by `F1_Tree`'s internals. Adds the item based on the value of `prefs.sortorder()`.

**Returns**

the item added.

**Version**

1.3.0 release

31.139.3.6 `calc_item_height()`

```
int F1_Tree_Item::calc_item_height (
 const F1_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 not NULL)
- `usericon()` height (if not NULL) Does NOT include `F1_Tree::linespacing()`;

**Returns**

maximum pixel height

### 31.139.3.7 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'.

### 31.139.3.8 deactivate()

```
void Fl_Tree_Item::deactivate () [inline]
```

Deactivate the item; the callback() won't be invoked when clicked.

Same as activate(0)

### 31.139.3.9 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)

### 31.139.3.10 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.

### 31.139.3.11 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".

### 31.139.3.12 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

### 31.139.3.13 draw\_horizontal\_connector()

```
void Fl_Tree_Item::draw_horizontal_connector (
 int x1,
 int x2,
 int y,
 const Fl_Tree_Prefs & prefs) [protected]
```

Internal: Horizontal connector line based on preference settings.

#### 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                                     |

## 31.139.3.14 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() { }
 ~MyTreeItem() { }
 // 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:

```
// Example #1: using add()
MyTreeItem *bart = new MyTreeItem(..); // class derived from Fl_Tree_Item
tree->add("/Simpsons/Bart", bart); // Add item as /Simpsons/Bart
```

..or you can insert or replace existing items:

```
// Example #2: using replace()
MyTreeItem *marge = new MyTreeItem(..); // class derived from Fl_Tree_Item
item = tree->add("/Simpsons/Marge"); // create item
item->replace(mi); // replace it with our own
```

#### Parameters

|    |                     |                                                                                                                                                                          |
|----|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| in | <code>render</code> | Whether we should render content (1), or just tally the geometry (0). <code>Fl_Tree</code> 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

**31.139.3.15 draw\_vertical\_connector()**

```
void Fl_Tree_Item::draw_vertical_connector (
 int x,
 int y1,
 int y2,
 const Fl_Tree_Prefs & prefs) [protected]
```

Internal: Vertical connector line based on preference settings.

**Parameters**

|    |              |                                          |
|----|--------------|------------------------------------------|
| in | <i>x</i>     | The x position of the vertical connector |
| in | <i>y1</i>    | The top of the vertical connector        |
| in | <i>y2</i>    | The bottom of the vertical connector     |
| in | <i>prefs</i> | The <a href="#">Fl_Tree</a> prefs        |

**31.139.3.16 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

**31.139.3.17 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

**31.139.3.18 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

**31.139.3.19 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.

### 31.139.3.20 `find_child_item()` [1/2]

```
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

### 31.139.3.21 `find_child_item()` [2/2]

```
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

### 31.139.3.22 `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 | <code>prefs</code> | The parent tree's <a href="#">Fl_Tree_Prefs</a>                                              |
| in | <code>yonly</code> | – 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

**31.139.3.23 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

**31.139.3.24 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.

**31.139.3.25 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\(\)](#)

**31.139.3.26 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.

**31.139.3.27 label()**

```
void Fl_Tree_Item::label (
 const char * name)
```

Set the label to 'name'.

Makes and manages an internal copy of 'name'.

**31.139.3.28 label\_h()**

```
int Fl_Tree_Item::label_h () const [inline]
```

The item's label height.

**Version**

1.3.3

**31.139.3.29 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

**31.139.3.30 label\_x()**

```
int Fl_Tree_Item::label_x () const [inline]
```

The item's label x position relative to the window.

**Version**

1.3.3

**31.139.3.31 label\_y()**

```
int Fl_Tree_Item::label_y () const [inline]
```

The item's label y position relative to the window.

**Version**

1.3.3

**31.139.3.32 labelbgcolor() [1/2]**

```
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.

**31.139.3.33 labelbgcolor() [2/2]**

```
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.)

**31.139.3.34 move()** [1/2]

```
int Fl_Tree_Item::move (
 int to,
 int from)
```

Move the item 'from' to sibling position of 'to'.

**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\(Fl\\_Tree\\_Item\\*,int,int\)](#)

**31.139.3.35 move()** [2/2]

```
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')

**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\(int,int\)](#)

### 31.139.3.36 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\(Fl\\_Tree\\_Item\\*,int,int\)](#)

### 31.139.3.37 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\(Fl\\_Tree\\_Item\\*,int,int\)](#)

### 31.139.3.38 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\(Fl\\_Tree\\_Item\\*,int,int\)](#)

### 31.139.3.39 next()

```
F1_Tree_Item * F1_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 [F1\\_Tree::first\(\)](#).

#### Returns

the next item in the tree, or 0 if there's no more items.

### 31.139.3.40 next\_displayed()

```
F1_Tree_Item * F1_Tree_Item::next_displayed (
 F1_Tree_Prefs & prefs)
```

Same as [next\\_visible\(\)](#).

**Deprecated** in 1.3.3 for confusing name, use [next\\_visible\(\)](#) instead

### 31.139.3.41 next\_sibling()

```
F1_Tree_Item * F1_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.

### 31.139.3.42 next\_visible()

```
F1_Tree_Item * F1_Tree_Item::next_visible (
 F1_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

### 31.139.3.43 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.

### 31.139.3.44 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

### 31.139.3.45 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).

### 31.139.3.46 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\(\)](#)

### 31.139.3.47 prev\_sibling()

```
F1_Tree_Item * F1_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.

### 31.139.3.48 prev\_visible()

```
F1_Tree_Item * F1_Tree_Item::prev_visible (
 F1_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.

### 31.139.3.49 recalc\_tree()

```
void F1_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

**31.139.3.50 remove\_child() [1/2]**

```
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.

**31.139.3.51 remove\_child() [2/2]**

```
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 | <i>name</i> | The label name of the immediate child to remove |
|----|-------------|-------------------------------------------------|

**Returns**

0 if removed, -1 if not found.

**Version**

1.3.3

**31.139.3.52 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.

### 31.139.3.53 replace()

```
F1_Tree_Item * F1_Tree_Item::replace (
 F1_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 [F1\\_Tree\\_Item](#); see [draw<-\\_item\\_content\(\)](#).

#### Parameters

|    |                |                                          |
|----|----------------|------------------------------------------|
| in | <i>newitem</i> | The new item to replace the current item |
|----|----------------|------------------------------------------|

#### Returns

*newitem* on success, NULL if could not be replaced.

#### See also

[F1\\_Tree\\_Item::draw\\_item\\_content\(\)](#), [F1\\_Tree::root\(F1\\_Tree\\_Item\\*\)](#)

#### Version

1.3.3 ABI feature

### 31.139.3.54 replace\_child()

```
F1_Tree_Item * F1_Tree_Item::replace_child (
 F1_Tree_Item * olditem,
 F1_Tree_Item * newitem)
```

Replace existing child '*olditem*' with '*newitem*'.

The '*olditem*' is destroyed if successful. Can be used to put custom items (derived from [F1\\_Tree\\_Item](#)) into the tree. No checks are made to see if an item with the same name exists.

#### Parameters

|    |                |                                                      |
|----|----------------|------------------------------------------------------|
| in | <i>olditem</i> | The item to be found and replaced                    |
| in | <i>newitem</i> | The new item to take the place of ' <i>olditem</i> ' |

#### 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

**31.139.3.55 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.

**31.139.3.56 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".

**31.139.3.57 show\_self()**

```
void Fl_Tree_Item::show_self (
 const char * indent = "") const
```

Print the tree as 'ascii art' to stdout.

Used mainly for debugging.

**31.139.3.58 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\(\)](#)

**31.139.3.59 swap\_children() [1/2]**

```
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 | <i>ax,bx</i> | the index of the items to swap |
|----|--------------|--------------------------------|

**31.139.3.60 swap\_children()** [2/2]

```
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 | <i>a,b</i> | 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.

**31.139.3.61 tree()** [1/2]

```
const Fl_Tree* Fl_Tree_Item::tree () const [inline]
```

Return the tree for this item.

**Version**

1.3.3

**31.139.3.62 tree()** [2/2]

```
Fl_Tree* Fl_Tree_Item::tree () [inline]
```

Return the tree for this item.

**Version**

1.3.4

## 31.139.3.63 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 | <code>index</code> | Our index# in the parent.<br>Special case if <code>index=-1</code> : become an orphan; null out all parent/sibling associations. |
|----|--------------------|----------------------------------------------------------------------------------------------------------------------------------|

## 31.139.3.64 userdeicon() [1/2]

```
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

## 31.139.3.65 userdeicon() [2/2]

```
Fl_Image* Fl_Tree_Item::userdeicon () const [inline]
```

Return the deactivated version of the user icon, if any.

Returns 0 if none.

**31.139.3.66 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\(Fl\\_Image\\*\)](#)

**31.139.3.67 visible\_r()**

```
int Fl_Tree_Item::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 invisible or [close\(\)](#)ed.

**31.139.3.68 w()**

```
int Fl_Tree_Item::w () const [inline]
```

The entire item's width to right edge of [Fl\\_Tree](#)'s inner width within scrollbars.

The documentation for this class was generated from the following files:

- [Fl\\_Tree\\_Item.H](#)
- [Fl\\_Tree\\_Item.cxx](#)

## 31.140 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 \(FI\\_Tree\\_Item \\*val\)](#)  
*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.*
- [FI\\_Tree\\_Item\\_Array \(int new\\_chunksize=10\)](#)  
*Constructor; creates an empty array.*
- [FI\\_Tree\\_Item\\_Array \(const FI\\_Tree\\_Item\\_Array \\*o\)](#)  
*Copy constructor. Makes new copy of array, with new instances of each item.*
- void [insert \(int pos, FI\\_Tree\\_Item \\*new\\_item\)](#)  
*Insert an item at index position pos.*
- void [manage\\_item\\_destroy \(int val\)](#)  
*Option to control if FI\_Tree\_Item\_Array's destructor will also destroy the FI\_Tree\_Item's.*
- int [manage\\_item\\_destroy \(\) const](#)
- int [move \(int to, int from\)](#)  
*Move item at 'from' to new position 'to' in the array.*
- [FI\\_Tree\\_Item \\* operator\[\] \(int i\)](#)  
*Return the item and index i.*
- const [FI\\_Tree\\_Item \\* operator\[\] \(int i\) const](#)  
*Const version of operator[](int i)*
- void [remove \(int index\)](#)  
*Remove the item at.*
- int [remove \(FI\\_Tree\\_Item \\*item\)](#)  
*Remove the item from the array.*
- int [reparent \(FI\\_Tree\\_Item \\*item, FI\\_Tree\\_Item \\*newparent, int pos\)](#)  
*Reparent specified item as a child of ourself.*
- void [replace \(int pos, FI\\_Tree\\_Item \\*new\\_item\)](#)  
*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.*
- [~FI\\_Tree\\_Item\\_Array \(\)](#)  
*Destructor. Calls each item's destructor, destroys internal\_items array.*

### 31.140.1 Detailed Description

Manages an array of [FI\\_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  $0 < \text{index} < \text{total}()$  (unless otherwise noted).

### 31.140.2 Constructor & Destructor Documentation

### 31.140.2.1 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 chunk-size is 10.

## 31.140.3 Member Function Documentation

### 31.140.3.1 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.

### 31.140.3.2 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.

### 31.140.3.3 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)

## 31.140.3.4 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 `pos >= total()`, the item is appended to the array. If `pos < 0`, the item is prepended (works like `pos == 0`).

## 31.140.3.5 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.

## 31.140.3.6 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)

## 31.140.3.7 remove() [1/2]

```
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.

### 31.140.3.8 `remove()` [2/2]

```
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.

### 31.140.3.9 `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)

### 31.140.3.10 `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](#)

## 31.141 Fl\_Tree\_Prefs Class Reference

Tree widget's preferences.

```
#include <Fl_Tree_Prefs.H>
```

## Public Member Functions

- `FI_Image * closedeicon () const`  
*Return the deactivated version of the close icon, if any.*
- `FI_Image * closeicon () const`  
*Gets the default 'close' icon Returns the FI\_Image\* of the icon, or 0 if none.*
- `void closeicon (FI_Image *val)`  
*Sets the icon to be used as the 'close' icon.*
- `FI_Color connectorcolor () const`  
*Get the connector color used for tree connection lines.*
- `void connectorcolor (FI_Color val)`  
*Set the connector color used for tree connection lines.*
- `FI_Tree_Connector connectorstyle () const`  
*Get the connector style.*
- `void connectorstyle (FI_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 (FI_Tree_Item *o) const`
- `FI_Tree_Prefs ()`  
*FI\_Tree\_Prefs constructor.*
- `void item_draw_callback (FI_Tree_Item_Draw_Callback *cb, void *data=0)`
- `FI_Tree_Item_Draw_Callback * item_draw_callback () const`
- `FI_Tree_Item_Draw_Mode item_draw_mode () const`  
*Get the 'item draw mode' used for the tree.*
- `void item_draw_mode (FI_Tree_Item_Draw_Mode val)`  
*Set the 'item draw mode' used for the tree to val.*
- `void * item_draw_user_data () const`
- `FI_Color item_labelbgcolor () const`  
*Get the default label background color.*
- `void item_labelbgcolor (FI_Color val)`  
*Set the default label background color.*
- `FI_Color item_labelfgcolor () const`  
*Get the default label foreground color.*
- `void item_labelfgcolor (FI_Color val)`  
*Set the default label foreground color.*
- `FI_Font item_labelfont () const`  
*Return the label's font.*
- `void item_labelfont (FI_Font val)`  
*Set the label's font to val.*
- `FI_Fontsize item_labelsize () const`  
*Return the label's size in pixels.*
- `void item_labelsize (FI_Fontsize val)`  
*Set the label's size in pixels to val.*
- `FI_Tree_Item_Reselect_Mode item_reselect_mode () const`  
*Returns the current item re-selection mode.*
- `void item_reselect_mode (FI_Tree_Item_Reselect_Mode mode)`

- **FI\_Color labelbgcolor () const**  
*Obsolete: Get the default label background color. Please use [item\\_labelbgcolor\(\)](#) instead.*
- **void labelbgcolor (FI\_Color val)**  
*Obsolete: Set the default label background color. Please use [item\\_labelbgcolor\(FI\\_Color\)](#) instead.*
- **FI\_Color labelfgcolor () const**  
*Obsolete: Get the default label foreground color. Please use [item\\_labelfgcolor\(\)](#) instead.*
- **void labelfgcolor (FI\_Color val)**  
*Obsolete: Set the default label foreground color. Please use [item\\_labelfgcolor\(FI\\_Color\)](#) instead.*
- **FI\_Font labelfont () const**  
*Obsolete: Return the label's font. Please use [item\\_labelfont\(\)](#) instead.*
- **void labelfont (FI\_Font val)**  
*Obsolete: Set the label's font to val. Please use [item\\_labelfont\(FI\\_Font\)](#) 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.*
- **FI\_Fontsize labelsize () const**  
*Obsolete: Return the label's size in pixels. Please use [item\\_labelsize\(\)](#) instead.*
- **void labelsize (FI\_Fontsize val)**  
*Obsolete: Set the label's size in pixels to val. Please use [item\\_labelsize\(FI\\_Fontsize\)](#) 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.*
- **FI\_Image \* opendeicon () const**  
*Return the deactivated version of the open icon, if any.*
- **FI\_Image \* openicon () const**  
*Get the current default 'open' icon.*
- **void openicon (FI\_Image \*val)**  
*Sets the default icon to be used as the 'open' icon when items are add()ed to the tree.*
- **FI\_Botype selectbox () const**  
*Get the default selection box's box drawing style as an FI\_Botype.*
- **void selectbox (FI\_Botype 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.*

### 31.141.1 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.

### 31.141.2 Member Function Documentation

### 31.141.2.1 closedeicon()

```
Fl_Image* Fl_Tree_Prefs::closedeicon () const [inline]
```

Return the deactivated version of the close icon, if any.

Returns 0 if none.

### 31.141.2.2 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. |
|----|-----|-------------------------------------------------------|

### 31.141.2.3 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.

### 31.141.2.4 item\_labelbgcolor() [1/2]

```
Fl_Color Fl_Tree_Prefs::item_labelbgcolor (
 void) const [inline]
```

Get the default label background color.

This returns the `Fl_Tree::color()` unless `item_labelbgcolor()` has been set explicitly.

### 31.141.2.5 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()`.

**31.141.2.6 marginbottom()** [1/2]

```
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.

**31.141.2.7 marginbottom()** [2/2]

```
void Fl_Tree_Prefs::marginbottom (
 int val) [inline]
```

Set the bottom margin's value in pixels This is the extra distance the vertical scroller lets you travel.

**31.141.2.8 opendeicon()**

```
Fl_Image* Fl_Tree_Prefs::opendeicon () const [inline]
```

Return the deactivated version of the open icon, if any.

Returns 0 if none.

**31.141.2.9 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.

**31.141.2.10 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. |
|----|-----|-------------------------------------------------------|

**31.141.2.11 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.

**31.141.2.12 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**

|                 |                  |                                                                |
|-----------------|------------------|----------------------------------------------------------------|
| <code>in</code> | <code>val</code> | 1: shows collapse icons (default),<br>0: hides collapse icons. |
|-----------------|------------------|----------------------------------------------------------------|

**31.141.2.13 showroot()**

```
void Fl_Tree_Prefs::showroot (
 int val) [inline]
```

Set if the root item should be shown or not.

**Parameters**

|                 |                  |                                                             |
|-----------------|------------------|-------------------------------------------------------------|
| <code>in</code> | <code>val</code> | 1 – show the root item (default)<br>0 – hide the root item. |
|-----------------|------------------|-------------------------------------------------------------|

**31.141.2.14 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.

## 31.141.2.15 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:

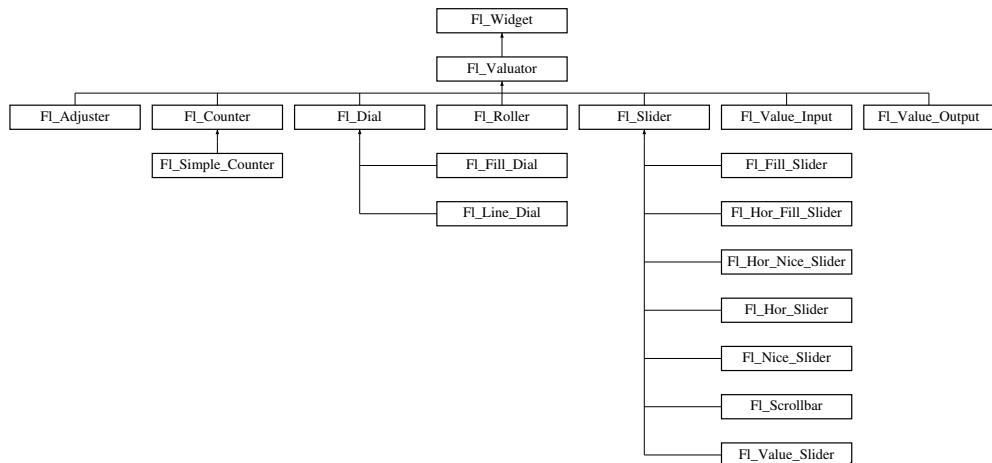
- [Fl\\_Tree\\_Prefs.H](#)
- [Fl\\_Tree\\_Prefs.cxx](#)

## 31.142 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.*
- 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.
- void **step** (int a)  
See double *FI\_Valuator::step() const*.
- void **step** (double a, int b)  
See double *FI\_Valuator::step() const*.
- void **step** (double s)  
See double *FI\_Valuator::step() const*.
- double **step** () const  
Gets or sets the step value.
- double **value** () const  
Gets the floating point(double) value.
- int **value** (double)  
Sets the current value.

## Protected Member Functions

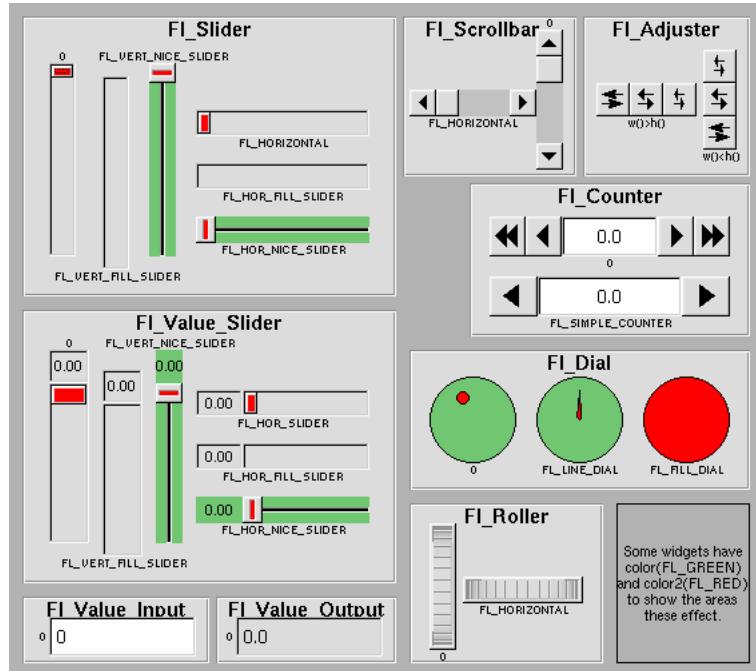
- **FI\_Valuator** (int X, int Y, int W, int H, const char \*L)  
Creates a new *FI\_Valuator* widget using the given position, size, and label string.
- void **handle\_drag** (double newvalue)  
Called during a drag operation, after an *FL\_WHEN\_CHANGED* event is received and before the callback.
- 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.

## Additional Inherited Members

### 31.142.1 Detailed Description

The [FI\\_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 31.56 Valuators derived from FI\_Valuator**

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.

### 31.142.2 Constructor & Destructor Documentation

#### 31.142.2.1 FI\_Valuator()

```
Fl_Valuator::Fl_Valuator (
 int X,
 int Y,
 int W,
 int H,
 const char * L) [protected]
```

Creates a new [FI\\_Valuator](#) widget using the given position, size, and label string.

The default boxtyle is `FL_NO_BOX`.

### 31.142.3 Member Function Documentation

#### 31.142.3.1 bounds()

```
void Fl_Valuator::bounds (
 double a,
 double b) [inline]
```

Sets the minimum (a) and maximum (b) values for the valuator widget.

#### 31.142.3.2 clamp()

```
double Fl_Valuator::clamp (
 double v)
```

Clamps the passed value to the valuator range.

#### 31.142.3.3 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 %.*n*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.

#### 31.142.3.4 handle\_drag()

```
void Fl_Valuator::handle_drag (
 double v) [protected]
```

Called during a drag operation, after an FL\_WHEN\_CHANGED event is received and before the callback.

**31.142.3.5 handle\_release()**

```
void Fl_Valuator::handle_release () [protected]
```

Called after an FL\_WHEN\_RELEASE event is received and before the callback.

**31.142.3.6 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.

**31.142.3.7 maximum() [1/2]**

```
double Fl_Valuator::maximum () const [inline]
```

Gets the maximum value for the valuator.

**31.142.3.8 maximum() [2/2]**

```
void Fl_Valuator::maximum (
 double a) [inline]
```

Sets the maximum value for the valuator.

**31.142.3.9 minimum() [1/2]**

```
double Fl_Valuator::minimum () const [inline]
```

Gets the minimum value for the valuator.

**31.142.3.10 minimum() [2/2]**

```
void Fl_Valuator::minimum (
 double a) [inline]
```

Sets the minimum value for the valuator.

### 31.142.3.11 precision()

```
void Fl_Valuator::precision (
 int digits)
```

Sets the step value to  $1.0 / 10^{\text{digits}}$ .

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.  $1.0/1 = 1$ .

### 31.142.3.12 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.

### 31.142.3.13 round()

```
double Fl_Valuator::round (
 double v)
```

Round the passed value to the nearest step increment.

Does nothing if step is zero.

### 31.142.3.14 set\_value()

```
void Fl_Valuator::set_value (
 double v) [inline], [protected]
```

Sets the current floating point value.

## 31.142.3.15 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.

## 31.142.3.16 value() [1/2]

```
double Fl_Valuator::value () const [inline]
```

Gets the floating point(double) value.

See int [value\(double\)](#)

## 31.142.3.17 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).

The documentation for this class was generated from the following files:

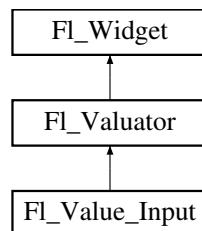
- [Fl\\_Valuator.H](#)
- [Fl\\_Valuator.cxx](#)

## 31.143 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

- `FI_Color cursor_color () const`  
*Gets the color of the text cursor.*
- `void cursor_color (FI_Color n)`  
*Sets the color of the text cursor.*
- `FI_Value_Input (int x, int y, int w, int h, const char *l=0)`  
*Creates a new `FI_Value_Input` widget using the given position, size, and label string.*
- `int handle (int)`  
*Handles the specified event.*
- `void resize (int, int, int, int)`  
*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.*
- `FI_Color textcolor () const`  
*Gets the color of the text in the value box.*
- `void textcolor (FI_Color n)`  
*Sets the color of the text in the value box.*
- `FI_Font textfont () const`  
*Gets the typeface of the text in the value box.*
- `void textfont (FI_Font s)`  
*Sets the typeface of the text in the value box.*
- `FI_Fontsize textsize () const`  
*Gets the size of the text in the value box.*
- `void textsize (FI_Fontsize s)`  
*Sets the size of the text in the value box.*

## Public Attributes

- `FI_Input input`

## Protected Member Functions

- `void draw ()`  
*Draws the widget.*

## Additional Inherited Members

### 31.143.1 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 31.57 Fl\_Value\_Input

## See also

[Fl\\_Widget::shortcut\\_label\(int\)](#)

### 31.143.2 Constructor & Destructor Documentation

#### 31.143.2.1 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 boxtyle is FL\_DOWN\_BOX.

### 31.143.3 Member Function Documentation

### 31.143.3.1 cursor\_color() [1/2]

```
F1_Color F1_Value_Input::cursor_color () const [inline]
```

Gets the color of the text cursor.

The text cursor is black by default.

### 31.143.3.2 cursor\_color() [2/2]

```
void F1_Value_Input::cursor_color (
 F1_Color n) [inline]
```

Sets the color of the text cursor.

The text cursor is black by default.

### 31.143.3.3 draw()

```
void F1_Value_Input::draw () [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:

```
F1_Widget *s = &scrollbar; // scrollbar is an embedded F1_Scrollbar
s->draw(); // calls F1_Scrollbar::draw()
```

Implements [F1\\_Widget](#).

### 31.143.3.4 handle()

```
int F1_Value_Input::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.

**Parameters**

|    |              |                            |
|----|--------------|----------------------------|
| in | <i>event</i> | 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](#).

**31.143.3.5 resize()**

```
void Fl_Value_Input::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 | <i>x,y</i> | new position relative to the parent window |
| in | <i>w,h</i> | new size                                   |

**See also**

[position\(int,int\)](#), [size\(int,int\)](#)

Reimplemented from [Fl\\_Widget](#).

### 31.143.3.6 `shortcut()` [1/2]

```
int Fl_Value_Input::shortcut () const [inline]
```

Returns the current shortcut key for the Input.

#### See also

[Fl\\_Value\\_Input::shortcut\(int\)](#)

### 31.143.3.7 `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).

### 31.143.3.8 `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.

### 31.143.3.9 `textcolor()` [1/2]

```
Fl_Color Fl_Value_Input::textcolor () const [inline]
```

Gets the color of the text in the value box.

### 31.143.3.10 `textcolor()` [2/2]

```
void Fl_Value_Input::textcolor (
 Fl_Color n) [inline]
```

Sets the color of the text in the value box.

## 31.143.3.11 textfont() [1/2]

```
Fl_Font Fl_Value_Input::textfont () const [inline]
```

Gets the typeface of the text in the value box.

## 31.143.3.12 textfont() [2/2]

```
void Fl_Value_Input::textfont (
 Fl_Font s) [inline]
```

Sets the typeface of the text in the value box.

## 31.143.3.13 textsize() [1/2]

```
Fl_Fontsize Fl_Value_Input::textsize () const [inline]
```

Gets the size of the text in the value box.

## 31.143.3.14 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:

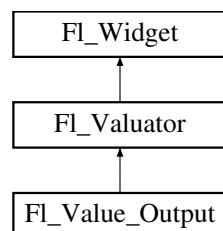
- Fl\_Value\_Input.H
- Fl\_Value\_Input.cxx

## 31.144 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)  
*Handles the specified event.*
- void **soft** (uchar *s*)  
*If "soft" is turned on, the user is allowed to drag the value outside the range.*
- uchar **soft** () const  
*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*)

## Protected Member Functions

- void **draw** ()  
*Draws the widget.*

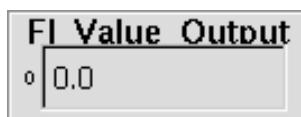
## Additional Inherited Members

### 31.144.1 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  $10 * \text{step}()$ , and the right button by  $100 * \text{step}()$ .

This is much lighter-weight than **Fl\_Value\_Input** because it contains no text editing code or character buffer.



**Figure 31.58 Fl\_Value\_Output**

### 31.144.2 Constructor & Destructor Documentation

### 31.144.2.1 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 boxtyle is `FL_NO_BOX`.

Inherited destructor destroys the Valuator.

## 31.144.3 Member Function Documentation

### 31.144.3.1 draw()

```
void Fl_Value_Output::draw () [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](#).

### 31.144.3.2 handle()

```
int Fl_Value_Output::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.

**Parameters**

|                 |                    |                            |
|-----------------|--------------------|----------------------------|
| <code>in</code> | <code>event</code> | the kind of event received |
|-----------------|--------------------|----------------------------|

**Return values**

|                |                                          |
|----------------|------------------------------------------|
| <code>0</code> | if the event was not used or understood  |
| <code>1</code> | if the event was used and can be deleted |

**See also**

[Fl\\_Event](#)

Reimplemented from [Fl\\_Widget](#).

**31.144.3.3 soft() [1/2]**

```
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.

**31.144.3.4 soft() [2/2]**

```
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.

**31.144.3.5 textcolor() [1/2]**

```
Fl_Color Fl_Value_Output::textcolor () const [inline]
```

Sets the color of the text in the value box.

**31.144.3.6 textcolor() [2/2]**

```
void Fl_Value_Output::textcolor (
 Fl_Color s) [inline]
```

Gets the color of the text in the value box.

## 31.144.3.7 textfont() [1/2]

```
Fl_Font Fl_Value_Output::textfont () const [inline]
```

Gets the typeface of the text in the value box.

## 31.144.3.8 textfont() [2/2]

```
void Fl_Value_Output::textfont (
 Fl_Font s) [inline]
```

Sets the typeface of the text in the value box.

## 31.144.3.9 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:

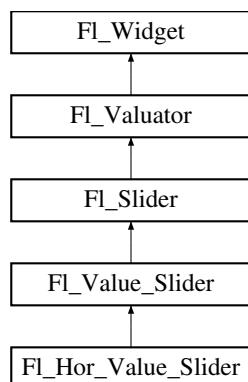
- Fl\_Value\_Output.H
- Fl\_Value\_Output.cxx

## 31.145 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

- **FI\_Value\_Slider** (int *x*, int *y*, int *w*, int *h*, const char \**l*=0)  
*Creates a new FI\_Value\_Slider widget using the given position, size, and label string.*
- int **handle** (int)  
*Handles the specified event.*
- **FI\_Color** **textcolor** () const  
*Gets the color of the text in the value box.*
- void **textcolor** (FI\_Color *s*)  
*Sets the color of the text in the value box.*
- **FI\_Font** **textfont** () const  
*Gets the typeface of the text in the value box.*
- void **textfont** (FI\_Font *s*)  
*Sets the typeface of the text in the value box.*
- **FI\_Fontsize** **textsize** () const  
*Gets the size of the text in the value box.*
- void **textsize** (FI\_Fontsize *s*)  
*Sets the size of the text in the value box.*

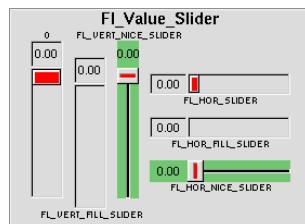
## Protected Member Functions

- void **draw** ()  
*Draws the widget.*

## Additional Inherited Members

### 31.145.1 Detailed Description

The **FI\_Value\_Slider** widget is a **FI\_Slider** widget with a box displaying the current value.



**Figure 31.59 FI\_Value\_Slider**

### 31.145.2 Constructor & Destructor Documentation

## 31.145.2.1 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 boxtyle is FL\_DOWN\_BOX.

## 31.145.3 Member Function Documentation

## 31.145.3.1 draw()

```
void Fl_Value_Slider::draw () [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](#).

## 31.145.3.2 handle()

```
int Fl_Value_Slider::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.

**Parameters**

|    |              |                            |
|----|--------------|----------------------------|
| in | <i>event</i> | 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](#).

**31.145.3.3 textcolor() [1/2]**

```
Fl_Color Fl_Value_Slider::textcolor () const [inline]
```

Gets the color of the text in the value box.

**31.145.3.4 textcolor() [2/2]**

```
void Fl_Value_Slider::textcolor (
 Fl_Color s) [inline]
```

Sets the color of the text in the value box.

**31.145.3.5 textfont() [1/2]**

```
Fl_Font Fl_Value_Slider::textfont () const [inline]
```

Gets the typeface of the text in the value box.

**31.145.3.6 textfont() [2/2]**

```
void Fl_Value_Slider::textfont (
 Fl_Font s) [inline]
```

Sets the typeface of the text in the value box.

**31.145.3.7 textsize() [1/2]**

```
Fl_Fontsize Fl_Value_Slider::textsize () const [inline]
```

Gets the size of the text in the value box.

**31.145.3.8 textsize() [2/2]**

```
void Fl_Value_Slider::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:

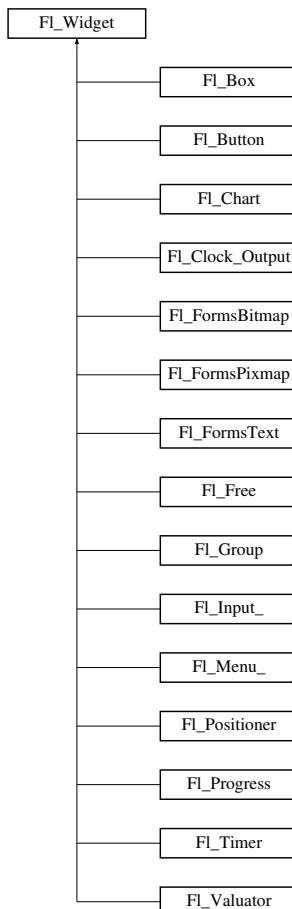
- [Fl\\_Value\\_Slider.H](#)
- [Fl\\_Value\\_Slider.cxx](#)

## 31.146 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 **Fl\_Group \* as\_group ()**  
*Returns an **Fl\_Group** pointer if this widget is an **Fl\_Group**.*
- virtual **Fl\_Window \* as\_window ()**  
*Returns an **Fl\_Window** pointer if this widget is an **Fl\_Window**.*
- **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, void \*p)**  
*Sets the current callback function for the widget.*
- void **callback (Fl\_Callback \*cb)**  
*Sets the current callback function 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 ()**

- **Fl\_Color color () const**  
*Disables keyboard focus navigation with this widget.*
- **void color (Fl\_Color bg)**  
*Gets the background color of the widget.*
- **void color (Fl\_Color bg, Fl\_Color sel)**  
*Sets the background and selection color of the widget.*
- **Fl\_Color color2 () const**  
*For back compatibility only.*
- **void color2 (unsigned a)**  
*For back compatibility only.*
- **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.*
- **void do\_callback ()**  
*Calls the widget callback function with default arguments.*
- **void do\_callback (Fl\_Widget \*widget, long arg)**  
*Calls the widget callback function with arbitrary arguments.*
- **void do\_callback (Fl\_Widget \*widget, void \*arg=0)**  
*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.*

- **FI\_Image \* image ()**  
Gets the image that is used as part of the widget label when in the active state.
- const **FI\_Image \* image () const**  
Gets the image that is used as part of the widget label when in the active state.
- void **image (FI\_Image \*img)**  
Sets the image to use as part of the widget label when in the active state.
- void **image (FI\_Image &img)**  
Sets the image to use as part of the widget label when in the active state.
- int **inside (const FI\_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 (FI\_Labeltype a, const char \*b)**  
Shortcut to set the label text and type in one call.
- **FI\_Color labelcolor () const**  
Gets the label color.
- void **labelcolor (FI\_Color c)**  
Sets the label color.
- **FI\_Font labelfont () const**  
Gets the font to use.
- void **labelfont (FI\_Font f)**  
Sets the font to use.
- **FI\_Fontsize labelsize () const**  
Gets the font size in pixels.
- void **labelszie (FI\_Fontsize pix)**  
Sets the font size in pixels.
- **FI\_Labeltype labeltype () const**  
Gets the label type.
- void **labeltype (FI\_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.
- unsigned int **output () const**  
Returns if a widget is used for output only.
- **FI\_Group \* parent () const**  
Returns a pointer to the parent widget.
- void **parent (FI\_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.
- **FI\_Color selection\_color () const**

- `void selection_color (Fl_Color a)`  
*Gets the selection color.*
- `void set_active ()`  
*Sets the selection color.*
- `void set_changed ()`  
*Marks the widget as active without sending events or changing focus.*
- `void set_output ()`  
*Marks the value of the widget as changed.*
- `void set_visible ()`  
*Makes the widget visible.*
- `void set_visible_focus ()`  
*Enables keyboard focus navigation with this widget.*
- `void shortcut_label (int value)`  
*Sets whether the widget's label uses '&' to indicate shortcuts.*
- `int shortcut_label () const`  
*Returns 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 current tooltip text.*
- `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 (void *v)`  
*Sets the user data for this widget.*
- `unsigned int visible () const`  
*Returns whether a widget is visible.*
- `void visible_focus (int v)`  
*Modifies keyboard focus navigation.*
- `unsigned int visible_focus ()`  
*Checks whether this widget has a visible focus.*

- int `visible_r () const`  
*Returns whether a widget and all its parents are visible.*
- int `w () const`  
*Gets the widget width.*
- `FI_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.*
- `FI_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 `~FI_Widget ()`  
*Destroys the widget.*

## Static Public Member Functions

- static void `default_callback (FI_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_ACENTS_MENU` = 1<<19,  
`NEEDS_KEYBOARD` = 1<<20, `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 its box style.*
- void `draw_box (FI_Boxtype t, FI_Color c) const`  
*Draws a box of type t, of color c at the widget's position and size.*
- void `draw_box (FI_Boxtype t, int x, int y, int w, int h, FI_Color c) const`

- Draws a box of type *t*, of color *c* at the position *X,Y* and size *W,H*.
- void **draw\_focus ()**  
*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 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

- void **Fl::focus (Fl\_Widget \*)**
- class **Fl\_Group**

### 31.146.1 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.

### 31.146.2 Member Enumeration Documentation

#### 31.146.2.1 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 ( <a href="#">Fl_Widget</a> )                                                                                                                    |
| FORCE_POSITION       | don't let the window manager position the window ( <a href="#">Fl_Widget</a> )                                                                                           |
| NON_MODAL            | this is a hovering toolbar window ( <a href="#">Fl_Widget</a> )                                                                                                          |
| SHORTCUT_LABEL       | the label contains a shortcut we need to draw                                                                                                                            |
| CHANGED              | the widget value changed                                                                                                                                                 |
| OVERRIDE             | position window on top ( <a href="#">Fl_Widget</a> )                                                                                                                     |
| 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 ( <a href="#">Fl_Group</a> )                                                                                              |
| MENU_WINDOW          | a temporary popup window, dismissed by clicking outside ( <a href="#">Fl_Widget</a> )                                                                                    |
| TOOLTIP_WINDOW       | a temporary popup, transparent to events, and dismissed easily ( <a href="#">Fl_Widget</a> )                                                                             |
| MODAL                | a window blocking input to all other windows ( <a href="#">Fl_Widget</a> )                                                                                               |
| NO_OVERLAY           | window not using a hardware overlay plane ( <a href="#">Fl_Menu_Window</a> )                                                                                             |
| 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 ( <a href="#">Fl_Widget</a> )                                                                                                                        |
| MAC_USE_ACCENTS_MENU | On the Mac OS platform, pressing and holding a key on the keyboard opens an accented-character menu window ( <a href="#">Fl_Input</a> , <a href="#">Fl_Text_Editor</a> ) |
| NEEDS_KEYBOARD       | set this on touch screen devices if a widget needs a keyboard when it gets Focus.<br><br>See also<br><br><a href="#">Fl_Screen_Driver::request_keyboard()</a>            |
| USERFLAG3            | reserved for 3rd party extensions                                                                                                                                        |
| USERFLAG2            | reserved for 3rd party extensions                                                                                                                                        |
| USERFLAG1            | reserved for 3rd party extensions                                                                                                                                        |

## 31.146.3 Constructor &amp; Destructor Documentation

31.146.3.1 [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 | <i>x,y</i>   | the position of the widget relative to the enclosing window |
| in | <i>w,h</i>   | size of the widget in pixels                                |
| in | <i>label</i> | optional text for the widget label                          |

**31.146.3.2 ~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.

**31.146.4 Member Function Documentation****31.146.4.1 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\(\)](#)

**31.146.4.2 active()**

```
unsigned int Fl_Widget::active () const [inline]
```

Returns whether the widget is active.

**Return values**

|          |                           |
|----------|---------------------------|
| <i>0</i> | if the widget is inactive |
|----------|---------------------------|

**See also**

[active\\_r\(\)](#), [activate\(\)](#), [deactivate\(\)](#)

**31.146.4.3 active\_r()**

```
int Fl_Widget::active_r () const
```

Returns whether the widget and all of its parents are active.

**Return values**

|          |                                                   |
|----------|---------------------------------------------------|
| <i>0</i> | if this or any of the parent widgets are inactive |
|----------|---------------------------------------------------|

**See also**

[active\(\)](#), [activate\(\)](#), [deactivate\(\)](#)

**31.146.4.4 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](#)

**31.146.4.5 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 | <i>alignment</i> | new label alignment |
|----|------------------|---------------------|

**See also**[align\(\)](#), [Fl\\_Align](#)**31.146.4.6 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\(\)](#)

**Todo** [Internal] The `user_data` value must be implemented using `fl_intptr_t` or similar to avoid 64-bit platform incompatibilities.

**31.146.4.7 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\(\)](#)**31.146.4.8 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**

|                   |                                                                   |
|-------------------|-------------------------------------------------------------------|
| <code>NULL</code> | if this widget is not derived from <a href="#">Fl_Gl_Window</a> . |
|-------------------|-------------------------------------------------------------------|

**Note**

This method is provided to avoid dynamic\_cast.

**See also**

[Fl\\_Widget::as\\_group\(\)](#), [Fl\\_Widget::as\\_window\(\)](#)

Reimplemented in [Fl\\_Gl\\_Window](#).

**31.146.4.9 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**

|                   |                                                               |
|-------------------|---------------------------------------------------------------|
| <code>NULL</code> | if this widget is not derived from <a href="#">Fl_Group</a> . |
|-------------------|---------------------------------------------------------------|

**Note**

This method is provided to avoid dynamic\_cast.

**See also**

[Fl\\_Widget::as\\_window\(\)](#), [Fl\\_Widget::as\\_gl\\_window\(\)](#)

Reimplemented in [Fl\\_Group](#).

#### 31.146.4.10 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.

##### Returns

|                   |                                                                |
|-------------------|----------------------------------------------------------------|
| <code>NULL</code> | if this widget is not derived from <a href="#">Fl_Window</a> . |
|-------------------|----------------------------------------------------------------|

##### Note

This method is provided to avoid dynamic\_cast.

##### See also

[Fl\\_Widget::as\\_group\(\)](#), [Fl\\_Widget::as\\_gl\\_window\(\)](#)

Reimplemented in [Fl\\_Window](#).

#### 31.146.4.11 box() [1/2]

```
Fl_Boxtype Fl_Widget::box () const [inline]
```

Gets the box type of the widget.

##### Returns

the current box type

##### See also

[box\(Fl\\_Boxtype\)](#), [Fl\\_Boxtype](#)

#### 31.146.4.12 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 | <i>new_box</i> | the new box type |
|----|----------------|------------------|

**See also**

[box\(\)](#), [Fl\\_Boxtype](#)

**31.146.4.13 callback() [1/5]**

```
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

**31.146.4.14 callback() [2/5]**

```
void Fl_Widget::callback (
 Fl_Callback * cb,
 void * p) [inline]
```

Sets the current callback function for the widget.

Each widget has a single callback.

**Parameters**

|    |           |              |
|----|-----------|--------------|
| in | <i>cb</i> | new callback |
| in | <i>p</i>  | user data    |

**31.146.4.15 callback() [3/5]**

```
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 |
|----|----|--------------|

**31.146.4.16 callback() [4/5]**

```
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 |
|----|----|--------------|

**31.146.4.17 callback() [5/5]**

```
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    |

**31.146.4.18 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\(Fl\\_Widget \\*widget, void \\*data\)](#)

**31.146.4.19 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\(\)](#)

**31.146.4.20 clear\_changed()**

```
void Fl_Widget::clear_changed () [inline]
```

Marks the value of the widget as unchanged.

**See also**

[changed\(\)](#), [set\\_changed\(\)](#)

**31.146.4.21 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\(\)](#)**31.146.4.22 clear\_output()**

```
void Fl_Widget::clear_output () [inline]
```

Sets a widget to accept input.

**See also**[set\\_output\(\)](#), [output\(\)](#)**31.146.4.23 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.

**31.146.4.24 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\(int\)](#)

**31.146.4.25 color()** [1/3]

```
Fl_Color Fl_Widget::color () const [inline]
```

Gets the background color of the widget.

**Returns**

current background color

**See also**

[color\(Fl\\_Color\)](#), [color\(Fl\\_Color, Fl\\_Color\)](#)

**31.146.4.26 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\(Fl\\_Color\)](#)

**31.146.4.27 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 | <i>bg</i>  | background color |
| in | <i>sel</i> | selection color  |

**See also**

[color\(unsigned\)](#), [selection\\_color\(unsigned\)](#)

**31.146.4.28 color2() [1/2]**

`Fl_Color Fl_Widget::color2 ( ) const [inline]`

For back compatibility only.

**Deprecated** Use [selection\\_color\(\)](#) instead.

**31.146.4.29 color2() [2/2]**

`void Fl_Widget::color2 ( unsigned a ) [inline]`

For back compatibility only.

**Deprecated** Use [selection\\_color\(unsigned\)](#) instead.

**31.146.4.30 contains()**

`int Fl_Widget::contains ( const Fl_Widget * w ) const`

Checks if w is a child of this widget.

**Parameters**

|    |          |                        |
|----|----------|------------------------|
| in | <i>w</i> | 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.

**31.146.4.31 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 | <i>new_label</i> | the new label text |
|----|------------------|--------------------|

**See also**

[label\(\)](#)

**31.146.4.32 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 | <i>text</i> | New tooltip text (an internal copy is made and managed) |
|----|-------------|---------------------------------------------------------|

**See also**

[tooltip\(const char\\*\)](#), [tooltip\(\)](#)

**31.146.4.33 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\(uchar\)](#)

**31.146.4.34 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\(uchar\)](#)

**31.146.4.35 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 | <i>c</i>       | bitmask of flags to set |
| in | <i>x,y,w,h</i> | size of damaged area    |

**See also**

[damage\(\)](#), [clear\\_damage\(uchar\)](#)

**31.146.4.36 damage\_resize()**

```
int Fl_Widget::damage_resize (
 int X,
 int Y,
 int W,
 int H)
```

Internal use only.

**31.146.4.37 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\(\)](#)

**31.146.4.38 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 | <i>widget</i> | the <a href="#">Fl_Widget</a> given to the callback |
| in | <i>data</i>   | user data associated with that callback             |

**See also**

[callback\(\)](#), [Fl::readqueue\(\)](#)  
[do\\_callback\(Fl\\_Widget \\*widget, void \\*data\)](#)

**31.146.4.39 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

**31.146.4.40 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

**31.146.4.41 deimage() [3/4]**

`void Fl_Widget::deimage (`  
    `Fl_Image * img ) [inline]`

Sets the image to use as part of the widget label when in the inactive state.

**Parameters**

|    |            |                                          |
|----|------------|------------------------------------------|
| in | <i>img</i> | 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.

**31.146.4.42 deimage() [4/4]**

```
void Fl_Widget::deimage (
 Fl_Image & img) [inline]
```

Sets the image to use as part of the widget label when in the inactive state.

**Parameters**

|    |            |                                          |
|----|------------|------------------------------------------|
| in | <i>img</i> | the new image for the deactivated widget |
|----|------------|------------------------------------------|

**See also**

[void deimage\(Fl\\_Image\\* img\)](#)

**31.146.4.43 do\_callback() [1/3]**

```
void Fl_Widget::do_callback () [inline]
```

Calls the widget callback function with default arguments.

This is the same as calling

```
do_callback(this, user_data());
```

**See also**

[callback\(\)](#)  
[do\\_callback\(Fl\\_Widget \\*widget, void \\*data\)](#)

**31.146.4.44 do\_callback() [2/3]**

```
void Fl_Widget::do_callback (
 Fl_Widget * widget,
 long arg) [inline]
```

Calls the widget callback function with arbitrary arguments.

**Parameters**

|    |               |                                                                      |
|----|---------------|----------------------------------------------------------------------|
| in | <i>widget</i> | call the callback with <i>widget</i> as the first argument           |
| in | <i>arg</i>    | call the callback with <i>arg</i> as the user data (second) argument |

**See also**

[callback\(\)](#)  
[do\\_callback\(Fl\\_Widget \\*widget, void \\*data\)](#)

**31.146.4.45 do\_callback() [3/3]**

```
void Fl_Widget::do_callback (
 Fl_Widget * widget,
 void * arg = 0)
```

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.

**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 | <i>widget</i> | call the callback with <i>widget</i> as the first argument |
| in | <i>arg</i>    | use <i>arg</i> as the user data (second) argument          |

**See also**

[default\\_callback\(\)](#)  
[callback\(\)](#)  
 class [Fl\\_Widget\\_Tracker](#)

**31.146.4.46 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\\_FormsText](#), [Fl\\_Table](#), [Fl\\_Text\\_Display](#), [Fl\\_Tree](#), [Fl\\_Input](#), [Fl\\_Help\\_View](#), [Fl\\_Tabs](#), [Fl\\_Browser](#), [Fl\\_Simple\\_Terminal](#), [Fl\\_Scroll](#), [Fl\\_Choice](#), [Fl\\_Sys\\_Menu\\_Bar](#), [Fl\\_Window](#), [Fl\\_Button](#), [Fl\\_Chart](#), [Fl\\_Clock\\_Output](#), [Fl\\_Slider](#), [Fl\\_GI\\_Window](#), [Fl\\_Menu\\_Bar](#), [Fl\\_Value\\_Input](#), [Fl\\_Pack](#), [Fl\\_File\\_Input](#), [Fl\\_Group](#), [Fl\\_Counter](#), [Fl\\_Free](#), [Fl\\_Cairo\\_Window](#), [Fl\\_Menu\\_Button](#), [Fl\\_Dial](#), [Fl\\_Scrollbar](#), [Fl\\_Positioner](#), [Fl\\_Adjuster](#), [Fl\\_Timer](#), [Fl\\_Value\\_Output](#), [Fl\\_Wizard](#), [Fl\\_Glut\\_Window](#), [Fl\\_Progress](#), [Fl\\_Roller](#), [Fl\\_Light\\_Button](#), [Fl\\_Value\\_Slider](#), [Fl\\_Box](#), [Fl\\_Return\\_Button](#), [Fl\\_FormsPixmap](#), and [Fl\\_FormsBitmap](#).

#### 31.146.4.47 draw\_box() [1/2]

```
void Fl_Widget::draw_box (
 Fl_Boxtype t,
 Fl_Color c) const [protected]
```

Draws a box of type t, of color c at the widget's position and size.

#### 31.146.4.48 draw\_box() [2/2]

```
void Fl_Widget::draw_box (
 Fl_Boxtype t,
 int X,
 int Y,
 int W,
 int H,
 Fl_Color c) const [protected]
```

Draws a box of type t, of color c at the position X,Y and size W,H.

#### 31.146.4.49 draw\_focus()

```
void Fl_Widget::draw_focus (
 Fl_Boxtype B,
 int X,
 int Y,
 int W,
 int H) const [protected]
```

Draws a focus box for the widget at the given position and size.

**31.146.4.50 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.

**31.146.4.51 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

**31.146.4.52 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.

**31.146.4.53 h() [1/2]**

```
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.

**31.146.4.54 h() [2/2]**

```
int Fl_Widget::h() const [inline]
```

Gets the widget height.

**Returns**

the height of the widget in pixels.

### 31.146.4.55 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.

#### Parameters

|    |              |                            |
|----|--------------|----------------------------|
| in | <i>event</i> | 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\\_Tree](#), [Fl\\_Help\\_View](#), [Fl\\_Input](#), [Fl\\_Table](#), [Fl\\_Tabs](#), [Fl\\_Browser\\_](#), [Fl\\_Window](#), [Fl\\_Clock](#), [Fl\\_Text\\_Display](#), [Fl\\_Scroll](#), [Fl\\_Color\\_Chooser](#), [Fl\\_Table\\_Row](#), [Fl\\_Check\\_Browser](#), [Fl\\_Choice](#), [Fl\\_Button](#), [Fl\\_GI\\_Window](#), [Fl\\_Slider](#), [Fl\\_Menu\\_Button](#), [Fl\\_Spinner](#), [Fl\\_Group](#), [Fl\\_Menu\\_Bar](#), [Fl\\_Value\\_Input](#), [Fl\\_Counter](#), [Fl\\_Free](#), [Fl\\_Spinner::Fl\\_Spinner\\_Input](#), [Fl\\_File\\_Input](#), [Fl\\_Dial](#), [Fl\\_Scrollbar](#), [Fl\\_Text\\_Editor](#), [Fl\\_Positioner](#), [Fl\\_Box](#), [Fl\\_Value\\_Output](#), [Fl\\_Timer](#), [Fl\\_Glut\\_Window](#), [Fl\\_Adjuster](#), [Fl\\_Roller](#), [Fl\\_Secret\\_Input](#), [Fl\\_Light\\_Button](#), [Fl\\_Value\\_Slider](#), [Fl\\_Return\\_Button](#), [Fl\\_Repeat\\_Button](#), and [Fl\\_Tile](#).

### 31.146.4.56 hide()

```
void Fl_Widget::hide () [virtual]
```

Makes a widget invisible.

#### See also

[show\(\)](#), [visible\(\)](#), [visible\\_r\(\)](#)

Reimplemented in [Fl\\_Window](#), [Fl\\_Browser](#), [Fl\\_GI\\_Window](#), [Fl\\_Overlay\\_Window](#), [Fl\\_Double\\_Window](#), and [Fl\\_Menu\\_Window](#).

**31.146.4.57 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

**31.146.4.58 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

**31.146.4.59 image() [3/4]**

```
void Fl_Widget::image (
 Fl_Image * img) [inline]
```

Sets the image to use as part of the widget label when in the active state.

**Parameters**

|    |            |                             |
|----|------------|-----------------------------|
| in | <i>img</i> | the new image for the label |
|----|------------|-----------------------------|

**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.

**31.146.4.60 image() [4/4]**

```
void Fl_Widget::image (
 Fl_Image & img) [inline]
```

Sets the image to use as part of the widget label when in the active state.

**Parameters**

|    |            |                             |
|----|------------|-----------------------------|
| in | <i>img</i> | the new image for the label |
|----|------------|-----------------------------|

**See also**

[void image\(Fl\\_Image\\* img\)](#)

**31.146.4.61 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 | <i>wgt</i> | the possible parent widget. |
|----|------------|-----------------------------|

**See also**

[contains\(\)](#)

**31.146.4.62 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 <a href="#">label()</a> .      |
| 1 | current label was assigned with <a href="#">copy_label()</a> . |

**31.146.4.63 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\(const char \\*\)](#)

**31.146.4.64 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 | <i>text</i> | pointer to new label text |
|----|-------------|---------------------------|

**See also**

[copy\\_label\(\)](#)

**31.146.4.65 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\(Fl\\_Labeltype\)](#)

**31.146.4.66 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**

|                |                                            |
|----------------|--------------------------------------------|
| <code>t</code> | text or label to search for '&x' shortcut. |
|----------------|--------------------------------------------|

**Returns**

Unicode (UCS-4) value of shortcut in `t` or 0.

**Note**

Internal use only.

**31.146.4.67 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

**31.146.4.68 labelcolor() [2/2]**

```
void Fl_Widget::labelcolor (
 Fl_Color c) [inline]
```

Sets the label color.

The default color is FL\_FOREGROUND\_COLOR.

**Parameters**

|                 |                |                     |
|-----------------|----------------|---------------------|
| <code>in</code> | <code>c</code> | the new label color |
|-----------------|----------------|---------------------|

**31.146.4.69 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](#)

### 31.146.4.70 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](#)

### 31.146.4.71 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

### 31.146.4.72 labelsize() [2/2]

```
void Fl_Widget::labelsize (
 Fl_Fontsize pix) [inline]
```

Sets the font size in pixels.

**Parameters**

|    |            |                   |
|----|------------|-------------------|
| in | <i>pix</i> | the new font size |
|----|------------|-------------------|

**See also**

[Fl\\_Fontsize labels\(\)](#)

**31.146.4.73 labeltype() [1/2]**

`Fl_Labeltype Fl_Widget::labeltype () const [inline]`

Gets the label type.

**Returns**

the current label type.

**See also**

[Fl\\_Labeltype](#)

**31.146.4.74 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_NORM`←  
`AL_LABEL` prints the label as plain text.

**Parameters**

|    |          |                |
|----|----------|----------------|
| in | <i>a</i> | new label type |
|----|----------|----------------|

**See also**

[Fl\\_Labeltype](#)

**31.146.4.75 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\(const char\\*, int&, int&, int\)](#)

**31.146.4.76 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**

|                |                                            |
|----------------|--------------------------------------------|
| <code>0</code> | if the widget is used for input and output |
|----------------|--------------------------------------------|

**See also**

[set\\_output\(\)](#), [clear\\_output\(\)](#)

**31.146.4.77 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**

|                   |                             |
|-------------------|-----------------------------|
| <code>NULL</code> | if the widget has no parent |
|-------------------|-----------------------------|

## See also

[Fl\\_Group::add\(Fl\\_Widget\\*\)](#)

### 31.146.4.78 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.

### 31.146.4.79 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\(int,int\)](#)

### 31.146.4.80 redraw()

```
void Fl_Widget::redraw ()
```

Schedules the drawing of the widget.

Marks the widget as needing its [draw\(\)](#) routine called.

### 31.146.4.81 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.

31.146.4.82 `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\\_Table](#), [Fl\\_Text\\_Display](#), [Fl\\_Tree](#), [Fl\\_Help\\_View](#), [Fl\\_Browser\\_](#), [Fl\\_Input\\_](#), [Fl\\_Window](#), [Fl\\_Scroll](#), [Fl\\_Group](#), [Fl\\_Input\\_Choice](#), [Fl\\_Gl\\_Window](#), [Fl\\_Spinner](#), [Fl\\_Value\\_Input](#), [Fl\\_Overlay\\_Window](#), [Fl\\_Double\\_Window](#), and [Fl\\_Tile](#).

31.146.4.83 `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\(Fl\\_Color, Fl\\_Color\)](#)

31.146.4.84 `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\(Fl\\_Color, Fl\\_Color\)](#)

**31.146.4.85 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\(\)](#)

**31.146.4.86 set\_changed()**

```
void Fl_Widget::set_changed () [inline]
```

Marks the value of the widget as changed.

**See also**

[changed\(\)](#), [clear\\_changed\(\)](#)

**31.146.4.87 set\_output()**

```
void Fl_Widget::set_output () [inline]
```

Sets a widget to output only.

**See also**

[output\(\)](#), [clear\\_output\(\)](#)

**31.146.4.88 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.

**31.146.4.89 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\(int\)](#)

**31.146.4.90 shortcut\_label() [1/2]**

```
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.

**31.146.4.91 shortcut\_label() [2/2]**

```
int Fl_Widget::shortcut_label () const [inline]
```

Returns whether the widget's label uses '&' to indicate shortcuts.

**See also**

[void shortcut\\_label\(int value\)](#)

### 31.146.4.92 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\\_Window](#), [Fl\\_Browser](#), [Fl\\_Gl\\_Window](#), [Fl\\_Overlay\\_Window](#), [Fl\\_Double\\_Window](#), [Fl\\_Single\\_Window](#), and [Fl\\_Menu\\_Window](#).

### 31.146.4.93 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\(int,int,int,int\)](#)

### 31.146.4.94 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.

**31.146.4.95 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**

|          |                               |
|----------|-------------------------------|
| <i>0</i> | if the widget takes no events |
|----------|-------------------------------|

**31.146.4.96 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**

Internal use only.

**31.146.4.97 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**

|                    |                                            |
|--------------------|--------------------------------------------|
| <i>t</i>           | text or label to search for '&x' shortcut. |
| <i>require_alt</i> | 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**

Internal use only.

**31.146.4.98 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\(const char\\*\)](#)

**31.146.4.99 tooltip() [2/2]**

```
void Fl_Widget::tooltip(
 const char * text)
```

Sets the current tooltip text.

Sets a string of text to display in a popup tooltip window when the user hovers the mouse over the widget. The string is *not* copied, so make sure any formatted string is stored in a static, global, or allocated buffer. If you want a copy made and managed for you, use the [copy\\_tooltip\(\)](#) method, which will manage the tooltip string automatically.

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 | <i>text</i> | New tooltip text (no copy is made) |
|----|-------------|------------------------------------|

**See also**[copy\\_tooltip\(const char\\*\), tooltip\(\)](#)**31.146.4.100 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\(\)](#)**31.146.4.101 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)

**31.146.4.102 type() [1/2]**

```
uchar Fl_Widget::type () const [inline]
```

Gets the widget type.

Returns the widget type value, which is used for Forms compatibility and to simulate RTTI.

**Todo** Explain "simulate RTTI" (currently only used to decide if a widget is a window, i.e. `type()>=FL_WINDOW ?`).  
Is `type()` really used in a way that ensures "Forms compatibility" ?

#### 31.146.4.103 `type()` [2/2]

```
void Fl_Widget::type (
 uchar t) [inline]
```

Sets the widget type.

This is used for Forms compatibility.

#### 31.146.4.104 `user_data()` [1/2]

```
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

#### 31.146.4.105 `user_data()` [2/2]

```
void Fl_Widget::user_data (
 void * v) [inline]
```

Sets the user data for this widget.

Sets the new user data (`void *`) argument that is passed to the callback function.

##### Parameters

|    |   |               |
|----|---|---------------|
| in | v | new user data |
|----|---|---------------|

#### 31.146.4.106 `visible()`

```
unsigned int Fl_Widget::visible () const [inline]
```

Returns whether a widget is visible.

**Return values**

|          |                                                 |
|----------|-------------------------------------------------|
| <i>O</i> | if the widget is not drawn and hence invisible. |
|----------|-------------------------------------------------|

**See also**

[show\(\)](#), [hide\(\)](#), [visible\\_r\(\)](#)

**31.146.4.107 visible\_focus() [1/2]**

```
void Fl_Widget::visible_focus (
 int v) [inline]
```

Modifies keyboard focus navigation.

**Parameters**

|    |          |                            |
|----|----------|----------------------------|
| in | <i>v</i> | set or clear visible focus |
|----|----------|----------------------------|

**See also**

[set\\_visible\\_focus\(\)](#), [clear\\_visible\\_focus\(\)](#), [visible\\_focus\(\)](#)

**31.146.4.108 visible\_focus() [2/2]**

```
unsigned int Fl_Widget::visible_focus () [inline]
```

Checks whether this widget has a visible focus.

**Return values**

|          |                                      |
|----------|--------------------------------------|
| <i>O</i> | if this widget has no visible focus. |
|----------|--------------------------------------|

**See also**

[visible\\_focus\(int\)](#), [set\\_visible\\_focus\(\)](#), [clear\\_visible\\_focus\(\)](#)

**31.146.4.109 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\(\)](#)

**31.146.4.110 w()** [1/2]

```
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.

**31.146.4.111 w()** [2/2]

```
int Fl_Widget::w () const [inline]
```

Gets the widget width.

**Returns**

the width of the widget in pixels.

**31.146.4.112 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\(uchar\)](#)

## 31.146.4.113 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\\_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>i</i> | set of flags |
|----|----------|--------------|

## 31.146.4.114 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\\_Widget](#) widget, this returns its *parent* window (if any), not *this* window.

**See also**

[top\\_window\(\)](#)

**31.146.4.115 x()** [1/2]

```
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.

**31.146.4.116 x()** [2/2]

```
int Fl_Widget::x () const [inline]
```

Gets the widget position in its window.

**Returns**

the x position relative to the window

**31.146.4.117 y()** [1/2]

```
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.

**31.146.4.118 y()** [2/2]

```
int Fl_Widget::y () const [inline]
```

Gets the widget position in its window.

**Returns**

the y position relative to the window

The documentation for this class was generated from the following files:

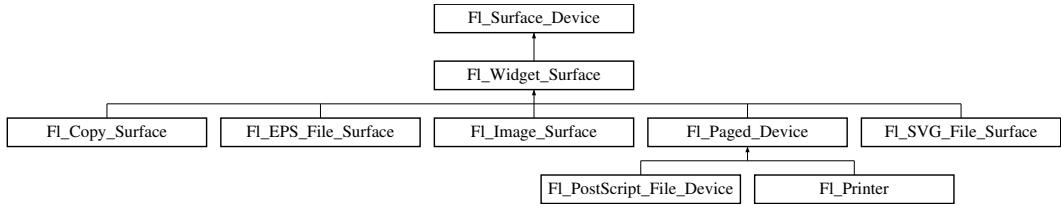
- [Fl\\_Widget.H](#)
- [Fl.cxx](#)
- [fl\\_boxtype.cxx](#)
- [fl\\_labeltype.cxx](#)
- [fl\\_shortcut.cxx](#)
- [Fl\\_Tooltip.cxx](#)
- [Fl\\_Widget.cxx](#)
- [Fl\\_Window.cxx](#)

## 31.147 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.*
- virtual 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 \(int x, int y\)](#)  
*Sets the position of the origin of graphics in the drawable part of the drawing surface.*
- virtual void [origin \(int \\*x, int \\*y\)](#)  
*Computes the coordinates of the current origin of graphics functions.*
- 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.*

### Protected Member Functions

- [Fl\\_Widget\\_Surface \(Fl\\_Graphics\\_Driver \\*d\)](#)  
*The constructor.*

### 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

### 31.147.1 Detailed Description

A surface on which any FLTK widget can be drawn.

### 31.147.2 Constructor & Destructor Documentation

#### 31.147.2.1 Fl\_Widget\_Surface()

```
Fl_Widget_Surface::Fl_Widget_Surface (
 Fl_Graphics_Driver * d) [protected]
```

The constructor.

#### Parameters

|          |             |
|----------|-------------|
| <i>d</i> | can be nul. |
|----------|-------------|

### 31.147.3 Member Function Documentation

#### 31.147.3.1 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 | <i>widget</i>          | Any FLTK widget (e.g., standard, custom, window).                                                                          |
| in | <i>delta_x,delta_y</i> | Optional horizontal and vertical offsets for positioning the widget top left relatively to the current origin of graphics. |

## 31.147.3.2 draw\_decorated\_window()

```
void Fl_Widget_Surface::draw_decorated_window (
 Fl_Window * win,
 int win_offset_x = 0,
 int win_offset_y = 0) [virtual]
```

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.

Reimplemented in `Fl_Printer`, and `Fl_Copy_Surface`.

## 31.147.3.3 origin() [1/2]

```
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_EPS_File_Surface`, `Fl_PostScript_File_Device`, `Fl_Printer`, `Fl_Image_Surface`, `Fl_Copy_Surface`, and `Fl_SVG_File_Surface`.

## 31.147.3.4 origin() [2/2]

```
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, <code>*x</code> and <code>*y</code> are set to the horizontal and vertical coordinates of the graphics origin. |
|-----|-----|-----------------------------------------------------------------------------------------------------------------------------|

Reimplemented in [Fl\\_EPS\\_File\\_Surface](#), [Fl\\_PostScript\\_File\\_Device](#), [Fl\\_Printer](#), [Fl\\_Image\\_Surface](#), and [Fl\\_Copy\\_Surface](#).

### 31.147.3.5 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

|                                 |                                                                                                                                        |
|---------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <i>win</i>                      | The window from where to capture. Can be an <a href="#">Fl_GI_Window</a> . Sub-windows that intersect the rectangle are also captured. |
| <i>x</i>                        | The rectangle left                                                                                                                     |
| <i>y</i>                        | The rectangle top                                                                                                                      |
| <i>w</i>                        | The rectangle width                                                                                                                    |
| <i>h</i>                        | The rectangle height                                                                                                                   |
| <i>delta_x</i> , <i>delta_y</i> | Optional horizontal and vertical offsets from current graphics origin where to draw the top left of the captured rectangle.            |

### 31.147.3.6 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\\_EPS\\_File\\_Surface](#), [Fl\\_PostScript\\_File\\_Device](#), [Fl\\_Printer](#), [Fl\\_Image\\_Surface](#), [Fl\\_SVG\\_File\\_Surface](#), and [Fl\\_Copy\\_Surface](#).

## 31.147.3.7 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\\_EPS\\_File\\_Surface](#), [Fl\\_PostScript\\_File\\_Device](#), [Fl\\_Printer](#), [Fl\\_Image\\_Surface](#), [Fl\\_SVG\\_File\\_Surface](#), and [Fl\\_Copy\\_Surface](#).

The documentation for this class was generated from the following files:

- [Fl\\_Widget\\_Surface.H](#)
- [Fl\\_Widget\\_Surface.cxx](#)

## 31.148 Fl\_Widget\_Tracker Class Reference

This class should be used to control safe widget deletion.

```
#include <Fl.H>
```

### Public Member Functions

- int [deleted \(\)](#)  
*Returns 1, if the watched widget has been deleted.*
- int [exists \(\)](#)  
*Returns 1, if the watched widget exists (has not been deleted).*
- [Fl\\_Widget\\_Tracker \(Fl\\_Widget \\*wi\)](#)  
*The constructor adds a widget to the watch list.*
- [Fl\\_Widget \\* widget \(\)](#)  
*Returns a pointer to the watched widget.*
- [~Fl\\_Widget\\_Tracker \(\)](#)  
*The destructor removes a widget from the watch list.*

### 31.148.1 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();
 }
}
```

### 31.148.2 Member Function Documentation

#### 31.148.2.1 deleted()

```
int Fl_Widget_Tracker::deleted () [inline]
```

Returns 1, if the watched widget has been deleted.

This is a convenience method. You can also use something like

```
if (wp.widget() == 0) // ...
```

where `wp` is an [Fl\\_Widget\\_Tracker](#) object.

## 31.148.2.2 exists()

```
int Fl_Widget_Tracker::exists () [inline]
```

Returns 1, if the watched widget exists (has not been deleted).

This is a convenience method. You can also use something like

```
if (wp.widget() != 0) // ...
```

where wp is an [Fl\\_Widget\\_Tracker](#) object.

## 31.148.2.3 widget()

```
Fl_Widget* Fl_Widget_Tracker::widget () [inline]
```

Returns a pointer to the watched widget.

This pointer is NULL, if the widget has been deleted.

The documentation for this class was generated from the following files:

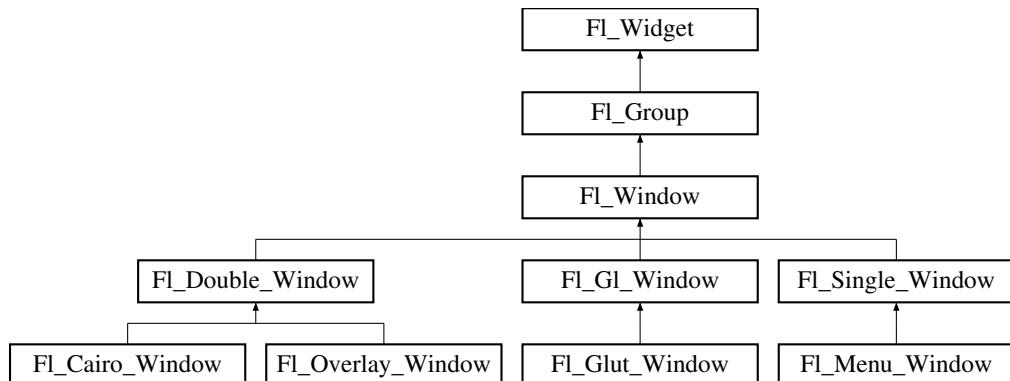
- [Fl.H](#)
- [Fl.cxx](#)

## 31.149 Fl\_Window Class Reference

This widget produces an actual window.

```
#include <Fl_Window.H>
```

Inheritance diagram for Fl\_Window:



## Public Member Functions

- virtual class `FI_Double_Window * as_double_window ()`  
*Return non-null if this is an `FI_Double_Window` object.*
- virtual class `FI_Overlay_Window * as_overlay_window ()`  
*Return non-null if this is an `FI_Overlay_Window` object.*
- virtual `FI_Window * as_window ()`  
*Returns an `FI_Window` pointer if this widget is an `FI_Window`.*
- void `border` (int b)  
*Sets whether or not the window manager border is around the window.*
- unsigned int `border` () const  
*Returns whether the window possesses a border.*
- 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` (`FI_Cursor`)  
*Changes the cursor for this window.*
- void `cursor` (const `FI_RGB_Image` \*, int, int)  
*Changes the cursor for this window.*
- void `cursor` (`FI_Cursor` c, `FI_Color`, `FI_Color`=`FL_WHITE`)  
*For back compatibility only.*
- 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` (`FI_Cursor`)  
*Sets the default window cursor.*
- void `default_cursor` (`FI_Cursor` c, `FI_Color`, `FI_Color`=`FL_WHITE`)  
*For back compatibility only.*
- `FI_Window` (int w, int h, const char \*title=0)  
*Creates a window from the given width w, height h, and title.*
- `FI_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.*
- virtual int `handle` (int)  
*Handles the specified event.*

- virtual void **hide** ()  
*Removes the window from the screen.*
- 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.*
- void **hotspot** (const Fl\_Widget \*, int offscreen=0)  
*See void Fl\_Window::hotspot(int x, int y, int offscreen = 0)*
- void **hotspot** (const Fl\_Widget &p, int offscreen=0)  
*See void Fl\_Window::hotspot(int x, int y, int offscreen = 0)*
- void **icon** (const Fl\_RGB\_Image \*)  
*Sets or resets a single window icon.*
- const void \* **icon** () const  
*Gets the current icon window target dependent data.*
- void **icon** (const void \*ic)  
*Sets the current icon window target dependent data.*
- 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.*
- 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.*
- 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.*
- unsigned int **override** () const  
*Returns non zero if FL\_OVERRIDE flag is set, 0 otherwise.*
- virtual void **resize** (int X, int Y, int W, int H)  
*Changes the size and position of the window.*
- int **screen\_num** ()  
*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|FL\_OVERRIDE.
  - **void set\_tooltip\_window ()**  
Marks the window as a tooltip window.
  - **void shape (const FL\_Image \*img)**  
Assigns a non-rectangular shape to the window.
  - **void shape (const FL\_Image &b)**  
Set the window's shape with an [FL\\_Image](#).
  - **const FL\_Image \* shape ()**  
Returns the image controlling the window shape or NULL.
  - **virtual void show ()**  
Puts the window on the screen.
  - **void show (int argc, char \*\*argv)**  
Puts the window on the screen and parses command-line arguments.
  - **int shown ()**  
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 the user can resize this window to.
  - **unsigned int tooltip\_window () const**  
Returns true if this window is a tooltip window.
  - **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.

## 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\_xclass (const char \*)**  
Sets the default window xclass.
- **static const char \* default\_xclass ()**  
Returns the default xclass.
- **static bool is\_a\_rescale ()**  
Returns true when a window is being rescaled.

## Protected Member Functions

- void [draw \(\)](#)  
*Draws the widget.*
- virtual void [flush \(\)](#)  
*Forces the window to be drawn, this window is also made current and calls draw().*
- void [force\\_position \(int force\)](#)  
*Sets an internal flag that tells FLTK and the window manager to honor position requests.*
- int [force\\_position \(\) const](#)  
*Returns the internal state of the window's FORCE\_POSITION flag.*
- void [free\\_icons \(\)](#)  
*Deletes all icons previously attached to the window.*

## Static Protected Attributes

- static [Fl\\_Window \\* current\\_](#)  
*Stores the last window that was made current.*

## Friends

- int [Fl::arg \(int argc, char \\*\\*argv, int &i\)](#)
- class [Fl\\_Window\\_Driver](#)
- class [Fl\\_X](#)

## Additional Inherited Members

### 31.149.1 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\(\)](#).

### 31.149.2 Constructor & Destructor Documentation

### 31.149.2.1 [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\(int x, int y, int w, int h, const char \\*title\)](#)

### 31.149.2.2 [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`.

#### See also

[Fl\\_Window\(int w, int h, const char \\*title\)](#)

### 31.149.2.3 [~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.

### 31.149.3 Member Function Documentation

#### 31.149.3.1 as\_window()

```
virtual Fl_Window* Fl_Window::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

|                   |                                                                |
|-------------------|----------------------------------------------------------------|
| <code>NULL</code> | if this widget is not derived from <a href="#">Fl_Window</a> . |
|-------------------|----------------------------------------------------------------|

##### Note

This method is provided to avoid dynamic\_cast.

##### See also

[Fl\\_Widget::as\\_group\(\)](#), [Fl\\_Widget::as\\_gl\\_window\(\)](#)

Reimplemented from [Fl\\_Widget](#).

#### 31.149.3.2 border()

```
void Fl_Window::border (
 int b)
```

Sets whether or not the window manager border is around the window.

The default value is true. *With some X window managers, this does not work after [show\(\)](#) has been called.*

#### 31.149.3.3 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.

### 31.149.3.4 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\(\)](#)

### 31.149.3.5 current()

```
Fl_Window * Fl_Window::current () [static]
```

Returns the last window that was made current.

#### See also

[Fl\\_Window::make\\_current\(\)](#)

**31.149.3.6 cursor() [1/3]**

```
void Fl_Window::cursor (
 Fl_Cursor c)
```

Changes the cursor for this window.

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\(\)](#)

**31.149.3.7 cursor() [2/3]**

```
void Fl_Window::cursor (
 const Fl_RGB_Image * image,
 int hotx,
 int hoty)
```

Changes the cursor for this window.

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.

**See also**

[cursor\(Fl\\_Cursor\)](#), [default\\_cursor\(\)](#)

**31.149.3.8 cursor() [3/3]**

```
void Fl_Window::cursor (
 Fl_Cursor c,
 Fl_Color ,
 Fl_Color = FL_WHITE)
```

For back compatibility only.

Same as `Fl_Window::cursor(Fl_Cursor)`

**31.149.3.9 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.

**31.149.3.10 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.

**31.149.3.11 default\_cursor() [1/2]**

```
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\(\)](#)

**31.149.3.12 default\_cursor() [2/2]**

```
void Fl_Window::default_cursor (
 Fl_Cursor c,
 Fl_Color ,
 Fl_Color = FL_WHITE)
```

For back compatibility only.

same as [Fl\\_Window::default\\_cursor\(Fl\\_Cursor\)](#)

**31.149.3.13 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 | <i>icon</i> | 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\)](#)

**31.149.3.14 default\_icons()**

```
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 | <i>icons</i> | default icons for all windows subsequently created                                    |
| in | <i>count</i> | number of images in <code>icons</code> . 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\)](#)

**31.149.3.15 default\_xclass() [1/2]**

```
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 | <code>xc</code> | default xclass for all windows subsequently created |
|----|-----------------|-----------------------------------------------------|

#### See also

[Fl\\_Window::xclass\(const char \\*\)](#)

### 31.149.3.16 default\_xclass() [2/2]

```
const char * Fl_Window::default_xclass () [static]
```

Returns the default xclass.

#### See also

[Fl\\_Window::default\\_xclass\(const char \\*\)](#)

### 31.149.3.17 draw()

```
void Fl_Window::draw () [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](#).

Reimplemented in [Fl\\_Glut\\_Window](#).

**31.149.3.18 flush()**

```
void Fl_Window::flush () [protected], [virtual]
```

Forces the window to be drawn, this window is also made current and calls [draw\(\)](#).

Reimplemented in [Fl\\_Gl\\_Window](#), [Fl\\_Overlay\\_Window](#), [Fl\\_Double\\_Window](#), and [Fl\\_Menu\\_Window](#).

**31.149.3.19 force\_position() [1/2]**

```
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 | <i>force</i> | 1 to set the FORCE_POSITION flag, 0 to clear it |
|----|--------------|-------------------------------------------------|

**31.149.3.20 force\_position() [2/2]**

```
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\(int\)](#)

**31.149.3.21 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\)](#)

### 31.149.3.22 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 for Forms compatibility only.

**Deprecated** please use [force\\_position\(0\)](#) instead

### 31.149.3.23 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\(\)](#)

### 31.149.3.24 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.

#### See also

[void Fl\\_Window::fullscreen\(\)](#)

### 31.149.3.25 handle()

```
int Fl_Window::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.

**Parameters**

|                 |                    |                            |
|-----------------|--------------------|----------------------------|
| <code>in</code> | <code>event</code> | the kind of event received |
|-----------------|--------------------|----------------------------|

**Return values**

|                |                                          |
|----------------|------------------------------------------|
| <code>0</code> | if the event was not used or understood  |
| <code>1</code> | if the event was used and can be deleted |

**See also**

[Fl\\_Event](#)

Reimplemented from [Fl\\_Group](#).

Reimplemented in [Fl\\_Gl\\_Window](#), and [Fl\\_Glut\\_Window](#).

**31.149.3.26 hide()**

```
void Fl_Window::hide () [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](#).

Reimplemented in [Fl\\_Gl\\_Window](#), [Fl\\_Overlay\\_Window](#), [Fl\\_Double\\_Window](#), and [Fl\\_Menu\\_Window](#).

**31.149.3.27 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\(\)](#)

## 31.149.3.28 icon() [1/3]

```
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.

**Parameters**

|                 |                   |                                                  |
|-----------------|-------------------|--------------------------------------------------|
| <code>in</code> | <code>icon</code> | 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\)](#)

**31.149.3.29 icon()** [2/3]

```
const void * Fl_Window::icon () const
```

Gets the current icon window target dependent data.

**Deprecated** in 1.3.3

**31.149.3.30 icon()** [3/3]

```
void Fl_Window::icon (
 const void * ic)
```

Sets the current icon window target dependent data.

**Deprecated** in 1.3.3

**31.149.3.31 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.

**31.149.3.32 iconlabel()**

```
void Fl_Window::iconlabel (
 const char * iname)
```

Sets the icon label.

**31.149.3.33 icons()**

```
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**

|           |              |                                                                         |
|-----------|--------------|-------------------------------------------------------------------------|
| <i>in</i> | <i>icons</i> | icons for this window                                                   |
| <i>in</i> | <i>count</i> | number of images in <i>icons</i> . 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\(const Fl\\_RGB\\_Image \\*\)](#)

**31.149.3.34 label() [1/2]**

```
void Fl_Window::label (
 const char * name)
```

Sets the window title bar label.

**31.149.3.35 label() [2/2]**

```
void Fl_Window::label (
 const char * label,
 const char * iconlabel)
```

Sets the icon label.

**31.149.3.36 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.

**31.149.3.37 menu\_window()**

```
unsigned int Fl_Window::menu_window () const [inline]
```

Returns true if this window is a menu window.

**31.149.3.38 modal()**

```
unsigned int Fl_Window::modal () const [inline]
```

Returns true if this window is modal.

**31.149.3.39 non\_modal()**

```
unsigned int Fl_Window::non_modal () const [inline]
```

Returns true if this window is modal or non-modal.

**31.149.3.40 override()**

```
unsigned int Fl_Window::override () const [inline]
```

Returns non zero if FL\_OVERRIDE flag is set, 0 otherwise.

31.149.3.41 `resize()`

```
void Fl_Window::resize (
 int X,
 int Y,
 int W,
 int H) [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 `size(x,y)` and `position(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](#).

Reimplemented in [Fl\\_GI\\_Window](#), [Fl\\_Overlay\\_Window](#), and [Fl\\_Double\\_Window](#).

31.149.3.42 `screen_num()`

```
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 position before [show\(\)](#)'ing the window. This can be necessary when a system has several screens with distinct scaling factor values because the window's [x\(\)](#) and [y\(\)](#) may not suffice to uniquely identify one screen. To see that, consider a system with two screens where the screen at left is A pixel-wide and has a scale factor of 1 whereas the screen at right has a scale factor of 2. For the sake of simplicity, consider only the X coordinates of windows. 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. Both window coordinates and screen number are necessary to uniquely identify where a window is to be mapped.

31.149.3.43 `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.

### 31.149.3.44 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\(\)](#).

### 31.149.3.45 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.

### 31.149.3.46 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.

## 31.149.3.47 shape() [1/2]

```
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\\_GI\\_Window](#) instances.
- On the Windows platform, this function does nothing with class [Fl\\_GI\\_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\\_GI\\_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

31.149.3.48 [shape\(\)](#) [2/2]

```
void Fl_Window::shape (
 const Fl_Image & img)
```

Set the window's shape with an [Fl\\_Image](#).

**See also**

[void shape\(const Fl\\_Image\\* img\)](#)

31.149.3.49 [show\(\)](#) [1/2]

```
void Fl_Window::show () [virtual]
```

Puts the window on the screen.

Usually (on X) this has the side effect of opening the display.

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](#).

Reimplemented in [Fl\\_Gl\\_Window](#), [Fl\\_Overlay\\_Window](#), [Fl\\_Double\\_Window](#), [Fl\\_Single\\_Window](#), and [Fl\\_Menu\\_Window](#).

31.149.3.50 [show\(\)](#) [2/2]

```
void Fl_Window::show (
 int argc,
 char ** argv)
```

Puts the window on the screen and parses command-line arguments.

Usually (on X) this has the side effect of opening the display.

This form should be used for top-level windows, at least for the first (main) window. It allows standard arguments to be parsed from the command-line. You can use `argc` and `argv` from `main(int argc, char **argv)` for this call.

The first call also sets up some system-specific internal variables like the system colors.

**Todo** explain which system parameters are set up.

### Parameters

|             |                                                   |
|-------------|---------------------------------------------------|
| <i>argc</i> | command-line argument count, usually from main()  |
| <i>argv</i> | command-line argument vector, usually from main() |

### See also

[virtual void Fl\\_Window::show\(\)](#)

#### 31.149.3.51 shown()

```
int Fl_Window::shown () [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\(\)](#)).

#### 31.149.3.52 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 the user can resize this window to.

This only works for top-level windows.

If this function is not called, FLTK tries to figure out the range from the setting of [resizable\(\)](#):

- 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.
- If either dimension of [resizable\(\)](#) is less than 100, then that is considered the minimum size. Otherwise the [resizable\(\)](#) has a minimum size of 100.
- If either dimension of [resizable\(\)](#) is zero, then that is also the maximum size (so the window cannot resize in that direction).

It is undefined what happens if the current size does not fit in the constraints passed to [size\\_range\(\)](#).

### Parameters

|                                                  |                           |                                                                                                                                                                                                                           |
|--------------------------------------------------|---------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>in</i>                                        | <i>minWidth,minHeight</i> | The smallest the window can be. Either value must be greater than 0.                                                                                                                                                      |
| <i>in</i>                                        | <i>maxWidth,maxHeight</i> | 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.  |
| <small>Generated by Doxygen</small><br><i>in</i> | <i>deltaX,deltaY</i>      | These are size increments. The window will be constrained to widths of <i>minWidth + N * deltaX</i> , where N is any non-negative integer. If these are less or equal to 1 they are ignored. (this is ignored on Windows) |
| <i>in</i>                                        | <i>aspectRatio</i>        | A flag that indicates that the window should preserve its aspect ratio. This only                                                                                                                                         |

### 31.149.3.53 tooltip\_window()

```
unsigned int Fl_Window::tooltip_window () const [inline]
```

Returns true if this window is a tooltip window.

### 31.149.3.54 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\(\)](#).

**31.149.3.55 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\(const char \\*\)](#)

**31.149.3.56 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 Microsoft Windows*, this string is used as the name of the WNDCLASS structure, though it is not clear if this can have any visible effect.

**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\(const char \\*\)](#)

**31.149.4 Member Data Documentation**

### 31.149.4.1 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:

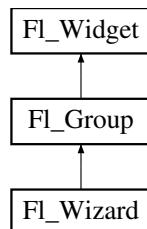
- [Fl\\_Window.H](#)
- [Fl\\_arg.cxx](#)
- [fl\\_cursor.cxx](#)
- [Fl\\_Window.cxx](#)
- [Fl\\_Window\\_fullscreen.cxx](#)
- [Fl\\_Window\\_hotspot.cxx](#)
- [Fl\\_Window\\_iconize.cxx](#)

## 31.150 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.*

## Protected Member Functions

- virtual void [draw \(\)](#)  
*Draws the wizard border and visible child.*

## Additional Inherited Members

### 31.150.1 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.

### 31.150.2 Constructor & Destructor Documentation

#### 31.150.2.1 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.

### 31.150.3 Member Function Documentation

#### 31.150.3.1 draw()

```
void Fl_Wizard::draw (
 void) [protected], [virtual]
```

Draws the wizard border and visible child.

Reimplemented from [Fl\\_Group](#).

### 31.150.3.2 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.

### 31.150.3.3 prev()

```
void Fl_Wizard::prev ()
```

Shows the previous child.

### 31.150.3.4 value() [1/2]

```
Fl_Widget * Fl_Wizard::value ()
```

Gets the current visible child widget.

### 31.150.3.5 value() [2/2]

```
void Fl_Wizard::value (
 Fl_Widget * kid)
```

Sets the child widget that is visible.

The documentation for this class was generated from the following files:

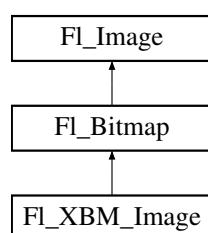
- Fl\_Wizard.H
- Fl\_Wizard.cxx

## 31.151 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.*

## Additional Inherited Members

### 31.151.1 Detailed Description

The [Fl\\_XBM\\_Image](#) class supports loading, caching, and drawing of X Bitmap (XBM) bitmap files.

### 31.151.2 Constructor & Destructor Documentation

#### 31.151.2.1 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

## 31.152 Fl\_XColor Struct Reference

## Public Attributes

- unsigned char **b**
- unsigned char **g**
- unsigned char **mapped**
- unsigned long **pixel**
- unsigned char **r**

The documentation for this struct was generated from the following file:

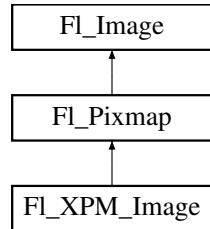
- Fl\_XColor.H

## 31.153 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.*

### Additional Inherited Members

#### 31.153.1 Detailed Description

The [Fl\\_XPM\\_Image](#) class supports loading, caching, and drawing of X Pixmap (XPM) images, including transparency.

#### 31.153.2 Constructor & Destructor Documentation

##### 31.153.2.1 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

## 31.154 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*

### 31.154.1 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

## 31.155 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](#) (unsigned int n)  
*Creates a group name or entry name on the fly.*
- [Name](#) (const char \*format,...)  
*Creates a group name or entry name on the fly.*
- [operator const char \\*](#) ()  
*Return the [Name](#) as a "C" string.*

### 31.155.1 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.

### 31.155.2 Constructor & Destructor Documentation

#### 31.155.2.1 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], "");
```

#### 31.155.2.2 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`

## 31.156 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** ()
- **Entry** & **entry** (int i)
- **Node** \* **find** (const char \*path)
- **RootNode** \* **findRoot** ()
- const char \* **get** (const char \*name)
- int **getEntry** (const char \*name)
- const char \* **name** ()
- int **nChildren** ()
- int **nEntry** ()
- **Node** (const char \*path)
- **Node** \* **parent** ()
- const char \* **path** ()
- char **remove** ()
- **Node** \* **search** (const char \*path, int offset=0)
- void **set** (const char \*name, const char \*value)
- void **set** (const char \*line)
- 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

## 31.157 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*

### 31.157.1 Detailed Description

width, height and name of a page format

The documentation for this struct was generated from the following file:

- [FI\\_Paged\\_Device.H](#)

## 31.158 FI\_Preferences::RootNode Class Reference

### Public Member Functions

- char **getPath** (char \*[path](#), int pathlen)
- int **read** ()
- **RootNode** ([FI\\_Preferences](#) \*, [Root](#) root, const char \*vendor, const char \*application)
- **RootNode** ([FI\\_Preferences](#) \*, const char \*[path](#), const char \*vendor, const char \*application)
- **RootNode** ([FI\\_Preferences](#) \*)
- int **write** ()

The documentation for this class was generated from the following files:

- [FI\\_Preferences.H](#)
- [FI\\_Preferences.cxx](#)

## 31.159 FI\_Scroll::ScrollInfo Struct Reference

Structure to manage scrollbar and widget interior sizes.

```
#include <F1_Scroll.H>
```

### Classes

- struct [FI\\_Region\\_LRTB](#)  
*A local struct to manage a region defined by left/right/top/bottom.*
- struct [FI\\_Region\\_XYWH](#)  
*A local struct to manage a region defined by xywh.*
- struct [FI\\_Scrollbar\\_Data](#)  
*A local struct to manage a scrollbar's xywh region and tab values.*

## Public Attributes

- [Fl\\_Region\\_LRTB child](#)  
*child bounding box: left/right/top/bottom*
- int [hneeded](#)  
*horizontal scrollbar visibility*
- [Fl\\_Scrollbar\\_Data hscroll](#)  
*horizontal scrollbar region + values*
- [Fl\\_Region\\_XYWH innerbox](#)  
*widget's inner box, excluding scrollbars*
- [Fl\\_Region\\_XYWH innerchild](#)  
*widget's inner box, including scrollbars*
- int [scrollsize](#)  
*the effective scrollbar thickness (local or global)*
- int [vneeded](#)  
*vertical scrollbar visibility*
- [Fl\\_Scrollbar\\_Data vscroll](#)  
*vertical scrollbar region + values*

### 31.159.1 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](#)

## 31.160 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](#)  
*currently unused (this may be changed in the future)*
- [Fl\\_Color color](#)  
*text color*
- [Fl\\_Font font](#)  
*text font*
- [Fl\\_Fontsize size](#)  
*text font size*

### 31.160.1 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 [FI\\_Text\\_Display](#) for displaying text. The style table is an array of struct [Style\\_Table\\_Entry](#).

The style table is associated with an [FI\\_Text\\_Display](#) by using [FI\\_Text\\_Display::highlight\\_data\(\)](#).

#### See also

[FI\\_Text\\_Display::highlight\\_data\(\)](#)

The documentation for this struct was generated from the following file:

- [FI\\_Text\\_Display.H](#)

# Chapter 32

## File Documentation

### 32.1 Enumerations.H File Reference

This file contains type definitions and general enumerations.

```
#include <FL/abi-version.h>
#include "Fl_Export.H"
#include "fl_types.h"
#include <FL/platform_types.h>
```

#### Macros

- #define **FL\_IMAGE\_WITH\_ALPHA** 0x40000000

#### Version Numbers

FLTK defines some constants to help the programmer to find out, for which FLTK version a program is compiled.

The following constants are defined:

- #define **FL\_MAJOR\_VERSION** 1  
*The major release version of this FLTK library.*
- #define **FL\_MINOR\_VERSION** 4  
*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.*
- #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\_ABI\_VERSION** **FL\_API\_VERSION**  
*The FLTK ABI (Application Binary Interface) version number as an int.*

#### Mouse and Keyboard Events

This and the following constants define the non-ASCII keys on the keyboard for **FL\_KEYBOARD** and **FL\_SHORTCUT** events.

**See also**

[Fl::event\\_key\(\)](#) and [Fl::get\\_key\(int\)](#) (use ASCII letters for all other keys):

**Todo** *FL\_Button and FL\_key... constants could be structured better (use an enum or some doxygen grouping?)*

- #define [FL\\_Button](#) 0xfee8  
*A mouse button; use [FL\\_Button](#) + n for mouse button n.*
- #define [FL\\_Backspace](#) 0xff08  
*The backspace key.*
- #define [FL\\_Tab](#) 0xff09  
*The tab key.*
- #define [FL\\_Iso\\_Key](#) 0xff0c  
*The additional key of ISO keyboards.*
- #define [FL\\_Enter](#) 0xff0d  
*The enter key.*
- #define [FL\\_Pause](#) 0xff13  
*The pause key.*
- #define [FL\\_Scroll\\_Lock](#) 0xff14  
*The scroll lock key.*
- #define [FL\\_Escape](#) 0xff1b  
*The escape key.*
- #define [FL\\_Kana](#) 0xff2e  
*The Kana key of JIS keyboards.*
- #define [FL\\_Eisu](#) 0xff2f  
*The Eisu key of JIS keyboards.*
- #define [FL\\_Yen](#) 0xff30  
*The Yen key of JIS keyboards.*
- #define [FL\\_JIS\\_Underscore](#) 0xff31  
*The underscore key of JIS keyboards.*
- #define [FL\\_Home](#) 0xff50  
*The home key.*
- #define [FL\\_Left](#) 0xff51  
*The left arrow key.*
- #define [FL\\_Up](#) 0xff52  
*The up arrow key.*
- #define [FL\\_Right](#) 0xff53  
*The right arrow key.*
- #define [FL\\_Down](#) 0xff54  
*The down arrow key.*
- #define [FL\\_Page\\_Up](#) 0xff55  
*The page-up key.*
- #define [FL\\_Page\\_Down](#) 0xff56  
*The page-down key.*
- #define [FL\\_End](#) 0xff57  
*The end key.*
- #define [FL\\_Print](#) 0xff61  
*The print (or print-screen) key.*
- #define [FL\\_Insert](#) 0xff63  
*The insert key.*
- #define [FL\\_Menu](#) 0xff67  
*The menu key.*
- #define [FL\\_Help](#) 0xff68  
*The 'help' key on Mac keyboards.*
- #define [FL\\_Num\\_Lock](#) 0xff7f  
*The num lock key.*
- #define [FL\\_KP](#) 0xff80  
*One of the keypad numbers; use [FL\\_KP](#) + 'n' for digit n.*
- #define [FL\\_KP\\_Enter](#) 0xff8d

- `#define FL_KP_Last 0xffffb`  
*The enter key on the keypad, same as `FL_KP+'r'`.*
- `#define FL_F 0xffffbd`  
*The last keypad key; use to range-check keypad.*
- `#define FL_F_Last 0xffffe0`  
*The last function key; use to range-check function keys.*
- `#define FL_Shift_L 0xffffe1`  
*The lefthand shift key.*
- `#define FL_Shift_R 0xffffe2`  
*The righthand shift key.*
- `#define FL_Control_L 0xffffe3`  
*The lefthand control key.*
- `#define FL_Control_R 0xffffe4`  
*The righthand control key.*
- `#define FL_Caps_Lock 0xffffe5`  
*The caps lock key.*
- `#define FL_Meta_L 0xffffe7`  
*The left meta/Windows key.*
- `#define FL_Meta_R 0xffffe8`  
*The right meta/Windows key.*
- `#define FL_Alt_L 0xffffe9`  
*The left alt key.*
- `#define FL_Alt_R 0xffffea`  
*The right alt key.*
- `#define FL_Delete 0xffffff`  
*The delete 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_Media_Play 0xEF14 /* Start playing of audio */`
- `#define FL_Media_Stop 0xEF15 /* Stop playing audio */`
- `#define FL_Media_Prev 0xEF16 /* Previous track */`
- `#define FL_Media_Next 0xEF17 /* Next track */`
- `#define FL_Home_Page 0xEF18 /* Display user's home page */`
- `#define FL_Mail 0xEF19 /* Invoke user's mail program */`
- `#define FL_Search 0xEF1B /* Search */`
- `#define FL_Back 0xEF26 /* Like back on a browser */`
- `#define FL_Forward 0xEF27 /* Like forward on a browser */`
- `#define FL_Stop 0xEF28 /* Stop current operation */`
- `#define FL_Refresh 0xEF29 /* Refresh the page */`
- `#define FL_Sleep 0xEF2F /* Put system to sleep */`
- `#define FL_Favorites 0xEF30 /* Show favorite locations */`

## Mouse Buttons

These constants define the button numbers for `FL_PUSH` and `FL_RELEASE` events.

See also

[Fl::event\\_button\(\)](#)

- `#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.*

## Event States

The following constants define bits in the `Fl::event_state()` value.

- `#define FL_SHIFT 0x00010000`  
*One of the shift keys is down.*
- `#define FL_CAPS_LOCK 0x00020000`  
*The caps lock is on.*
- `#define FL_CTRL 0x00040000`  
*One of the ctrl keys is down.*
- `#define FL_ALT 0x00080000`  
*One of the alt keys is down.*
- `#define FL_NUM_LOCK 0x00100000`  
*The num lock is on.*
- `#define FL_META 0x00400000`  
*One of the meta/Windows keys is down.*
- `#define FL_SCROLL_LOCK 0x00800000`  
*The scroll lock is on.*
- `#define FL_BUTTON1 0x01000000`  
*Mouse button 1 is pushed.*
- `#define FL_BUTTON2 0x02000000`  
*Mouse button 2 is pushed.*
- `#define FL_BUTTON3 0x04000000`  
*Mouse button 3 is pushed.*
- `#define FL_BUTTONS 0x7f000000`  
*Any mouse button is pushed.*
- `#define FL_BUTTON(n) (0x00800000<<(n))`  
*Mouse button n (n > 0) is pushed.*
- `#define FL_KEY_MASK 0x0000ffff`  
*All keys are 16 bit for now.*

## 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_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 }`

*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_EMBOSSDED_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 }

*visual types and FL\_Gl\_Window::mode() (values match Glut)*

## 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 }

*These constants determine when 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_EXPORT FL_Fontsize FL_NORMAL_SIZE`  
*normal font size*

## Box Types

FLTK standard box types

This enum defines the standard box types included with FLTK.

**Note**

The documented enum `Fl_Boxtype` contains some values (names) with leading underscores, e.g. `_FL_SHADOW_BOX`. 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!

`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 Figure 1 is the area that is not drawn by the frame types.



Figure 32.1 FLTK standard box types

**Todo** Description of boxtypes is incomplete. See below for the defined enum `Fl_Boxtype`.

**See also**

[src/Fl\\_get\\_system\\_colors.cxx](#)

- `#define FL_ROUND_UP_BOX fl_define_FL_ROUND_UP_BOX()`
- `#define FL_ROUND_DOWN_BOX (Fl_Boxtype)(fl_define_FL_ROUND_UP_BOX()+1)`
- `#define FL_SHADOW_BOX fl_define_FL_SHADOW_BOX()`
- `#define FL_SHADOW_FRAME (Fl_Boxtype)(fl_define_FL_SHADOW_BOX()+2)`
- `#define FL_ROUNDED_BOX fl_define_FL_ROUNDED_BOX()`
- `#define FL_ROUNDED_FRAME (Fl_Boxtype)(fl_define_FL_ROUNDED_BOX()+2)`
- `#define FL_RFLAT_BOX fl_define_FL_RFLAT_BOX()`
- `#define FL_RSHADOW_BOX fl_define_FL_RSHADOW_BOX()`
- `#define FL_DIAMOND_UP_BOX fl_define_FL_DIAMOND_BOX()`
- `#define FL_DIAMOND_DOWN_BOX (Fl_Boxtype)(fl_define_FL_DIAMOND_BOX()+1)`
- `#define FL_OVAL_BOX fl_define_FL_OVAL_BOX()`
- `#define FL_OSHADOW_BOX (Fl_Boxtype)(fl_define_FL_OVAL_BOX()+1)`
- `#define FL_OVAL_FRAME (Fl_Boxtype)(fl_define_FL_OVAL_BOX()+2)`
- `#define FL_OFLAT_BOX (Fl_Boxtype)(fl_define_FL_OVAL_BOX()+3)`

- #define **FL\_PLASTIC\_UP\_BOX** fl\_define\_FL\_PLASTIC\_UP\_BOX()
  - #define **FL\_PLASTIC\_DOWN\_BOX** (FL\_Boxtype)(fl\_define\_FL\_PLASTIC\_UP\_BOX()+1)
  - #define **FL\_PLASTIC\_UP\_FRAME** (FL\_Boxtype)(fl\_define\_FL\_PLASTIC\_UP\_BOX()+2)
  - #define **FL\_PLASTIC\_DOWN\_FRAME** (FL\_Boxtype)(fl\_define\_FL\_PLASTIC\_UP\_BOX()+3)
  - #define **FL\_PLASTIC\_THIN\_UP\_BOX** (FL\_Boxtype)(fl\_define\_FL\_PLASTIC\_UP\_BOX()+4)
  - #define **FL\_PLASTIC\_THIN\_DOWN\_BOX** (FL\_Boxtype)(fl\_define\_FL\_PLASTIC\_UP\_BOX()+5)
  - #define **FL\_PLASTIC\_ROUND\_UP\_BOX** (FL\_Boxtype)(fl\_define\_FL\_PLASTIC\_UP\_BOX()+6)
  - #define **FL\_PLASTIC\_ROUND\_DOWN\_BOX** (FL\_Boxtype)(fl\_define\_FL\_PLASTIC\_UP\_BOX()+7)
  - #define **FL\_GTK\_UP\_BOX** fl\_define\_FL\_GTK\_UP\_BOX()
  - #define **FL\_GTK\_DOWN\_BOX** (FL\_Boxtype)(fl\_define\_FL\_GTK\_UP\_BOX()+1)
  - #define **FL\_GTK\_UP\_FRAME** (FL\_Boxtype)(fl\_define\_FL\_GTK\_UP\_BOX()+2)
  - #define **FL\_GTK\_DOWN\_FRAME** (FL\_Boxtype)(fl\_define\_FL\_GTK\_UP\_BOX()+3)
  - #define **FL\_GTK\_THIN\_UP\_BOX** (FL\_Boxtype)(fl\_define\_FL\_GTK\_UP\_BOX()+4)
  - #define **FL\_GTK\_THIN\_DOWN\_BOX** (FL\_Boxtype)(fl\_define\_FL\_GTK\_UP\_BOX()+5)
  - #define **FL\_GTK\_THIN\_UP\_FRAME** (FL\_Boxtype)(fl\_define\_FL\_GTK\_UP\_BOX()+6)
  - #define **FL\_GTK\_THIN\_DOWN\_FRAME** (FL\_Boxtype)(fl\_define\_FL\_GTK\_UP\_BOX()+7)
  - #define **FL\_GTK\_ROUND\_UP\_BOX** (FL\_Boxtype)(fl\_define\_FL\_GTK\_UP\_BOX()+8)
  - #define **FL\_GTK\_ROUND\_DOWN\_BOX** (FL\_Boxtype)(fl\_define\_FL\_GTK\_UP\_BOX()+9)
  - #define **FL\_GLEAM\_UP\_BOX** fl\_define\_FL\_GLEAM\_UP\_BOX()
  - #define **FL\_GLEAM\_DOWN\_BOX** (FL\_Boxtype)(fl\_define\_FL\_GLEAM\_UP\_BOX()+1)
  - #define **FL\_GLEAM\_UP\_FRAME** (FL\_Boxtype)(fl\_define\_FL\_GLEAM\_UP\_BOX()+2)
  - #define **FL\_GLEAM\_DOWN\_FRAME** (FL\_Boxtype)(fl\_define\_FL\_GLEAM\_UP\_BOX()+3)
  - #define **FL\_GLEAM\_THIN\_UP\_BOX** (FL\_Boxtype)(fl\_define\_FL\_GLEAM\_UP\_BOX()+4)
  - #define **FL\_GLEAM\_THIN\_DOWN\_BOX** (FL\_Boxtype)(fl\_define\_FL\_GLEAM\_UP\_BOX()+5)
  - #define **FL\_GLEAM\_ROUND\_UP\_BOX** (FL\_Boxtype)(fl\_define\_FL\_GLEAM\_UP\_BOX()+6)
  - #define **FL\_GLEAM\_ROUND\_DOWN\_BOX** (FL\_Boxtype)(fl\_define\_FL\_GLEAM\_UP\_BOX()+7)
  - #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
  - 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\_EMBOSSSED\_BOX,
 FL\_ENGRAVED\_FRAME, FL\_EMBOSSSED\_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\_FREE\_BOXTYPE }
- FLTK standard box types.
- FL\_EXPORT FL\_Boxtype fl\_define\_FL\_ROUND\_UP\_BOX ()
  - FL\_EXPORT FL\_Boxtype fl\_define\_FL\_SHADOW\_BOX ()
  - FL\_EXPORT FL\_Boxtype fl\_define\_FL\_ROUNDED\_BOX ()

- FL\_EXPORT `FL_Boxtype fl_define_FL_RFLAT_BOX ()`
- FL\_EXPORT `FL_Boxtype fl_define_FL_RSHADOW_BOX ()`
- FL\_EXPORT `FL_Boxtype fl_define_FL_DIAMOND_BOX ()`
- FL\_EXPORT `FL_Boxtype fl_define_FL_OVAL_BOX ()`
- FL\_EXPORT `FL_Boxtype fl_define_FL_PLASTIC_UP_BOX ()`
- FL\_EXPORT `FL_Boxtype fl_define_FL_GTK_UP_BOX ()`
- FL\_EXPORT `FL_Boxtype fl_define_FL_GLEAM_UP_BOX ()`
- `FL_Boxtype fl_box (FL_Boxtype b)`  
*Get the filled version of a frame.*
- `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_SYMBOL_LABEL FL_NORMAL_LABEL`  
*Sets the current label type and returns its corresponding FL\_Labeltype value.*
- `#define FL_SHADOW_LABEL fl_define_FL_SHADOW_LABEL()`  
*Draws a label with shadows behind the text.*
- `#define FL_ENGRAVED_LABEL fl_define_FL_ENGRAVED_LABEL()`  
*Draws a label with engraved text.*
- `#define FL_EMBOSSDED_LABEL fl_define_FL_EMBOSSDED_LABEL()`  
*Draws a label with embossed text.*
- `#define FL_MULTI_LABEL fl_define_FL_MULTI_LABEL()`  
*Draws a label that can comprise several parts like text and images.*
- `#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.*
- `FL_Labeltype FL_EXPORT fl_define_FL_SHADOW_LABEL ()`  
*Initializes the internal table entry for FL\_SHADOW\_LABEL and returns its internal value.*
- `FL_Labeltype FL_EXPORT fl_define_FL_ENGRAVED_LABEL ()`  
*Initializes the internal table entry for FL\_ENGRAVED\_LABEL and returns its internal value.*
- `FL_Labeltype FL_EXPORT fl_define_FL_EMBOSSDED_LABEL ()`  
*Initializes the internal table entry for FL\_EMBOSSDED\_LABEL and returns its internal value.*
- `FL_Labeltype FL_EXPORT fl_define_FL_MULTI_LABEL ()`  
*Initializes the internal table entry for FL\_MULTI\_LABEL and returns its internal value.*
- `FL_Labeltype FL_EXPORT fl_define_FL_ICON_LABEL ()`  
*Initializes the internal table entry for FL\_ICON\_LABEL and returns its internal value.*
- `FL_Labeltype FL_EXPORT fl_define_FL_IMAGE_LABEL ()`  
*Initializes the internal table entry for FL\_IMAGE\_LABEL and returns its internal value.*

## Alignment Flags

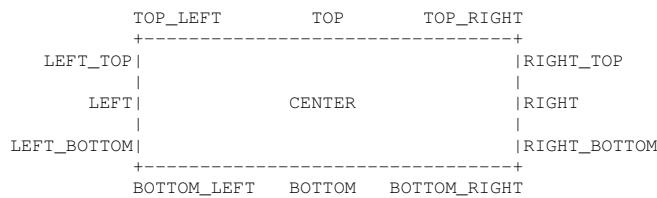
Flags to control the label alignment.

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.

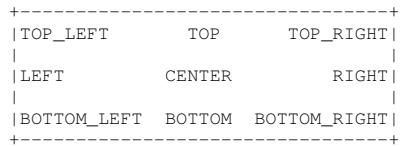
All alignment flags use the common prefix "FL\_ALIGN\_". In the following descriptions this prefix is sometimes omitted for brevity.

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):



Inside alignments (`FL_ALIGN_INSIDE` is set):



## 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\_CENTER = 0x0000**  
*Align the label horizontally in the middle.*
- **const Fl\_Align FL\_ALIGN\_TOP = 0x0001**  
*Align the label at the top of the widget.*
- **const Fl\_Align FL\_ALIGN\_BOTTOM = 0x0002**  
*Align the label at the bottom of the widget.*
- **const Fl\_Align FL\_ALIGN\_LEFT = 0x0004**  
*Align the label at the left of the widget.*
- **const Fl\_Align FL\_ALIGN\_RIGHT = 0x0008**  
*Align the label to the right of the widget.*
- **const Fl\_Align FL\_ALIGN\_INSIDE = 0x0010**  
*Draw the label inside of the widget.*
- **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\_IMAGE\_OVER\_TEXT = 0x0000**

- const `FL_Align FL_ALIGN_CLIP = 0x0040`

*If the label contains an image, draw the text below the image.*
- const `FL_Align FL_ALIGN_WRAP = 0x0080`

*All parts of the label that are larger than the widget will not be drawn.*
- const `FL_Align FL_ALIGN_IMAGE_NEXT_TO_TEXT = 0x0100`

*Wrap text that does not fit the width of the widget.*
- const `FL_Align FL_ALIGN_TEXT_NEXT_TO_IMAGE = 0x0120`

*If the label contains an image, draw the text to the right of the image.*
- 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_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`

*Outside only, left of widget, top position, magic value: TOP | BOTTOM | LEFT.*
- 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_LEFT_BOTTOM = 0x000d`

*Outside only, left of widget, bottom position, magic value: TOP | LEFT | RIGHT.*
- 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_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_IMAGE_MASK = 0x0320`

*Mask value to test for image alignment flags.*

## Font Numbers

The following constants define the standard FLTK fonts:

- `typedef int FL_Font`

*A font number is an index into the internal font table.*
- 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_ITALIC = 2`

*Helvetica (or Arial) oblique.*
- const `FL_Font FL_HELVETICA_BOLD_ITALIC = 3`

*Helvetica (or Arial) bold-oblique.*
- const `FL_Font FL_COURIER = 4`

*Courier normal.*
- const `FL_Font FL_COURIER_BOLD = 5`

*Courier bold.*
- const `FL_Font FL_COURIER_ITALIC = 6`

- const `FL_Font FL_COURIER_BOLD_ITALIC` = 7  
*Courier italic.*
- const `FL_Font FL_TIMES` = 8  
*Courier bold-italic.*
- const `FL_Font FL_TIMES_BOLD` = 9  
*Times roman.*
- const `FL_Font FL_TIMES_ITALIC` = 10  
*Times roman bold.*
- const `FL_Font FL_TIMES_BOLD_ITALIC` = 11  
*Times roman bold-italic.*
- const `FL_Font FL_SYMBOL` = 12  
*Standard symbol font.*
- 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_ZAPF_DINGBATS` = 15  
*Zapf-dingbats font.*
- const `FL_Font FL_FREE_FONT` = 16  
*first one to allocate*
- const `FL_Font FL_BOLD` = 1  
*add this to helvetica, courier, or times*
- const `FL_Font FL_ITALIC` = 2  
*add this to helvetica, courier, or times*
- const `FL_Font FL_BOLD_ITALIC` = 3  
*add this to helvetica, courier, or times*

## 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 => 0xrrggbbi
 | | | |
 | | | +-- 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.

- #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`
- `typedef unsigned int FL_Color`  
*An FLTK color value; see also [Colors](#).*
- `const FL_Color FL_FOREGROUND_COLOR = 0`  
*the default foreground color (0) used for labels and text*
- `const FL_Color FL_BACKGROUND2_COLOR = 7`  
*the default background color for text, list, and valuator widgets*
- `const FL_Color FL_INACTIVE_COLOR = 8`  
*the inactive foreground color*
- `const FL_Color FL_SELECTION_COLOR = 15`  
*the default selection/highlight color*
- `const FL_Color FL_GRAY0 = 32`
- `const FL_Color FL_DARK3 = 39`
- `const FL_Color FL_DARK2 = 45`
- `const FL_Color FL_DARK1 = 47`
- `const FL_Color FL_BACKGROUND_COLOR = 49`
- `const FL_Color FL_LIGHT1 = 50`
- `const FL_Color FL_LIGHT2 = 52`
- `const FL_Color FL_LIGHT3 = 54`
- `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`
- `FL_EXPORT FL_Color fl_inactive (FL_Color c)`  
*Returns the inactive, dimmed version of the given color.*
- `FL_EXPORT FL_Color fl_contrast (FL_Color fg, FL_Color bg)`  
*Returns a color that contrasts with the background color.*
- `FL_EXPORT FL_Color fl_color_average (FL_Color c1, FL_Color c2, float weight)`  
*Returns the weighted average color between the two given colors.*
- `FL_Color fl_lighter (FL_Color c)`  
*Returns a lighter version of the specified color.*
- `FL_Color fl_darker (FL_Color c)`  
*Returns a darker version of the specified color.*
- `FL_Color fl_rgb_color (uchar r, uchar g, uchar b)`  
*Returns the 24-bit color value closest to r, g, b.*
- `FL_Color fl_rgb_color (uchar g)`  
*Returns the 24-bit color value closest to g (grayscale).*
- `FL_Color fl_gray_ramp (int i)`  
*Returns a gray color value from black (i == 0) to white (i == FL\_NUM\_GRAY - 1).*
- `FL_Color fl_color_cube (int r, int g, int b)`  
*Returns a color out of the color cube.*

### 32.1.1 Detailed Description

This file contains type definitions and general enumerations.

### 32.1.2 Macro Definition Documentation

#### 32.1.2.1 FL\_ABI\_VERSION

```
#define FL_ABI_VERSION FL_API_VERSION
```

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.

#### See also

[README.abi-version.txt](#)

#### 32.1.2.2 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\(\)](#)

### 32.1.2.3 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.

### 32.1.2.4 FL\_MAJOR\_VERSION

```
#define FL_MAJOR_VERSION 1
```

The major release version of this FLTK library.

See also

[FL\\_VERSION](#)

### 32.1.2.5 FL\_MINOR\_VERSION

```
#define FL_MINOR_VERSION 4
```

The minor release version for this library.

FLTK remains mostly source-code compatible between minor version changes.

### 32.1.2.6 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](#)

### 32.1.2.7 FL\_PATCH\_VERSION

```
#define FL_PATCH_VERSION 0
```

The patch version for this library.

FLTK remains binary compatible between patches.

### 32.1.2.8 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.

### 32.1.2.9 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\(\)](#)

### 32.1.3 Typedef Documentation

#### 32.1.3.1 Fl\_Align

```
typedef unsigned Fl_Align
```

FLTK type for alignment control.

### 32.1.3.2 Fl\_Font

```
typedef int Fl_Font
```

A font number is an index into the internal font table.

### 32.1.3.3 Fl\_Fontsize

```
typedef int Fl_Fontsize
```

Size of a font in pixels.

This is the approximate height of a font in pixels.

## 32.1.4 Enumeration Type Documentation

### 32.1.4.1 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.        |

### 32.1.4.2 Fl\_Boxtype

```
enum Fl_Boxtype
```

FLTK standard box types.

This enum defines the standard box types included with FLTK.

#### Note

The documented enum Fl\_Boxtype contains some values (names) with leading underscores, e.g. `_FL_SHADOW_BOX`. 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!

## Enumerator

|                            |                                                                     |
|----------------------------|---------------------------------------------------------------------|
| FL_NO_BOX                  | nothing is drawn at all, this box is invisible                      |
| FL_FLAT_BOX                | a flat box                                                          |
| FL_UP_BOX                  | see figure 1                                                        |
| FL_DOWN_BOX                | see figure 1                                                        |
| FL_UP_FRAME                | see figure 1                                                        |
| FL_DOWN_FRAME              | see figure 1                                                        |
| FL_THIN_UP_BOX             | see figure 1                                                        |
| FL_THIN_DOWN_BOX           | see figure 1                                                        |
| FL_THIN_UP_FRAME           | see figure 1                                                        |
| FL_THIN_DOWN_FRAME         | see figure 1                                                        |
| FL_ENGRAVED_BOX            | see figure 1                                                        |
| FL_EMBOSSSED_BOX           | see figure 1                                                        |
| FL_ENGRAVED_FRAME          | see figure 1                                                        |
| FL_EMBOSSSED_FRAME         | see figure 1                                                        |
| FL_BORDER_BOX              | see figure 1                                                        |
| _FL_SHADOW_BOX             | see figure 1, use FL_SHADOW_BOX                                     |
| FL_BORDER_FRAME            | see figure 1                                                        |
| _FL_SHADOW_FRAME           | see figure 1, use FL_SHADOW_FRAME                                   |
| _FL_ROUNDED_BOX            | see figure 1, use FL_ROUNDED_BOX                                    |
| _FL_RSHADOW_BOX            | see figure 1, use FL_RSHADOW_BOX                                    |
| _FL_ROUNDED_FRAME          | see figure 1, use FL_ROUNDED_FRAME                                  |
| _FL_RFLAT_BOX              | see figure 1, use FL_RFLAT_BOX                                      |
| FL_ROUND_UP_BOX            | see figure 1, use FL_ROUND_UP_BOX                                   |
| _FL_ROUND_DOWN_BOX         | see figure 1, use FL_ROUND_DOWN_BOX                                 |
| _FL_DIAMOND_UP_BOX         | see figure 1, use FL_DIAMOND_UP_BOX                                 |
| _FL_DIAMOND_DOWN_BOX       | see figure 1, use FL_DIAMOND_DOWN_BOX                               |
| _FL_OVAL_BOX               | see figure 1, use FL_OVAL_BOX                                       |
| _FL_OSHADOW_BOX            | see figure 1, use FL_OSHADOW_BOX                                    |
| _FL_OVAL_FRAME             | see figure 1, use FL_OVAL_FRAME                                     |
| _FL_OFLAT_BOX              | see figure 1, use FL_OFLAT_BOX                                      |
| _FL_PLASTIC_UP_BOX         | plastic version of FL_UP_BOX, use FL_PLASTIC_UP_BOX                 |
| _FL_PLASTIC_DOWN_BOX       | plastic version of FL_DOWN_BOX, use FL_PLASTIC_DOWN_BOX             |
| _FL_PLASTIC_UP_FRAME       | plastic version of FL_UP_FRAME, use FL_PLASTIC_UP_FRAME             |
| _FL_PLASTIC_DOWN_FRAME     | plastic version of FL_DOWN_FRAME, use FL_PLASTIC_DOWN_FRAME         |
| _FL_PLASTIC_THIN_UP_BOX    | plastic version of FL_THIN_UP_BOX, use FL_PLASTIC_THIN_UP_BOX       |
| _FL_PLASTIC_THIN_DOWN_BOX  | plastic version of FL_THIN_DOWN_BOX, use FL_PLASTIC_THIN_DOWN_BOX   |
| _FL_PLASTIC_ROUND_UP_BOX   | plastic version of FL_ROUND_UP_BOX, use FL_PLASTIC_ROUND_UP_BOX     |
| _FL_PLASTIC_ROUND_DOWN_BOX | plastic version of FL_ROUND_DOWN_BOX, use FL_PLASTIC_ROUND_DOWN_BOX |
| _FL_GTK_UP_BOX             | gtk+ version of FL_UP_BOX, use FL_GTK_UP_BOX                        |
| _FL_GTK_DOWN_BOX           | gtk+ version of FL_DOWN_BOX, use FL_GTK_DOWN_BOX                    |
| _FL_GTK_UP_FRAME           | gtk+ version of FL_UP_FRAME, use FL_GTK_UP_FRAME                    |
| _FL_GTK_DOWN_FRAME         | gtk+ version of FL_DOWN_FRAME, use FL_GTK_DOWN_FRAME                |

## Enumerator

|                          |                                                                 |
|--------------------------|-----------------------------------------------------------------|
| _FL_GTK_THIN_UP_BOX      | gtk+ version of FL_THIN_UP_BOX, use FL_GTK_THIN_UP_BOX          |
| _FL_GTK_THIN_DOWN_BOX    | gtk+ version of FL_THIN_DOWN_BOX, use FL_GTK_THIN_DOWN_BOX      |
| _FL_GTK_THIN_UP_FRAME    | gtk+ version of FL_THIN_UP_FRAME, use FL_GTK_THIN_UP_FRAME      |
| _FL_GTK_THIN_DOWN_FRAME  | gtk+ version of FL_THIN_DOWN_FRAME, use FL_GTK_THIN_DOWN_FRAME  |
| _FL_GTK_ROUND_UP_BOX     | gtk+ version of FL_ROUND_UP_BOX, use FL_GTK_ROUND_UP_BOX        |
| _FL_GTK_ROUND_DOWN_BOX   | gtk+ version of FL_ROUND_DOWN_BOX, use FL_GTK_ROUND_DOWN_BOX    |
| _FL_GLEAM_UP_BOX         | gleam version of FL_UP_BOX, use FL_GLEAM_UP_BOX                 |
| _FL_GLEAM_DOWN_BOX       | gleam version of FL_DOWN_BOX, use FL_GLEAM_DOWN_BOX             |
| _FL_GLEAM_UP_FRAME       | gleam version of FL_UP_FRAME, use FL_GLEAM_UP_FRAME             |
| _FL_GLEAM_DOWN_FRAME     | gleam version of FL_DOWN_FRAME, use FL_GLEAM_DOWN_FRAME         |
| _FL_GLEAM_THIN_UP_BOX    | gleam version of FL_THIN_UP_BOX, use FL_GLEAM_THIN_UP_BOX       |
| _FL_GLEAM_THIN_DOWN_BOX  | gleam version of FL_THIN_DOWN_BOX, use FL_GLEAM_THIN_DOWN_BOX   |
| _FL_GLEAM_ROUND_UP_BOX   | gleam version of FL_ROUND_UP_BOX, use FL_GLEAM_ROUND_UP_BOX     |
| _FL_GLEAM_ROUND_DOWN_BOX | gleam version of FL_ROUND_DOWN_BOX, use FL_GLEAM_ROUND_DOWN_BOX |
| FL_FREE_BOXTYPE          | the first free box type for creation of new box types           |

## 32.1.4.3 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.

**Todo** enum Fl\_Cursor needs maybe an image.

## 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.                 |

## Enumerator

|                |                          |
|----------------|--------------------------|
| FL_CURSOR_NS   | up/down resize.          |
| 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.               |

## 32.1.4.4 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 <a href="#">Fl_Scroll</a> 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.                    |

## 32.1.4.5 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 defined in #include <[F/L/names.h](#)>

## See also

[Fl::event\\_text\(\)](#), [Fl::event\\_key\(\)](#), class [Fl::](#)

## Enumerator

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| FL_NO_EVENT | No event.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| FL_PUSH     | A mouse button has gone down with the mouse pointing at this widget. You can find out what button by calling <a href="#">Fl::event_button()</a> . You find out the mouse position by calling <a href="#">Fl::event_x()</a> and <a href="#">Fl::event_y()</a> .<br>A widget indicates that it "wants" the mouse click by returning non-zero from its <a href="#">Fl_Widget::handle()</a> method. It will then become the <a href="#">Fl::pushed()</a> widget and will get FL_DRAG and the matching FL_RELEASE events. If <a href="#">Fl_Widget::handle()</a> 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 <a href="#">Fl::event_button()</a> .<br>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 <a href="#">Fl::belowmouse()</a> 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 <a href="#">Fl::event_state()</a> . The mouse position is in <a href="#">Fl::event_x()</a> and <a href="#">Fl::event_y()</a> .<br>In order to receive FL_DRAG events, the widget must return non-zero when handling FL_PUSH.                                                                                                                                                                                                                                                                                                                                                                               |
| FL_FOCUS    | This indicates an <i>attempt</i> 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 <a href="#">Fl::focus()</a> 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 <a href="#">Fl::event_key()</a> 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 <a href="#">Fl::focus()</a> widget when another widget gets the focus or the window loses focus.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

## Enumerator

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| FL_KEYDOWN    | <p>A key was pressed (FL_KEYDOWN) or released (FL_KEYUP). FL_KEYBOARD is a synonym for FL_KEYDOWN. The key can be found in <a href="#">Fl::event_key()</a>. The text that the key should insert can be found with <a href="#">Fl::event_text()</a> and its length is in <a href="#">Fl::event_length()</a>. 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.</p> <p>To receive FL_KEYBOARD events you must also respond to the FL_FOCUS and FL_UNFOCUS events.</p> <p>If you are writing a text-editing widget you may also want to call the <a href="#">Fl::compose()</a> function to translate individual keystrokes into non-ASCII characters.</p> <p>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.</p> |
| FL_KEYBOARD   | <p>Equivalent to FL_KEYDOWN.</p> <p><b>See also</b></p> <p><a href="#">FL_KEYDOWN</a></p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| FL_KEYUP      | <p>Key release event.</p> <p><b>See also</b></p> <p><a href="#">FL_KEYDOWN</a></p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| FL_CLOSE      | <p>The user clicked the close button of a window. This event is used internally only to trigger the callback of <a href="#">Fl_Window</a> derived classes. The default callback closes the window calling <a href="#">Fl_Window::hide()</a>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| FL_MOVE       | <p>The mouse has moved without any mouse buttons held down. This event is sent to the <a href="#">Fl::belowmouse()</a> widget. In order to receive FL_MOVE events, the widget must return non-zero when handling FL_ENTER.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| FL_SHORTCUT   | <p>If the <a href="#">Fl::focus()</a> 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 <a href="#">Fl::belowmouse()</a> 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 ignore any keystrokes!</p> <p>You can also make "global" shortcuts by using <a href="#">Fl::add_handler()</a>. A global shortcut will work no matter what windows are displayed or which one has the focus.</p>                                                                                                                                                                                                                                                                                                                                                                                                                    |
| FL_DEACTIVATE | <p>This widget is no longer active, due to <a href="#">Fl_Widget::deactivate()</a> being called on it or one of its parents. <a href="#">Fl_Widget::active()</a> may still be true after this, the widget is only active if <a href="#">Fl_Widget::active()</a> is true on it and all its parents (use <a href="#">Fl_Widget::active_r()</a> to check this).</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| FL_ACTIVATE   | <p>This widget is now active, due to <a href="#">Fl_Widget::activate()</a> being called on it or one of its parents.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

## Enumerator

|                                 |                                                                                                                                                                                                                                                                                                                                                                                               |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| FL_HIDE                         | This widget is no longer visible, due to <a href="#">Fl_Widget::hide()</a> being called on it or one of its parents, or due to a parent window being minimized. <a href="#">Fl_Widget::visible()</a> may still be true after this, but the widget is visible only if <a href="#">visible()</a> is true for it and all its parents (use <a href="#">Fl_Widget::visible_r()</a> to check this). |
| FL_SHOW                         | This widget is visible again, due to <a href="#">Fl_Widget::show()</a> 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 <a href="#">Fl_Widget::handle()</a> method!                           |
| FL_PASTE                        | You should get this event some time after you call <a href="#">Fl::paste()</a> . The contents of <a href="#">Fl::event_text()</a> is the text to insert and the number of characters is in <a href="#">Fl::event_length()</a> .                                                                                                                                                               |
| FL_SELECTIONCLEAR               | The <a href="#">Fl::selection_owner()</a> 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 <a href="#">Fl::event_dx()</a> and <a href="#">Fl::event_dy()</a> 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 <a href="#">Fl::add_handler()</a> 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 <a href="#">Fl::event_dy()</a> method can be used to find magnification amount, <a href="#">Fl::event_x()</a> and <a href="#">Fl::event_y()</a> 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 <a href="#">Fl::add_handler()</a> to be notified of this event.                                                                                                                                                                                                                                                                   |

## 32.1.4.6 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**

|                                 |                                                                               |
|---------------------------------|-------------------------------------------------------------------------------|
| <code>FL_NORMAL_LABEL</code>    | draws the text (0)                                                            |
| <code>FL_NO_LABEL</code>        | does nothing                                                                  |
| <code>_FL_SHADOW_LABEL</code>   | draws a drop shadow under the text                                            |
| <code>_FL_ENGRAVED_LABEL</code> | draws edges as though the text is engraved                                    |
| <code>_FL_EMBOSS_LABEL</code>   | draws edges as though the text is raised                                      |
| <code>_FL_MULTI_LABEL</code>    | draws a composite label<br><br>See also<br><br><a href="#">Fl_Multi_Label</a> |
| <code>_FL_ICON_LABEL</code>     | draws the icon associated with the text                                       |
| <code>_FL_IMAGE_LABEL</code>    | the label displays an "icon" based on a <a href="#">Fl_Image</a>              |
| <code>FL_FREE_LABELTYPE</code>  | first free labeltype to use for creating own labeltypes                       |

**32.1.4.7 Fl\_When**

```
enum Fl_When
```

These constants determine when a callback is performed.

**See also**

[Fl\\_Widget::when\(\)](#);

**Todo** doxygen comments for values are incomplete and maybe wrong or unclear

**Enumerator**

|                                        |                                                                                                        |
|----------------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>FL_WHEN_NEVER</code>             | Never call the callback.                                                                               |
| <code>FL_WHEN_CHANGED</code>           | Do the callback only when the widget value changes.                                                    |
| <code>FL_WHEN_NOT_CHANGED</code>       | Do the callback whenever the user interacts with the widget.                                           |
| <code>FL_WHEN_RELEASE</code>           | Do the callback when the button or key is released and the value changes.                              |
| <code>FL_WHEN_RELEASE_ALWAYS</code>    | Do the callback when the button or key is released, even if the value doesn't change.                  |
| <code>FL_WHEN_ENTER_KEY</code>         | Do the callback when the user presses the ENTER key and the value changes.                             |
| <code>FL_WHEN_ENTER_KEY_ALWAYS</code>  | Do the callback when the user presses the ENTER key, even if the value doesn't change.                 |
| <code>FL_WHEN_ENTER_KEY_CHANGED</code> | = ( <code>FL_WHEN_ENTER_KEY</code>   <code>FL_WHEN_CHANGED</code>   <code>FL_WHEN_NOT_CHANGED</code> ) |

### 32.1.5 Function Documentation

#### 32.1.5.1 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.

#### 32.1.5.2 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);
```

#### 32.1.5.3 fl\_darker()

```
Fl_Color fl_darker (
 Fl_Color c) [inline]
```

Returns a darker version of the specified color.

#### 32.1.5.4 fl\_define\_FL\_EMBOSSSED\_LABEL()

```
Fl_Labeltype FL_EXPORT fl_define_FL_EMBOSSSED_LABEL ()
```

Initializes the internal table entry for FL\_EMBOSSSED\_LABEL and returns its internal value.

Internal use only.

### 32.1.5.5 fl\_define\_FL\_ENGRAVED\_LABEL()

```
Fl_Labeltype FL_EXPORT fl_define_FL_ENGRAVED_LABEL ()
```

Initializes the internal table entry for FL\_ENGRAVED\_LABEL and returns its internal value.

Internal use only.

### 32.1.5.6 fl\_define\_FL\_ICON\_LABEL()

```
Fl_Labeltype FL_EXPORT fl_define_FL_ICON_LABEL ()
```

Initializes the internal table entry for FL\_ICON\_LABEL and returns its internal value.

Internal use only.

### 32.1.5.7 fl\_define\_FL\_IMAGE\_LABEL()

```
Fl_Labeltype FL_EXPORT fl_define_FL_IMAGE_LABEL ()
```

Initializes the internal table entry for FL\_IMAGE\_LABEL and returns its internal value.

Internal use only.

### 32.1.5.8 fl\_define\_FL\_MULTI\_LABEL()

```
Fl_Labeltype FL_EXPORT fl_define_FL_MULTI_LABEL ()
```

Initializes the internal table entry for FL\_MULTI\_LABEL and returns its internal value.

Internal use only.

### 32.1.5.9 fl\_define\_FL\_SHADOW\_LABEL()

```
Fl_Labeltype FL_EXPORT fl_define_FL_SHADOW_LABEL ()
```

Initializes the internal table entry for FL\_SHADOW\_LABEL and returns its internal value.

Internal use only.

### 32.1.5.10 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.

### 32.1.5.11 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.

### 32.1.5.12 fl\_gray\_ramp()

```
Fl_Color fl_gray_ramp (
 int i) [inline]
```

Returns a gray color value from black ( $i == 0$ ) to white ( $i == \text{FL\_NUM\_GRAY} - 1$ ).

`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 * (\text{FL_NUM_GRAY} - 1) / 255)
```

### 32.1.5.13 fl\_lighter()

```
Fl_Color fl_lighter (
 Fl_Color c) [inline]
```

Returns a lighter version of the specified color.

### 32.1.5.14 fl\_rgb\_color() [1/2]

```
Fl_Color fl_rgb_color (
 uchar r,
 uchar g,
 uchar b) [inline]
```

Returns the 24-bit color value closest to `r`, `g`, `b`.

### 32.1.5.15 fl\_rgb\_color() [2/2]

```
Fl_Color fl_rgb_color (
 uchar g) [inline]
```

Returns the 24-bit color value closest to `g` (grayscale).

### 32.1.6 Variable Documentation

#### 32.1.6.1 FL\_ALIGN\_BOTTOM

```
const Fl_Align FL_ALIGN_BOTTOM = 0x0002
```

Align the label at the bottom of the widget.

#### 32.1.6.2 FL\_ALIGN\_CENTER

```
const Fl_Align FL_ALIGN_CENTER = 0x0000
```

Align the label horizontally in the middle.

#### 32.1.6.3 FL\_ALIGN\_CLIP

```
const Fl_Align FL_ALIGN_CLIP = 0x0040
```

All parts of the label that are larger than the widget will not be drawn.

#### 32.1.6.4 FL\_ALIGN\_IMAGE\_BACKDROP

```
const Fl_Align FL_ALIGN_IMAGE_BACKDROP = 0x0200
```

If the label contains an image, draw the image or deimage in the background.

#### 32.1.6.5 FL\_ALIGN\_IMAGE\_MASK

```
const Fl_Align FL_ALIGN_IMAGE_MASK = 0x0320
```

Mask value to test for image alignment flags.

### 32.1.6.6 FL\_ALIGN\_IMAGE\_NEXT\_TO\_TEXT

```
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.

### 32.1.6.7 FL\_ALIGN\_IMAGE\_OVER\_TEXT

```
const Fl_Align FL_ALIGN_IMAGE_OVER_TEXT = 0x0000
```

If the label contains an image, draw the text below the image.

### 32.1.6.8 FL\_ALIGN\_INSIDE

```
const Fl_Align FL_ALIGN_INSIDE = 0x0010
```

Draw the label inside of the widget.

### 32.1.6.9 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.

### 32.1.6.10 FL\_ALIGN\_LEFT\_BOTTOM

```
const Fl_Align FL_ALIGN_LEFT_BOTTOM = 0x000d
```

Outside only, left of widget, bottom position, magic value: TOP | LEFT | RIGHT.

### 32.1.6.11 FL\_ALIGN\_LEFT\_TOP

```
const Fl_Align FL_ALIGN_LEFT_TOP = 0x0007
```

Outside only, left of widget, top position, magic value: TOP | BOTTOM | LEFT.

### 32.1.6.12 FL\_ALIGN\_NOWRAP

```
const F1_Align FL_ALIGN_NOWRAP = 0x0000
```

Nothing, same as FL\_ALIGN\_CENTER, for back compatibility.

### 32.1.6.13 FL\_ALIGN\_POSITION\_MASK

```
const F1_Align FL_ALIGN_POSITION_MASK = 0x000f
```

Mask value to test for TOP, BOTTOM, LEFT, and RIGHT flags.

### 32.1.6.14 FL\_ALIGN\_RIGHT

```
const F1_Align FL_ALIGN_RIGHT = 0x0008
```

Align the label to the right of the widget.

### 32.1.6.15 FL\_ALIGN\_RIGHT\_BOTTOM

```
const F1_Align FL_ALIGN_RIGHT_BOTTOM = 0x000e
```

Outside only, right of widget, bottom position, magic value: BOTTOM | LEFT | RIGHT.

### 32.1.6.16 FL\_ALIGN\_RIGHT\_TOP

```
const F1_Align FL_ALIGN_RIGHT_TOP = 0x000b
```

Outside only, right of widget, top position, magic value: TOP | BOTTOM | RIGHT.

### 32.1.6.17 FL\_ALIGN\_TEXT\_NEXT\_TO\_IMAGE

```
const F1_Align FL_ALIGN_TEXT_NEXT_TO_IMAGE = 0x0120
```

If the label contains an image, draw the text to the left of the image.

### 32.1.6.18 FL\_ALIGN\_TEXT\_OVER\_IMAGE

```
const Fl_Align FL_ALIGN_TEXT_OVER_IMAGE = 0x0020
```

If the label contains an image, draw the text on top of the image.

### 32.1.6.19 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.

### 32.1.6.20 FL\_ALIGN\_WRAP

```
const Fl_Align FL_ALIGN_WRAP = 0x0080
```

Wrap text that does not fit the width of the widget.

### 32.1.6.21 FL\_NORMAL\_SIZE

```
FL_EXPORT Fl_Fontsize FL_NORMAL_SIZE
```

normal font size

normal font size

## 32.2 filename.H File Reference

File names and URI utility functions.

```
#include "Fl_Export.H"
#include <FL/platform_types.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

- `FL_EXPORT void fl_decode_uri (char *uri)`  
*Decodes a URL-encoded string.*
- `FL_EXPORT int fl_filename_absolute (char *to, int tolen, const char *from)`  
*Makes a filename absolute from a relative filename.*
- `FL_EXPORT int fl_filename_expand (char *to, int tolen, const char *from)`  
*Expands a filename containing shell variables and tilde (~).*
- `FL_EXPORT const char * fl_filename_ext (const char *buf)`  
*Gets the extension of a filename.*
- `FL_EXPORT void fl_filename_free_list (struct dirent ***l, int n)`  
*Free the list of filenames that is generated by `fl_filename_list()`.*
- `FL_EXPORT int fl_filename_isdir (const char *name)`  
*Determines if a file exists and is a directory from its filename.*
- `FL_EXPORT 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.*
- `FL_EXPORT int fl_filename_match (const char *name, const char *pattern)`  
*Checks if a string s matches a pattern p.*
- `FL_EXPORT const char * fl_filename_name (const char *filename)`  
*Gets the file name from a path.*
- `FL_EXPORT int fl_filename_relative (char *to, int tolen, const char *from)`  
*Makes a filename relative to the current working directory.*
- `FL_EXPORT char * fl_filename_setext (char *to, int tolen, const char *ext)`  
*Replaces the extension in buf of max.*
- `FL_EXPORT int fl_open_uri (const char *uri, char *msg, int msglen)`  
*Opens the specified Uniform Resource Identifier (URI).*

### 32.2.1 Detailed Description

File names and URI utility functions.

## 32.3 Fl.hxx File Reference

Implementation of the member functions of class `Fl`.

```
#include "config_lib.h"
#include <FL/Fl.H>
#include <FL/platform.H>
#include "Fl_Screen_Driver.H"
#include "Fl_Window_Driver.H"
#include "Fl_System_Driver.H"
#include <FL/Fl_Window.H>
#include <FL/Fl_Tooltip.H>
#include <FL/fl_draw.H>
#include <ctype.h>
#include <stdlib.h>
#include "flstring.h"
```

## Macros

- #define **FOREVER** 1e20

## Functions

- bool **fl\_clipboard\_notify\_empty** (void)
- void **fl\_close\_display** ()
- const char \* **fl\_filename\_name** (const char \*name)
 

*Gets the file name from a path.*
- **FL\_Window \* fl\_find** (Window xid)
- 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)
- **FL\_EXPORT Window fl\_xid\_** (const **FL\_Window** \*w)

## Variables

- **FL\_EXPORT const char \* fl\_local\_alt** = **Fl::system\_driver()**->alt\_name()  
*string pointer used in shortcuts, you can change it to another language*
- **FL\_EXPORT const char \* fl\_local\_ctrl** = **Fl::system\_driver()**->control\_name()  
*string pointer used in shortcuts, you can change it to another language*
- int(\* **fl\_local\_grab** )(int)
- **FL\_EXPORT const char \* fl\_local\_meta** = **Fl::system\_driver()**->meta\_name()  
*string pointer used in shortcuts, you can change it to another language*
- **FL\_EXPORT const char \* fl\_local\_shift** = **Fl::system\_driver()**->shift\_name()  
*string pointer used in shortcuts, you can change it to another language*
- **FL\_Widget** \* **fl\_oldfocus**
- **FL\_Widget** \* **fl\_selection\_requestor**

### 32.3.1 Detailed Description

Implementation of the member functions of class **Fl**.

### 32.3.2 Function Documentation

#### 32.3.2.1 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.

## 32.4 Fl.H File Reference

[Fl](#) static class.

```
#include <FL/Fl_Export.h>
#include <FL/platform_types.h>
#include <FL/Fl_Cairo.h>
#include "fl_utf8.h"
#include "Enumerations.h"
```

### Classes

- class [Fl](#)

*The [Fl](#) is the FLTK global (static) class containing state information and global methods for the current application.*

- class [Fl\\_Widget\\_Tracker](#)

*This class should be used to control safe widget deletion.*

### Macros

- #define [Fl\\_Object](#) [Fl\\_Widget](#)  
*for back compatibility - use [Fl\\_Widget](#)!*

### 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\\_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 functions passed as parameters.*
- 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 set\_idle callback functions passed as parameters.*
- 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 some timeout callback functions passed as parameters.*

## Variables

- `FL_EXPORT const char * fl_local_alt`  
*string pointer used in shortcuts, you can change it to another language*
- `FL_EXPORT const char * fl_local_ctrl`  
*string pointer used in shortcuts, you can change it to another language*
- `FL_EXPORT const char * fl_local_meta`  
*string pointer used in shortcuts, you can change it to another language*
- `FL_EXPORT const char * fl_local_shift`  
*string pointer used in shortcuts, you can change it to another language*

### 32.4.1 Detailed Description

`Fl` static class.

## 32.5 fl\_arc.cxx File Reference

Utility functions for drawing arcs and circles.

```
#include <FL/fl_draw.H>
#include <FL/math.h>
```

### 32.5.1 Detailed Description

Utility functions for drawing arcs and circles.

## 32.6 fl\_ask.cxx File Reference

Utility functions for common dialogs.

```
#include <stdio.h>
#include <stdarg.h>
#include "flstring.h"
#include <FL/Fl.H>
#include <FL/fl_string.h>
#include <FL/fl_ask.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Button.H>
#include <FL/Fl_Return_Button.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Input.H>
#include <FL/Fl_Secret_Input.H>
#include <FL/platform.H>
#include "Fl_Screen_Driver.H"
#include <FL/fl_draw.H>
```

## Functions

- void `fl_alert` (const char \*fmt,...)
 

*Shows an alert message dialog box.*
- int `fl_ask` (const char \*fmt,...)
 

*Shows a dialog displaying the `fmt` message, this dialog features 2 yes/no buttons.*
- void `fl_beep` (int type)
 

*Emits a system beep message.*
- int `fl_choice` (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.*
- void `fl_message` (const char \*fmt,...)
 

*Shows an information message dialog box.*
- void `fl_message_hotspot` (int enable)
 

*Sets whether or not to move the common 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 common 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_position` (const int x, const int y, const int center)
 

*Sets the preferred position for the common 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 common 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 common 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.*

## 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*

### 32.6.1 Detailed Description

Utility functions for common dialogs.

## 32.7 fl\_ask.H File Reference

API for common dialogs.

```
#include "Enumerations.H"
```

## Macros

- #define **fl\_attr(x)**

## Enumerations

- ```
• enum FL_Beep {  
    FL_BEEP_DEFAULT = 0, FL_BEEP_MESSAGE, FL_BEEP_ERROR, FL_BEEP_QUESTION,  
    FL_BEEP_PASSWORD, FL_BEEP_NOTIFICATION }
```

Different system beeps available.

Functions

- FL_EXPORT void FL_EXPORT void **fl_alert** (const char *,...) __fl_attr((__format__(__printf__))
 - FL_EXPORT void FL_EXPORT void FL_EXPORT int **fl_ask** (const char *,...) __fl_attr((__format__(__printf__))
 - FL_EXPORT void **fl_beep** (int type=**FL_BEEP_DEFAULT**)
 - Emits a system beep message.*
 - FL_EXPORT int **fl_choice** (const char *q, const char *b0, const char *b1, const char *b2,...) __fl_attr((__format__(__printf__))
 - FL_EXPORT int FL_EXPORT const char * **fl_input** (const char *label, const char *deflt=0,...) __fl_attr((__format__(__printf__))
 - FL_EXPORT void **fl_message** (const char *,...) __fl_attr((__format__(__printf__))
 - void **fl_message_font** (**FL_Font** f, **FL_Fontsize** s)
 - FL_EXPORT void **fl_message_hotspot** (int enable)
 - Sets whether or not to move the common 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_EXPORT int **fl_message_hotspot** (void)
 - Gets whether or not to move the common 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_EXPORT int FL_EXPORT const char FL_EXPORT const char FL_EXPORT **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()***
 - FL_EXPORT void **fl_message_position** (const int x, const int y, const int center=0)
 - Sets the preferred position for the common message box used in many common dialogs like **fl_message()**, **fl_alert()**, **fl_ask()**, **fl_choice()**, **fl_input()**, **fl_password()**.*
 - FL_EXPORT void **fl_message_position** (**FL_Widget** *widget)

- Sets the preferred position for the common message box used in many common dialogs like `fl_message()`, `fl_alert()`, `fl_ask()`, `fl_choice()`, `fl_input()`, `fl_password()`.
- FL_EXPORT int `fl_message_position` (int *x=0, int *y=0)
 - Gets the preferred position for the common 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)
- FL_EXPORT void `fl_message_title` (const char *title)
 - Sets the title of the dialog window used in many common dialogs.
- FL_EXPORT void `fl_message_title_default` (const char *title)
 - Sets the default title of the dialog window used in many common dialogs.
- FL_EXPORT int FL_EXPORT const char FL_EXPORT const char * `fl_password` (const char *label, const char *deflt=0,...) __fl_attr((__format__(__printf__

Variables

- FL_EXPORT void FL_EXPORT void FL_EXPORT int __deprecated__
- FL_EXPORT const char * `fl_cancel`
 - string pointer used in common dialogs, you can change it to another language
- FL_EXPORT const char * `fl_close`
 - string pointer used in common dialogs, you can change it to another language
- FL_EXPORT FL_Font `fl_message_font`
- FL_EXPORT FL_Fontsize `fl_message_size`
- FL_EXPORT const char * `fl_no`
 - string pointer used in common dialogs, you can change it to another language
- FL_EXPORT const char * `fl_ok`
 - string pointer used in common dialogs, you can change it to another language
- FL_EXPORT const char * `fl_yes`
 - string pointer used in common dialogs, you can change it to another language

32.7.1 Detailed Description

API for common dialogs.

32.7.2 Enumeration Type Documentation

32.7.2.1 FL_Beep

```
enum Fl_Beep
```

Different system beeps available.

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.

32.7.3 Function Documentation

32.7.3.1 [fl_message_position\(\)](#)

```
void fl_message_position (
    Fl_Widget & widget ) [inline]
```

See also

[fl_message_position\(Fl_Widget *widget\)](#).

32.8 [fl_boxtype.cxx](#) File Reference

drawing code for common box types.

```
#include <FL/Fl.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

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_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_embossed_box` (int x, int y, int w, int h, `Fl_Color` c)
Draws a box of type FL_EMBOSSSED_BOX.
- void `fl_embossed_frame` (int x, int y, int w, int h, `Fl_Color`)
Draws a frame of type FL_EMBOSSSED_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.
- const `uchar` * `fl_gray_ramp` ()
- void `fl_internal_boxtype` (`Fl_Boxtype` t, `Fl_Box_Draw_F` *f)
Sets the drawing function for a given box type.
- void `fl_no_box` (int, int, int, int, `Fl_Color`)
Draws a box of type FL_NO_BOX.
- 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_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.

32.8.1 Detailed Description

drawing code for common box types.

32.8.2 Function Documentation

32.8.2.1 fl_internal_boxtype()

```
void fl_internal_boxtype (
    Fl_Boxtype t,
    Fl_Box_Draw_F * f )
```

Sets the drawing function for a given box type.

Parameters

in	<i>t</i>	box type
in	<i>f</i>	box drawing function

32.8.2.2 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.

32.9 fl_color.hxx 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_contrast (Fl_Color fg, Fl_Color bg)**
Returns a color that contrasts with the background color.
- **Fl_Color fl_inactive (Fl_Color c)**
Returns the inactive, dimmed version of the given color.

Variables

- unsigned **fl_cmap** [256]

32.9.1 Detailed Description

Color handling.

32.10 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>
```

Classes

- class [Fl_Color_Chooser](#)

The [Fl_Color_Chooser](#) widget provides a standard RGB color chooser.

32.10.1 Detailed Description

[Fl_Color_Chooser](#) widget .

32.11 Fl_compose.cxx File Reference

Utility functions to support text input.

```
#include <FL/Fl.H>
#include "Fl_Screen_Driver.H"
```

32.11.1 Detailed Description

Utility functions to support text input.

32.12 fl_curve.cxx File Reference

Utility for drawing Bezier curves, adding the points to the current fl_begin/fl_vertex/fl_end path.

```
#include <FL/fl_draw.H>
#include <math.h>
```

32.12.1 Detailed Description

Utility for drawing Bezier 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!

32.13 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>
#include <stdlib.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](#)
A display to which the computer can draw.
- class [Fl_Surface_Device](#)
A drawing surface that's susceptible to receive graphical output.

32.13.1 Detailed Description

declaration of classes [Fl_Surface_Device](#), [Fl_Display_Device](#), [Fl_Device_Plugin](#).

32.14 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"
```

32.14.1 Detailed Description

[Fl_Double_Window](#) implementation.

32.15 fl_draw.H File Reference

utility header to pull drawing functions together

```
#include <FL/Enumerations.H>
#include <FL/Fl_Graphics_Driver.H>
```

Macros

- `#define fl_clip fl_push_clip`

Intersects the current clip region with a rectangle and pushes this new region onto the stack (deprecated).

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 }

Functions

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

Adds a symbol to the system.
- `void fl_arc (int x, int y, int w, int h, double a1, double a2)`

Draw ellipse sections using integer coordinates.
- `void fl_arc (double x, double y, double r, double start, double end)`

Adds a series of points to the current path on the arc of a circle.
- `void fl_begin_complex_polygon ()`

Starts drawing a complex filled polygon.
- `void fl_begin_line ()`

Starts drawing a list of lines.
- `void fl_begin_loop ()`

Starts drawing a closed sequence of lines.
- `FL_EXPORT void fl_begin_offscreen (Fl_Offscreen ctx)`

Send all subsequent drawing commands to this offscreen buffer.
- `void fl_begin_points ()`

Starts drawing a list of points.
- `void fl_begin_polygon ()`

Starts drawing a convex filled polygon.
- `char fl_can_do_alpha_blending ()`

Checks whether platform supports true alpha blending for RGBA images.
- `FL_EXPORT Fl_RGB_Image * fl_capture_window_part (Fl_Window *win, int x, int y, int w, int h)`

Captures the content of a rectangular zone of a mapped window.

- FL_EXPORT 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)

Intersects a rectangle with the current clip region and returns the bounding box of the result.
- void [fl_clip_region](#) ([FI_Region](#) r)

Replaces the top of the clipping stack with a clipping region of any shape.
- [FI_Region](#) [fl_clip_region](#) ()

Returns the current clipping region.
- void [fl_color](#) ([FI_Color](#) c)

Sets the color for all subsequent drawing operations.
- void [fl_color](#) (int c)

for back compatibility - use fl_color(FI_Color c) instead
- void [fl_color](#) (uchar r, uchar g, uchar b)

Sets the color for all subsequent drawing operations.
- [FI_Color](#) [fl_color](#) ()

Returns the last fl_color() that was set.
- void [fl_copy_offscreen](#) (int x, int y, int w, int h, [FI_Offscreen](#) pixmap, int srcx, int srcy)

Copy a rectangular area of the given offscreen buffer into the current drawing destination.
- FL_EXPORT [FI_Offscreen](#) [fl_create_offscreen](#) (int w, int h)

Creation of an offscreen graphics buffer.
- FL_EXPORT void [fl_cursor](#) ([FI_Cursor](#))

Sets the cursor for the current window to the specified shape and colors.
- FL_EXPORT void [fl_cursor](#) ([FI_Cursor](#), [FI_Color](#) fg, [FI_Color](#) bg=FL_WHITE)
- void [fl_curve](#) (double X0, double Y0, double X1, double Y1, double X2, double Y2, double X3, double Y3)

Adds a series of points on a Bezier curve to the path.
- FL_EXPORT void [fl_delete_offscreen](#) ([FI_Offscreen](#) ctx)

Deletion of an offscreen graphics buffer.
- int [fl_descent](#) ()

Returns 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_EXPORT void [fl_draw](#) (const char *str, int x, int y)

Draws a nul-terminated UTF-8 string starting at the given x, y location.
- FL_EXPORT void [fl_draw](#) (int angle, const char *str, int x, int y)

Draws a nul-terminated UTF-8 string starting at the given x, y location and rotating angle degrees counter-clockwise.
- 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](#) (int angle, const char *str, int n, int x, int y)

Draws at the given x, y location a UTF-8 string of length n bytes rotating angle degrees counter-clockwise.
- FL_EXPORT void [fl_draw](#) (const char *str, int x, int y, int w, int h, [FI_Align](#) align, [FI_Image](#) *img=0, int draw←_symbols=1)

Fancy string drawing function which is used to draw all the labels.
- FL_EXPORT void [fl_draw](#) (const char *str, int x, int y, int w, int h, [FI_Align](#) align, void(*callthis)(const char *, int, int, int), [FI_Image](#) *img=0, int draw_symbols=1)

The same as fl_draw(const char,int,int,int,FI_Align,FI_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.*
- FL_EXPORT void [fl_draw_box](#) ([FI_Boxtype](#), int x, int y, int w, int h, [FI_Color](#))

Draws a box using given type, position, size and color.
- void [fl_draw_image](#) (const uchar *buf, int X, int Y, int W, int H, int D=3, int L=0)

- Draws 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)
Draws 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)
Draws 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)
Draws a gray-scale image using a callback function to generate image data.
 - FL_EXPORT int `fl_drawPixmap` (const 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_drawPixmap` (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.
 - FL_EXPORT 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` ()
Ends complex filled polygon, and draws.
 - void `fl_end_line` ()
Ends list of lines, and draws.
 - void `fl_end_loop` ()
Ends closed sequence of lines, and draws.
 - FL_EXPORT void `fl_end_offscreen` ()
Quit sending drawing commands to the current offscreen buffer.
 - void `fl_end_points` ()
Ends list of points, and draws.
 - void `fl_end_polygon` ()
Ends convex filled polygon, and draws.
 - 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)
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_EXPORT void `fl_font` (Fl_Font face, Fl_Fontsize fsize)
Sets the current font, which is then used in various drawing routines.
 - `Fl_Font fl_font` ()
Returns the `face` set by the most recent call to `fl_font()`.
 - FL_EXPORT void `fl_frame` (const char *s, int x, int y, int w, int h)
Draws a series of line segments around the given box.
 - FL_EXPORT 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` ()
Call `fl_gap()` to separate loops of the path.
 - int `fl_height` ()
Returns the recommended minimum line spacing for the current font.
 - FL_EXPORT int `fl_height` (int font, int size)
This function returns the actual height of the specified `font` and `size`.
 - FL_EXPORT const char * `fl_latin1_to_local` (const char *t, int n=-1)
Converts text from Windows/X11 latin1 character set to local encoding.
 - void `fl_line` (int x, int y, int x1, int y1)
Draws a line from (x,y) to (x1,y1)
 - void `fl_line` (int x, int y, int x1, int y1, int x2, int y2)
Draws 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)

- Sets how to draw lines (the "pen").
- FL_EXPORT const char * **fl_local_to_latin1** (const char *t, int n=-1)

Converts text from local encoding to Windowx/X11 latin1 character set.
- FL_EXPORT const char * **fl_local_to_mac_roman** (const char *t, int n=-1)

Converts text from local encoding to Mac Roman character set.
- void **fl_loop** (int x, int y, int x1, int y1, int x2, int y2)

Outlines 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)

Outlines a 4-sided polygon with lines.
- FL_EXPORT const char * **fl_mac_roman_to_local** (const char *t, int n=-1)

Converts text from Mac Roman character set to local encoding.
- FL_EXPORT 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.
- FL_EXPORT int **fl_measure_pixmap** (char *const *data, int &w, int &h)

Get the dimensions of a pixmap.
- FL_EXPORT int **fl_measure_pixmap** (const char *const *data, 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)

Concatenates 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?
- FL_EXPORT unsigned int **fl_old_shortcut** (const char *s)

Emulation of XForms named shortcuts.
- FL_EXPORT void **fl_overlay_clear** ()

Erase a selection rectangle without drawing a new one.
- FL_EXPORT void **fl_overlay_rect** (int x, int y, int w, int h)

Draws a selection rectangle, erasing a previous one by XOR'ing it first.
- 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)

Draws a single pixel at the given coordinates.
- void **fl_polygon** (int x, int y, int x1, int y1, int x2, int y2)

Fills a 3-sided polygon.
- void **fl_polygon** (int x, int y, int x1, int y1, int x2, int y2, int x3, int y3)

Fills a 4-sided polygon.
- void **fl_pop_clip** ()

Restores the previous clip region.
- void **fl_pop_matrix** ()

Restores the current transformation matrix from the stack.
- void **fl_push_clip** (int x, int y, int w, int h)

Intersects the current clip region with a rectangle and pushes this new region onto the stack.
- void **fl_push_matrix** ()

Saves the current transformation matrix on the stack.
- void **fl_push_no_clip** ()

Pushes an empty clip region onto the stack so nothing will be clipped.
- FL_EXPORT 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** (int x, int y, int w, int h)

Draws a 1-pixel border inside the given bounding box.
- void **fl_rect** (int x, int y, int w, int h, **FL_Color** c)

Draws with passed color a 1-pixel border inside the given bounding box.

- void `fl_rectf` (int x, int y, int w, int h)
Colors 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)
Colors with passed color a rectangle that exactly fills the given bounding box.
- FL_EXPORT void `fl_rectf` (int x, int y, int w, int h, `uchar` r, `uchar` g, `uchar` b)
Colors a rectangle with "exactly" the passed r, g, b color.
- FL_EXPORT void `fl_rescale_offscreen` (`FL_Offscreen` &ctx)
Adapts an offscreen buffer to a changed value of the scale factor.
- FL_EXPORT void `fl_reset_spot` (void)
- void `fl_restore_clip` ()
Undoes any clobbering of clip done by your program.
- void `fl_rotate` (double d)
Concatenates rotation transformation onto the current one.
- void `fl_rtl_draw` (const char *str, int n, int x, int y)
Draws a UTF-8 string of length n bytes right to left starting at the given x, y location.
- void `fl_scale` (double x, double y)
Concatenates scaling transformation onto the current one.
- void `fl_scale` (double x)
Concatenates scaling transformation onto the current one.
- 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)
Scroll a rectangle and draw the newly exposed portions.
- 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_set_status` (int X, int Y, int W, int H)
- FL_EXPORT const char * `fl_shortcut_label` (unsigned int shortcut)
Get a human-readable string from a shortcut value.
- FL_EXPORT const char * `fl_shortcut_label` (unsigned int shortcut, const char **eom)
Get a human-readable string from a shortcut value.
- `FL_Fontsize fl_size` ()
Returns the size set by the most recent call to `fl_font()`.
- FL_EXPORT void `fl_text_extents` (const char *, int &dx, int &dy, int &w, int &h)
Determines 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)
Determines the minimum pixel dimensions of a sequence of n characters using the current `fl_font()`.
- double `fl_transform_dx` (double x, double y)
Transforms distance using current transformation matrix.
- double `fl_transform_dy` (double x, double y)
Transforms distance using current transformation matrix.
- double `fl_transform_x` (double x, double y)
Transforms coordinate using the current transformation matrix.
- double `fl_transform_y` (double x, double y)
Transforms coordinate using the current transformation matrix.
- void `fl_transformed_vertex` (double xf, double yf)
Adds coordinate pair to the vertex list without further transformations.
- void `fl_translate` (double x, double y)
Concatenates translation transformation onto the current one.
- void `fl_vertex` (double x, double y)
Adds a single vertex to the current path.
- FL_EXPORT double `fl_width` (const char *txt)
Returns the typographical width of a nul-terminated string using the current font face and size.
- double `fl_width` (const char *txt, int n)

- `double fl_width (unsigned int c)`
Returns the typographical width of a sequence of n characters using the current font face and size.
- `void fl_xyline (int x, int y, int x1)`
Returns the typographical width of a single character using the current font face and size.
- `void fl_xyline (int x, int y, int x1)`
Draws a horizontal line from (x,y) to (x1,y)
- `void fl_xyline (int x, int y, int x1, int y2)`
Draws 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)`
Draws 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_xyline (int x, int y, int y1)`
Draws a vertical line from (x,y) to (x,y1)
- `void fl_xyline (int x, int y, int y1, int x2)`
Draws a vertical line from (x,y) to (x,y1), then a horizontal from (x,y1) to (x2,y1)
- `void fl_xyline (int x, int y, int y1, int x2, int y3)`
Draws 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

- `FL_EXPORT char fl_draw_shortcut`

32.15.1 Detailed Description

utility header to pull drawing functions together

32.16 Fl_Image.H File Reference

[Fl_Image](#), [Fl_RGB_Image](#) classes.

```
#include "Enumerations.H"
#include <stdlib.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.

32.16.1 Detailed Description

[Fl_Image](#), [Fl_RGB_Image](#) classes.

32.16.2 Enumeration Type Documentation

32.16.2.1 Fl_RGB_Scaling

enum [Fl_RGB_Scaling](#)

The scaling algorithm to use for RGB images.

Enumerator

<code>FL_RGB_SCALING_NEAREST</code>	default RGB image scaling algorithm
<code>FL_RGB_SCALING_BILINEAR</code>	more accurate, but slower RGB image scaling algorithm

32.17 Fl_Menu_Item.H File Reference

```
#include "Fl_Widget.H"
#include "Fl_Image.H"
#include <FL/platform_types.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 }
- 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` }

Functions

- FL_EXPORT [Fl_Shortcut fl_old_shortcut](#) (const char *)

Emulation of XForms named shortcuts.

32.17.1 Enumeration Type Documentation

32.17.1.1 anonymous enum

anonymous enum

Enumerator

<code>FL_MENU_INACTIVE</code>	Deactivate menu item (gray out)
<code>FL_MENU_TOGGLE</code>	Item is a checkbox toggle (shows checkbox for on/off state)
<code>FL_MENU_VALUE</code>	The on/off state for checkbox/radio buttons (if set, state is 'on')
<code>FL_MENU_RADIO</code>	Item is a radio button (one checkbox of many can be on)
<code>FL_MENU_INVISIBLE</code>	Item will not show up (shortcut will work)
<code>FL_SUBMENU_POINTER</code>	Indicates user_data() is a pointer to another menu array.
<code>FL_SUBMENU</code>	This item is a submenu to other items.
<code>FL_MENU_DIVIDER</code>	Creates divider line below this item. Also ends a group of radio buttons.
<code>FL_MENU_HORIZONTAL</code>	??? – reserved

32.18 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.

32.18.1 Detailed Description

[Fl_Native_File_Chooser](#) widget.

32.19 Fl_Paged_Device.hxx File Reference

implementation of class [Fl_Paged_Device](#).

```
#include <FL/Fl_Paged_Device.H>
#include <FL/Fl.H>
#include <FL/fl_draw.H>
```

32.19.1 Detailed Description

implementation of class [Fl_Paged_Device](#).

32.20 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.

32.20.1 Detailed Description

declaration of class [Fl_Paged_Device](#).

32.21 Fl_PostScript.H File Reference

declaration of classes [Fl_PostScript_Graphics_Driver](#), [Fl_PostScript_File_Device](#).

```
#include <FL/Fl_Paged_Device.H>
#include <FL/fl_draw.H>
#include <stdarg.h>
```

Classes

- class [FI_EPS_File_Surface](#)
Encapsulated PostScript drawing surface.
- class [FI_PostScript_File_Device](#)
To send graphical output to a PostScript file.

Typedefs

- `typedef int() FI_PostScript_Close_Command(FILE *)`

32.21.1 Detailed Description

declaration of classes `FI_PostScript_Graphics_Driver`, [FI_PostScript_File_Device](#).

32.22 FI_Printer.H File Reference

declaration of class [FI_Printer](#).

```
#include <FL/F1_Paged_Device.H>
```

Classes

- class [FI_Printer](#)
OS-independent print support.

32.22.1 Detailed Description

declaration of class [FI_Printer](#).

32.23 fl_rect.cxx File Reference

Drawing and clipping routines for rectangles.

```
#include <FL/platform.H>
#include <FL/F1_Graphics_Driver.H>
```

32.23.1 Detailed Description

Drawing and clipping routines for rectangles.

32.24 Fl_Shared_Image.H File Reference

[Fl_Shared_Image](#) class.

```
#include "Fl_Image.h"
```

Classes

- 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)

Functions

- [FL_EXPORT void fl_register_images \(\)](#)
Register the image formats.

32.24.1 Detailed Description

[Fl_Shared_Image](#) class.

32.24.2 Function Documentation

32.24.2.1 [fl_register_images\(\)](#)

```
FL_EXPORT void fl_register_images ( )
```

Register the image formats.

This 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.

32.25 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_EXPORT Fl_Color fl_show_colormap (Fl_Color oldcol)
Pops up a window to let the user pick a colormap entry.

32.25.1 Detailed Description

The [fl_show_colormap\(\)](#) function hides the implementation classes used to provide the popup window and color selection mechanism.

32.26 fl_string.h File Reference

Public header for FLTK's own platform agnostic string handling.

```
#include "Fl_Export.h"
#include "fl_types.h"
```

Functions

- FL_EXPORT char * [fl_strdup](#) (const char *s)
Cross platform interface to POSIX function strdup().

32.26.1 Detailed Description

Public header for FLTK's own platform agnostic string handling.

32.27 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.

32.27.1 Detailed Description

Definition of class [Fl_Sys_Menu_Bar](#).

32.28 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](#) =0, [FL_TREE_REASON_SELECTED](#), [FL_TREE_REASON_DESELECTED](#),
 [FL_TREE_REASON_RESELECTED](#),
 [FL_TREE_REASON_OPENED](#), [FL_TREE_REASON_CLOSED](#), [FL_TREE_REASON_DRAGGED](#) }

The reason the callback was invoked.

32.28.1 Detailed Description

This file contains the definitions of the [Fl_Tree](#) class.

32.28.2 Enumeration Type Documentation

32.28.2.1 Fl_Tree_Reason

```
enum Fl\_Tree\_Reason
```

The reason the callback was invoked.

Enumerator

<code>FL_TREE_REASON_NONE</code>	unknown reason
<code>FL_TREE_REASON_SELECTED</code>	an item was selected
<code>FL_TREE_REASON_DESELECTED</code>	an item was de-selected
<code>FL_TREE_REASON_RESELECTED</code>	an item was re-selected (double-clicked). See FL_Tree_Item_Reselect_Mode to enable this.
<code>FL_TREE_REASON_OPENED</code>	an item was opened
<code>FL_TREE_REASON_CLOSED</code>	an item was closed
<code>FL_TREE_REASON_DRAGGED</code>	an item was dragged into a new place

32.29 `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.

32.29.1 Detailed Description

This file contains the definitions for [FL_Tree_Item](#).

32.30 `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.

Variables

- class FL_EXPORT Fl_Tree_Item

32.30.1 Detailed Description

This file defines a class that manages an array of [Fl_Tree_Item](#) pointers.

32.31 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](#) =1 }
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.

32.31.1 Detailed Description

This file contains the definitions for `Fl_Tree`'s preferences.

```

Fl_Tree_Prefs
  :
  .....
  :
  Fl_Tree
    :
  |____ Fl_Tree_Item

```

32.31.2 Enumeration Type Documentation

32.31.2.1 Fl_Tree_Connector

```
enum Fl_Tree_Connector
```

Defines the style of connection lines between items.

Enumerator

<code>FL_TREE_CONNECTOR_NONE</code>	Use no lines connecting items.
<code>FL_TREE_CONNECTOR_DOTTED</code>	Use dotted lines connecting items (default)
<code>FL_TREE_CONNECTOR_SOLID</code>	Use solid lines connecting items.

32.31.2.2 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

<code>FL_TREE_ITEM_DRAW_DEFAULT</code>	If <code>widget()</code> defined, draw in place of label, and <code>widget()</code> tracks item height (default)
<code>FL_TREE_ITEM_DRAW_LABEL_AND_WIDGET</code>	If <code>widget()</code> defined, include label to the left of the widget.
<code>FL_TREE_ITEM_HEIGHT_FROM_WIDGET</code>	If <code>widget()</code> defined, <code>widget()</code> 's height controls item's height.

32.31.2.3 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

<code>FL_TREE_SELECTABLE_ONCE</code>	Item can only be selected once (default)
<code>FL_TREE_SELECTABLE_ALWAYS</code>	Enables <code>FL_TREE_REASON_RESELECTED</code> events for callbacks.

32.31.2.4 Fl_Tree_Select

```
enum Fl_Tree_Select
```

Tree selection style.

Enumerator

<code>FL_TREE_SELECT_NONE</code>	Nothing selected when items are clicked.
<code>FL_TREE_SELECT_SINGLE</code>	Single item selected when item is clicked (default)
<code>FL_TREE_SELECT_MULTI</code>	Multiple items can be selected by clicking with SHIFT, CTRL or mouse drags.
<code>FL_TREE_SELECT_SINGLE_DRAGGABLE</code>	Single items may be selected, and they may be reordered by mouse drag.

32.31.2.5 Fl_Tree_Sort

```
enum Fl_Tree_Sort
```

Sort order options for items added to the tree.

Enumerator

<code>FL_TREE_SORT_NONE</code>	No sorting; items are added in the order defined (default).
<code>FL_TREE_SORT_ASCENDING</code>	Add items in ascending sort order.
<code>FL_TREE_SORT_DESCENDING</code>	Add items in descending sort order.

32.32 fl_types.h File Reference

This file contains simple "C"-style type definitions.

TypeDefs**Miscellaneous**

- `typedef unsigned char uchar`

- unsigned char*
- **typedef unsigned long ulong**
unsigned long
- **typedef unsigned int Fl_Shortcut**
16-bit Unicode character + 8-bit indicator for keyboard flags.
- **typedef unsigned int Fl_Char**
24-bit Unicode character - upper 8 bits are unused

32.32.1 Detailed Description

This file contains simple "C"-style type definitions.

32.32.2 Typedef Documentation

32.32.2.1 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.

32.33 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 <string.h>
#include <stdlib.h>
```

Functions

- FL_EXPORT int **fl_access** (const char *f, int mode)

Cross-platform function to test a files access() with a UTF-8 encoded name or value.
- FL_EXPORT int **fl_chdir** (const char *path)

Cross-platform function to change the current working directory, given as a UTF-8 encoded string.
- FL_EXPORT int **fl_chmod** (const char *f, int mode)

Cross-platform function to set a files mode() with a UTF-8 encoded name or value.
- FL_EXPORT int **fl_execvp** (const char *file, char *const *argv)
- FL_EXPORT FILE * **fl_fopen** (const char *f, const char *mode)

Cross-platform function to open files with a UTF-8 encoded name.
- FL_EXPORT char * **fl_getcwd** (char *buf, int len)

Cross-platform function to get the current working directory as a UTF-8 encoded value.
- FL_EXPORT char * **fl_getenv** (const char *v)

Cross-platform function to get environment variables with a UTF-8 encoded name or value.
- FL_EXPORT char **fl_make_path** (const char *path)

Cross-platform function to recursively create a path in the file system.
- FL_EXPORT void **fl_make_path_for_file** (const char *path)

Cross-platform function to create a path for the file in the file system.
- FL_EXPORT int **fl_mkdir** (const char *f, int mode)

Cross-platform function to create a directory with a UTF-8 encoded name.
- FL_EXPORT unsigned int **fl_nonspacing** (unsigned int ucs)

Returns true if the Unicode character `ucs` is non-spacing.
- FL_EXPORT int **fl_open** (const char *fname, int oflags,...)

Cross-platform function to open files with a UTF-8 encoded name.
- FL_EXPORT int **fl_open_ext** (const char *fname, int binary, int oflags,...)

Cross-platform function to open files with a UTF-8 encoded name.
- FL_EXPORT int **fl_putenv** (const char *var)

Cross-platform function to write environment variables with a UTF-8 encoded name or value.
- FL_EXPORT int **fl_rename** (const char *f, const char *n)

Cross-platform function to rename a filesystem object using UTF-8 encoded names.
- FL_EXPORT int **fl_rmdir** (const char *f)

Cross-platform function to remove a directory with a UTF-8 encoded name.
- FL_EXPORT int **fl_stat** (const char *f, struct stat *b)

Cross-platform function to stat() a file using a UTF-8 encoded name or value.
- FL_EXPORT int **fl_system** (const char *cmd)

Cross-platform function to run a system command with a UTF-8 encoded string.
- FL_EXPORT int **fl_tolower** (unsigned int ucs)

Returns the Unicode lower case value of `ucs`.
- FL_EXPORT int **fl_toupper** (unsigned int ucs)

Returns the Unicode upper case value of `ucs`.
- FL_EXPORT 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.
- FL_EXPORT int **fl_unlink** (const char *filename)

Cross-platform function to unlink() (that is, delete) a file using a UTF-8 encoded filename.
- FL_EXPORT char * **fl_utf2mbcs** (const char *s)

Converts UTF-8 string `s` to a local multi-byte character string.
- FL_EXPORT 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.
- FL_EXPORT int **fl_utf8bytes** (unsigned ucs)

Return the number of bytes needed to encode the given UCS4 character in UTF-8.

- FL_EXPORT unsigned [fl_utf8decode](#) (const char *p, const char *end, int *len)

Decode a single UTF-8 encoded character starting at p.
- FL_EXPORT int [fl_utf8encode](#) (unsigned ucs, char *buf)

Write the UTF-8 encoding of ucs into buf and return the number of bytes written.
- FL_EXPORT 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.
- FL_EXPORT 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.
- FL_EXPORT 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.
- FL_EXPORT 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.
- FL_EXPORT int [fl_utf8len](#) (char c)

Returns the byte length of the UTF-8 sequence with first byte c, or -1 if c is not valid.
- FL_EXPORT 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.
- FL_EXPORT int [fl_utf8locale](#) ()

Return true if the "locale" seems to indicate that UTF-8 encoding is used.
- FL_EXPORT 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.
- FL_EXPORT 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).
- FL_EXPORT unsigned [fl_utf8toa](#) (const char *src, unsigned srclen, char *dst, unsigned dstlen)

Convert a UTF-8 sequence into an array of 1-byte characters.
- FL_EXPORT 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.
- FL_EXPORT unsigned [fl_utf8towc](#) (const char *src, unsigned srclen, wchar_t *dst, unsigned dstlen)

Converts a UTF-8 string into a wide character string.
- FL_EXPORT int [fl_utf_nb_char](#) (const unsigned char *buf, int len)

Returns the number of Unicode chars in the UTF-8 string.
- FL_EXPORT int [fl_utf_strcasecmp](#) (const char *s1, const char *s2)

UTF-8 aware strcasecmp - converts to Unicode and tests.
- FL_EXPORT int [fl_utf_strncasecmp](#) (const char *s1, const char *s2, int n)

UTF-8 aware strncasecmp - converts to lower case Unicode and tests.
- FL_EXPORT int [fl_utf_tolower](#) (const unsigned char *str, int len, char *buf)

Converts the string str to its lower case equivalent into buf.
- FL_EXPORT int [fl_utf_toupper](#) (const unsigned char *str, int len, char *buf)

Converts the string str to its upper case equivalent into buf.
- FL_EXPORT int [fl_wcwidth](#) (const char *src)

extended wrapper around fl_wcwidth_(unsigned int ucs) function.
- FL_EXPORT int [fl_wcwidth_](#) (unsigned int ucs)

Wrapper to adapt Markus Kuhn's implementation of wcwidth() for FLTK.

32.33.1 Detailed Description

header for Unicode and UTF-8 character handling

32.34 fl_vertex.hxx 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>
```

32.34.1 Detailed Description

Portable drawing code for drawing arbitrary shapes with simple 2D transformations.

32.35 Fl_Widget.H File Reference

[Fl_Widget](#), [Fl_Label](#) classes .

```
#include "Enumerations.H"
#include "Fl.H"
```

Classes

- 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.

32.35.1 Detailed Description

[Fl_Widget](#), [Fl_Label](#) classes .

32.35.2 Macro Definition Documentation

32.35.2.1 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.

32.36 Fl_Window.H File Reference

[Fl_Window](#) widget .

```
#include <FL/Fl.H>
#include <FL/Fl_Group.H>
#include <FL/Fl_Bitmap.H>
#include <stdlib.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

32.36.1 Detailed Description

[Fl_Window](#) widget .

32.37 gl.h File Reference

This file defines wrapper functions for OpenGL in FLTK.

```
#include "Enumerations.H"
#include <GL/gl.h>
```

Functions

- **FL_EXPORT void gl_color (Fl_Color i)**
Sets the current OpenGL color to an FLTK color.
- **void gl_color (int c)**
back compatibility
- **FL_EXPORT int gl_descent ()**
Returns the current font's descent.
- **FL_EXPORT void gl_draw (const char *)**
Draws a nul-terminated string in the current font at the current position.
- **FL_EXPORT void gl_draw (const char *, int n)**
Draws an array of n characters of the string in the current font at the current position.
- **FL_EXPORT void gl_draw (const char *, int x, int y)**
Draws a nul-terminated string in the current font at the given position.
- **FL_EXPORT void gl_draw (const char *, float x, float y)**
Draws a nul-terminated string in the current font at the given position.
- **FL_EXPORT 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.
- **FL_EXPORT 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.
- **FL_EXPORT 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.
- **FL_EXPORT void gl_draw_image (const uchar *, int x, int y, int w, int h, int d=3, int ld=0)**
- **FL_EXPORT void gl_finish ()**
Releases an OpenGL context.
- **FL_EXPORT void gl_font (int fontid, int size)**
Sets the current OpenGL font to the same font as calling fl_font().
- **FL_EXPORT int gl_height ()**
Returns the current font's height.
- **FL_EXPORT 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.
- **FL_EXPORT 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.
- **FL_EXPORT void gl_start ()**
Creates an OpenGL context.
- **FL_EXPORT void gl_texture_pile_height (int max)**
Changes the maximum height of the pile of pre-computed string textures.
- **FL_EXPORT int gl_texture_pile_height ()**
Returns the current maximum height of the pile of pre-computed string textures.
- **FL_EXPORT double gl_width (const char *)**
Returns the width of the string in the current font.
- **FL_EXPORT double gl_width (const char *, int n)**
Returns the width of n characters of the string in the current font.
- **FL_EXPORT double gl_width (uchar)**
Returns the width of the character in the current font.

32.37.1 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)

32.37.2 Function Documentation

32.37.2.1 `gl_color()`

```
FL_EXPORT void gl_color (
    Fl_Color i )
```

Sets the current 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!

32.37.2.2 `gl_draw()` [1/7]

```
FL_EXPORT 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\(int\)](#)

32.37.2.3 `gl_draw()` [2/7]

```
FL_EXPORT 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\(int\)](#)

32.37.2.4 gl_draw() [3/7]

```
FL_EXPORT 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\(int\)](#)

32.37.2.5 gl_draw() [4/7]

```
FL_EXPORT 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\(int\)](#)

32.37.2.6 gl_draw() [5/7]

```
FL_EXPORT 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\(int\)](#)

32.37.2.7 gl_draw() [6/7]

```
FL_EXPORT 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\(int\)](#)

32.37.2.8 gl_draw() [7/7]

```
FL_EXPORT void gl_draw (
    const char * str,
    int x,
    int y,
    int w,
    int h,
    F1_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\(\)](#).

32.37.2.9 gl_font()

```
FL_EXPORT 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\(int val\)](#)

32.37.2.10 gl_rect()

```
FL_EXPORT void gl_rect (
    int x,
    int y,
    int w,
    int h )
```

Outlines the given rectangle with the current color.

If [FI_Gl_Window::ortho\(\)](#) has been called, then the rectangle will exactly fill the given pixel rectangle.

32.37.2.11 gl_rectf()

```
void gl_rectf (
    int x,
    int y,
    int w,
    int h ) [inline]
```

Fills the given rectangle with the current color.

See also

[gl_rect\(int x, int y, int w, int h\)](#)

32.37.2.12 gl_texture_pile_height() [1/2]

```
FL_EXPORT 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

<i>max</i>	Maximum height of the texture pile
------------	------------------------------------

See also

[Fl::draw_GL_text_with_textures\(int\)](#)

32.37.2.13 gl_texture_pile_height() [2/2]

```
FL_EXPORT 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\(int\)](#)

32.38 mac.H File Reference

Mac OS X-specific symbols.

Classes

- class [Fl_Mac_App_Menu](#)

Functions

- 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.

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).

32.38.1 Detailed Description

Mac OS X-specific symbols.

32.39 numericsort.c File Reference

```
#include <ctype.h>
#include <stdlib.h>
#include <FL/platform_types.h>
#include <FL/filename.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).

32.39.1 Function Documentation

32.39.1.1 fl_casenumericssort()

```
int fl_casenumericssort (
    struct dirent ** A,
    struct dirent ** B )
```

Compares directory entries alphanumerically (case-insensitive).

Note

This comparison is not (yet) UTF-8 aware.

Todo Make comparison UTF-8 aware.

See also

[fl_numericssort\(\)](#)

32.39.1.2 fl_numericssort()

```
int fl_numericssort (
    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 not (yet) UTF-8 aware:

- UTF-8 characters are compared according to their binary values.
- Locale settings may influence the result in unexpected ways.
- The latter is particularly true for [fl_casenumericssort\(\)](#). This may be changed in a future release.

Todo Make comparison UTF-8 aware.

See also

[fl_casenumericssort\(\)](#)

32.40 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_Bitmask`
mask
- `typedef opaque fl_intptr_t`
An integral type large enough to store a pointer or a long value.
- `typedef opaque FL_Offscreen`
an offscreen drawing buffer
- `typedef opaque FL_Region`
a region made of several rectangles
- `typedef opaque FL_SOCKET`
socket or file descriptor
- `typedef opaque fl_uintptr_t`
An unsigned integral type large enough to store a pointer or an unsigned long value.
- `typedef opaque GLContext`
an OpenGL graphics context, into which all OpenGL calls are rendered

32.40.1 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.

32.40.2 Typedef Documentation

32.40.2.1 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.

32.40.2.2 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.

Index

_remove
 Fl_Browser, 418
~Fl_Check_Browser
 Fl_Check_Browser, 480
~Fl_Double_Window
 Fl_Double_Window, 522
~Fl_EPS_File_Surface
 Fl_EPS_File_Surface, 526
~Fl_File_Chooser
 Fl_File_Chooser, 537
~Fl_Group
 Fl_Group, 587
~Fl_Help_View
 Fl_Help_View, 610
~Fl_Image_Surface
 Fl_Image_Surface, 634
~Fl_Input
 Fl_Input, 645
~Fl_Menu_Window
 Fl_Menu_Window, 722
~Fl_Native_File_Chooser
 Fl_Native_File_Chooser, 735
~Fl_Plugin_Manager
 Fl_Plugin_Manager, 759
~Fl_Preferences
 Fl_Preferences, 783
~Fl_SVG_File_Surface
 Fl_SVG_File_Surface, 886
~Fl_SVG_Image
 Fl_SVG_Image, 891
~Fl_Scrollbar
 Fl_Scrollbar, 837
~Fl_Shared_Image
 Fl_Shared_Image, 846
~Fl_Surface_Device
 Fl_Surface_Device, 882
~Fl_Table
 Fl_Table, 912
~Fl_Table_Row
 Fl_Table_Row, 928
~Fl_Text_Display
 Fl_Text_Display, 971
~Fl_Widget
 Fl_Widget, 1167
~Fl_Window
 Fl_Window, 1218

abi_check
 Fl, 380
abi_version

 Fl, 380
about
 Fl_Sys_Menu_Bar, 897
absolute_top_line_number
 Fl_Text_Display, 972
activate
 Fl_Menu_Item, 708
 Fl_Tree_Item, 1107
 Fl_Widget, 1167
active
 Fl_Menu_Item, 708
 Fl_Widget, 1167
active_r
 Fl_Widget, 1168
activevisible
 Fl_Menu_Item, 709
add
 Fl_Browser, 419
 Fl_Chart, 474
 Fl_Check_Browser, 481
 Fl_File_Icon, 546
 Fl_Input_Choice, 666
 Fl_Menu_, 682, 684
 Fl_Menu_Item, 709
 Fl_Shared_Image, 846
 Fl_Sys_Menu_Bar, 897, 899
 Fl_Tree, 1058, 1059
 Fl_Tree_Item, 1107, 1108
 Fl_Tree_Item_Array, 1130
add_awake_handler_
 Fl, 380
add_check
 Fl, 380
add_clipboard_notify
 Selection & Clipboard functions, 256
add_color
 Fl_File_Icon, 547
add_default_key_bindings
 Fl_Text_Editor, 1013
add_extra
 Fl_File_Chooser, 538
add_fd
 Fl, 381
add_handler
 Events handling functions, 240
add_idle
 Fl, 382
add_key_binding
 Fl_Text_Editor, 1014

a
 add_modify_callback
 Fl_Text_Buffer, 945
 add_system_handler
 Events handling functions, 240
 add_timeout
 Fl, 382
 add_vertex
 Fl_File_Icon, 547
 addPlugin
 Fl_Plugin_Manager, 759
 address
 Fl_Text_Buffer, 945, 946
 align
 Fl_Widget, 1168
 angle1
 Fl_Dial, 520
 ansi
 Fl_Simple_Terminal, 856, 857
 api_version
 Fl, 382
 append
 Fl_Input_, 645
 Fl_Simple_Terminal, 857
 Fl_Text_Buffer, 946
 appendfile
 Fl_Text_Buffer, 946
 arg
 Fl, 383
 args
 Fl, 383, 384
 argument
 Fl_Menu_Item, 709
 Fl_Widget, 1169
 array
 Fl_Group, 588
 Fl_RGB_Image, 822
 Fl_Table, 913
 as_gl_window
 Fl_Gl_Window, 572
 Fl_Widget, 1169
 as_group
 Fl_Group, 588
 Fl_Widget, 1170
 as_shared_image
 Fl_Image, 624
 Fl_Shared_Image, 846
 as_svg_image
 Fl_RGB_Image, 819
 Fl_SVG_Image, 891
 as_window
 Fl_Widget, 1170
 Fl_Window, 1219
 atclose
 Windows handling functions, 237
 autosize
 Fl_Chart, 474
 awake
 Multithreading support functions, 313

b
 b
 Fl_Color_Chooser, 504
 Fl_Rect, 810
 background
 Fl, 384
 background2
 Fl, 385
 bbox
 Fl_Browser_, 447
 Fl_Scroll, 831
 begin
 Fl_Group, 589
 begin_job
 Fl_Paged_Device, 750
 Fl_PostScript_File_Device, 770, 771
 Fl_Printer, 799
 begin_page
 Fl_Paged_Device, 750
 Fl_PostScript_File_Device, 771
 Fl_Printer, 800
 belowmouse
 Events handling functions, 241
 bitmap
 Fl_FormsBitmap, 559
 border
 Fl_Window, 1219
 bottomline
 Fl_Browser, 419
 bound_key_function
 Fl_Text_Editor, 1014
 bounds
 Fl_Chart, 475
 Fl_Group, 589
 Fl_Slider, 868
 Fl_Valuator, 1142
 box
 Fl_Widget, 1171
 box_border_radius_max
 Fl, 385
 box_color
 Fl, 385
 box_dh
 Fl, 386
 box_dw
 Fl, 386
 box_dx
 Fl, 386
 box_dy
 Fl, 386
 box_shadow_width
 Fl, 387
 buffer
 Fl_Text_Display, 972, 973
 buffer_modified_cb
 Fl_Text_Display, 973
 buffer_predelete_cb
 Fl_Text_Display, 974
 byte_at

FI_Text_Buffer, 946
CORE_READ_OK
 FI_Preferences, 795
CORE_WRITE_OK
 FI_Preferences, 796
Cairo Support Functions and Classes, 319
 cairo_autolink_context, 319
 cairo_cc, 320
 cairo_make_current, 320
cairo_autolink_context
 Cairo Support Functions and Classes, 319
cairo_cc
 Cairo Support Functions and Classes, 320
cairo_make_current
 Cairo Support Functions and Classes, 320
calc_dimensions
 FI_Tree, 1060
calc_item_height
 FI_Tree_Item, 1108
calc_last_char
 FI_Text_Display, 974
calc_line_starts
 FI_Text_Display, 974
calc_tree
 FI_Tree, 1060
callback
 FI_Menu_Item, 710, 711
 FI_Table, 913
 FI_Widget, 1172, 1173
Callback function typedefs, 229
 FI_Event_Dispatch, 230
callback_col
 FI_Table, 914
callback_context
 FI_Table, 914
callback_item
 FI_Tree, 1060, 1061
callback_reason
 FI_Tree, 1061
callback_row
 FI_Table, 914
can_do
 FI_GI_Window, 572, 573
can_do_overlay
 FI_GI_Window, 573
cc
 FI_Cairo_State, 470
changed
 FI_Input_Choice, 667
 FI_Widget, 1173
char_at
 FI_Text_Buffer, 947
check
 FI, 387
 FI_Menu_Item, 711
check_all
 FI_Check_Browser, 481
check_none
 FI_Check_Browser, 481
checkbox
 FI_Menu_Item, 711
checked
 FI_Check_Browser, 481
 FI_Menu_Item, 711
child
 FI_Group, 590
 FI_Table, 914
 FI_Tree_Item, 1108
children
 FI_Table, 915
clamp
 FI_Valuator, 1142
clear
 FI_Browser, 420
 FI_Button, 466
 FI_Check_Browser, 482
 FI_File_Icon, 548
 FI_Group, 590
 FI_Input_Choice, 667
 FI_Menu_, 685
 FI_Menu_Item, 712
 FI_Scroll, 831
 FI_Simple_Terminal, 858
 FI_Sys_Menu_Bar, 899
 FI_Table, 915
 FI_Table_Row, 929
 FI_Tree, 1061
 FI_Tree_Item_Array, 1130
clear_active
 FI_Widget, 1174
clear_border
 FI_Window, 1219
clear_changed
 FI_Input_Choice, 667
 FI_Widget, 1174
clear_children
 FI_Tree, 1062
clear_damage
 FI_Widget, 1174
clear_modal_states
 FI_Window, 1219
clear_output
 FI_Widget, 1175
clear_overlay
 FI_Menu_Window, 723
clear_rect
 FI_Text_Display, 976
clear_selection
 FI_Help_View, 610
clear_submenu
 FI_Menu_, 685
 FI_Sys_Menu_Bar, 900
clear_visible
 FI_Widget, 1175
clear_visible_focus
 FI_Widget, 1175

clear_widget_pointer
Safe widget deletion support functions, 316

client_area
Fl_Tabs, 935

clip_children
Fl_Group, 590

clipboard_contains
Selection & Clipboard functions, 257

close
Fl_EPS_File_Surface, 527
Fl_SVG_File_Surface, 886
Fl_Tree, 1062

closedeicon
Fl_Tree_Prefs, 1135

closeicon
Fl_Tree, 1063
Fl_Tree_Prefs, 1136

col
FL_CHART_ENTRY, 478

col_header
Fl_Table, 915

col_resize
Fl_Table, 916

col_resize_min
Fl_Table, 916

col_to_x
Fl_Text_Display, 976

col_width
Fl_Table, 916

col_width_all
Fl_Table, 916

color
Fl_File_Chooser, 538
Fl_Tooltip, 1042, 1043
Fl_Widget, 1175, 1176

Color & Font functions, 267

fl_color, 268, 269
fl_color_average, 269
fl_contrast, 270
fl_font, 270
fl_height, 270, 271
fl_latin1_to_local, 271
fl_local_to_latin1, 271
fl_local_to_mac_roman, 272
fl_mac_roman_to_local, 272
fl_show_colormap, 273
fl_size, 273
fl_text_extents, 274
fl_width, 274, 275
free_color, 275
get_color, 275
get_font, 276
get_font_name, 276
get_font_sizes, 276
set_color, 276, 277
set_font, 277
set_fonts, 277

color2
Fl_Widget, 1177

color_average
Fl_Image, 624
Fl_Pixmap, 755
Fl_RGB_Image, 819
Fl_SVG_Image, 891
Fl_Shared_Image, 846
Fl_Tiled_Image, 1034

column_char
Fl_Browser, 420

column_widths
Fl_Browser, 420, 421

Common Dialogs classes and functions, 344

error, 358
fatal, 358
fl_alert, 345
fl_ask, 346
fl_beep, 346
fl_choice, 347
fl_color_chooser, 348, 349
fl_dir_chooser, 350
fl_file_chooser, 351
fl_file_chooser_callback, 352
fl_file_chooser_ok_label, 352
fl_input, 352
fl_message, 353
fl_message_hotspot, 353, 354
fl_message_icon, 354
fl_message_position, 354, 355
fl_message_title, 356
fl_message_title_default, 356
fl_password, 357
warning, 358

compare
Fl_Shared_Image, 847

compose
Events handling functions, 241

compose_reset
Events handling functions, 242

connectorstyle
Fl_Tree, 1064

contains
Fl_Widget, 1177

context
Fl_GL_Window, 573

context_valid
Fl_GL_Window, 574

copy
Fl_Bitmap, 408
Fl_Image, 624, 625
Fl_Input_, 645
Fl_Menu_, 686
Fl_Pixmap, 755
Fl_RGB_Image, 819
Fl_SVG_Image, 891
Fl_Shared_Image, 848
Fl_Text_Buffer, 947
Fl_Tiled_Image, 1034

Selection & Clipboard functions, 257
copy_cuts
 Fl_Input_, 646
copy_label
 Fl_Widget, 1178
copy_tooltip
 Fl_Widget, 1178
count
 Fl_File_Chooser, 539
 Fl_Help_Font_Stack, 604
 Fl_Image, 625
 Fl_Native_File_Chooser, 735
count_displayed_characters
 Fl_Text_Buffer, 948
count_lines
 Fl_Text_Buffer, 948
 Fl_Text_Display, 976
create_window_menu
 Fl_Sys_Menu_Bar, 900
current
 Fl_Group, 591
 Fl_Tooltip, 1043
 Fl_Window, 1220
current_
 Fl_Window, 1239
current_style_index
 Fl_Simple_Terminal, 858
cursor
 Fl_Window, 1220, 1221
cursor2rowcol
 Fl_Table, 916
cursor_color
 Fl_Input_, 646
 Fl_Text_Display, 977
 Fl_Value_Input, 1147, 1148
cursor_style
 Fl_Text_Display, 977
custom_application_menu_items
 Fl_Mac_App_Menu, 678
cut
 Fl_Input_, 647
d
 Fl_Image, 625
damage
 Fl, 388
 Fl_Widget, 1178, 1179
damage_resize
 Fl_Widget, 1180
damage_zone
 Fl_Table, 917
data
 Fl_Browser, 421, 422
 Fl_Image, 625
deactivate
 Fl_Menu_Item, 712
 Fl_Tree_Item, 1109
 Fl_Widget, 1180
decorated_h
 Fl_Window, 1221
decorated_w
 Fl_Window, 1222
default_atclose
 Windows handling functions, 235
default_callback
 Fl_Widget, 1180
default_cursor
 Fl_Window, 1222
default_icon
 Fl_Window, 1222
default_icons
 Fl_Window, 1223
default_key_function
 Fl_Text_Editor, 1014
default_xclass
 Fl_Window, 1223, 1224
deimage
 Fl_Widget, 1181, 1182
delay
 Fl_Tooltip, 1043
delete_widget
 Safe widget deletion support functions, 316
deleteEntry
 Fl_Preferences, 783
deleteGroup
 Fl_Preferences, 784
deleted
 Fl_Widget_Tracker, 1212
deleting
 Fl_Browser_, 447
deparent
 Fl_Tree_Item, 1109
 Fl_Tree_Item_Array, 1130
depth
 Fl_Tree_Item, 1109
desaturate
 Fl_Image, 625
 Fl_Pixmap, 755
 Fl_RGB_Image, 820
 Fl_SVG_Image, 892
 Fl_Shared_Image, 848
 Fl_Tiled_Image, 1035
deselect
 Fl_Browser_, 447
 Fl_Tree, 1064
deselect_all
 Fl_Tree, 1065
 Fl_Tree_Item, 1109
direction
 Fl_Timer, 1037
directory
 Fl_File_Chooser, 539
 Fl_Help_View, 610
 Fl_Native_File_Chooser, 735
disable
 Fl_Tooltip, 1043
disable_im

Events handling functions, 242

display
 FI, 388
 FI_Browser, 422
 FI_Browser_, 448
 FI_Tree, 1066

display_insert
 FI_Text_Display, 978

displayed
 FI_Browser, 422
 FI_Browser_, 448
 FI_Tree, 1066

dnd
 Selection & Clipboard functions, 257

dnd_text_ops
 FI, 388

do_callback
 FI_Menu_Item, 712
 FI_Table, 917
 FI_Widget, 1182, 1183

do_widget_deletion
 Safe widget deletion support functions, 316

down_box
 FI_Button, 466
 FI_File_Input, 553, 554
 FI_Menu_, 686

draw
 FI_Adjuster, 405
 FI_Bitmap, 408
 FI_Box, 413
 FI_Button, 467
 FI_Chart, 475
 FI_Choice, 491
 FI_Clock_Output, 499
 FI_Counter, 514
 FI_Dial, 520
 FI_File_Icon, 548
 FI_FormsBitmap, 560
 FI_FormsPixmap, 562
 FI_FormsText, 563
 FI_Free, 565
 FI_GI_Window, 574
 FI_Glut_Window, 583
 FI_Group, 591
 FI_Help_View, 610
 FI_Image, 625, 626
 FI_Input, 640
 FI_Label, 673
 FI_Light_Button, 676
 FI_Menu_Bar, 699
 FI_Menu_Button, 702
 FI_Menu_Item, 713
 FI_Pack, 746
 FI_Pixmap, 756
 FI_Positioner, 765
 FI_Progress, 805
 FI_RGB_Image, 820
 FI_Return_Button, 815

FI_Roller, 824
 FI_SVG_Image, 892
 FI_Scroll, 831
 FI_Scrollbar, 837
 FI_Shared_Image, 848
 FI_Simple_Terminal, 859
 FI_Slider, 868
 FI_Sys_Menu_Bar, 900
 FI_Table, 917
 FI_Tabs, 936
 FI_Text_Display, 978
 FI_Tiled_Image, 1035
 FI_Timer, 1038
 FI_Tree_Item, 1109
 FI_Value_Input, 1148
 FI_Value_Output, 1153
 FI_Value_Slider, 1157
 FI_Widget, 1183
 FI_Widget_Surface, 1208
 FI_Window, 1224
 FI_Wizard, 1241

draw_GL_text_with_textures
 FI, 389

draw_box
 FI_Widget, 1184

draw_box_active
 FI, 388

draw_cell
 FI_Table, 917

draw_child
 FI_Group, 591

draw_children
 FI_Group, 592

draw_cursor
 FI_Text_Display, 978

draw_decorated_window
 FI_Copy_Surface, 510
 FI_Printer, 800
 FI_Widget_Surface, 1208

draw_empty
 FI_Image, 626

draw_focus
 FI_Widget, 1184

draw_horizontal_connector
 FI_Tree_Item, 1110

draw_item_content
 FI_Tree_Item, 1110

draw_label
 FI_Widget, 1184, 1185

draw_line_numbers
 FI_Text_Display, 979

draw_outside_label
 FI_Group, 592

draw_overlay
 FI_Glut_Window, 584
 FI_Overlay_Window, 743

draw_range
 FI_Text_Display, 979

draw_scaled
 Fl_Image, 626
draw_string
 Fl_Text_Display, 979
draw_text
 Fl_Text_Display, 980
draw_vertical_connector
 Fl_Tree_Item, 1112
draw_vline
 Fl_Text_Display, 980
drawbgcolor
 Fl_Tree_Item, 1112
drawfgcolor
 Fl_Tree_Item, 1112
Drawing functions, 278
 fl_add_symbol, 283
 fl_arc, 283, 284
 fl_begin_complex_polygon, 285
 fl_begin_offscreen, 285
 fl_begin_points, 286
 fl_can_do_alpha_blending, 286
 fl_capture_window_part, 286
 fl_circle, 287
 fl_clip, 282
 fl_clip_box, 287
 fl_clip_region, 288, 289
 fl_copy_offscreen, 289
 fl_create_offscreen, 289
 fl_cursor, 290
 fl_curve, 290
 fl_delete_offscreen, 291
 fl_draw, 291, 292
 fl_draw_box, 292
 fl_draw_image, 293
 fl_draw_image_mono, 294
 fl_draw_pixmap, 295
 fl_draw_symbol, 296
 fl_expand_text, 296
 fl_frame, 296
 fl_frame2, 297
 fl_gap, 297
 fl_line_style, 297
 fl_measure, 298
 fl_measure_pixmap, 299
 fl_mult_matrix, 299
 fl_not_clipped, 300
 fl_old_shortcut, 300
 fl_pie, 301
 fl_polygon, 302
 fl_pop_clip, 303
 fl_push_clip, 303
 fl_push_matrix, 303
 fl_read_image, 303
 fl_rect, 304
 fl_rectf, 304
 fl_rescale_offscreen, 305
 fl_reset_spot, 305
 fl_rotate, 305
 fl_scale, 306
 fl_scroll, 306
 fl_set_spot, 307
 fl_set_status, 307
 fl_shortcut_label, 307, 309
 fl_transform_dx, 309
 fl_transform_dy, 309
 fl_transform_x, 311
 fl_transform_y, 311
 fl_transformed_vertex, 311
 fl_translate, 311
 fl_vertex, 312
drawtext
 Fl_Input_, 648
driver
 Fl_EPS_File_Surface, 527
 Fl_Surface_Device, 882
ERRORS_TO_CP1252
 Unicode and UTF-8 functions, 323
ERRORS_TO_ISO8859_1
 Unicode and UTF-8 functions, 323
empty_vlines
 Fl_Text_Display, 981
enable
 Fl_Tooltip, 1043
enable_im
 Events handling functions, 242
enabled
 Fl_Tooltip, 1044
end
 Fl_Group, 592
 Fl_Text_Selection, 1025
end_page
 Fl_Paged_Device, 750
 Fl_PostScript_File_Device, 771
 Fl_Printer, 800
enforce_history_lines
 Fl_Simple_Terminal, 859
enforce_stay_at_bottom
 Fl_Simple_Terminal, 859
enter_area
 Fl_Tooltip, 1044
entries
 Fl_Preferences, 784
entry
 Fl_Preferences, 784
entryExists
 Fl_Preferences, 785
Enumerations.H, 1251
 FL_ABI_VERSION, 1263
 FL_ALIGN_BOTTOM, 1277
 FL_ALIGN_CENTER, 1277
 FL_ALIGN_CLIP, 1277
 FL_ALIGN_IMAGE_BACKDROP, 1277
 FL_ALIGN_IMAGE_MASK, 1277
 FL_ALIGN_IMAGE_NEXT_TO_TEXT, 1277
 FL_ALIGN_IMAGE_OVER_TEXT, 1278
 FL_ALIGN_INSIDE, 1278

FL_ALIGN_LEFT_BOTTOM, 1278
 FL_ALIGN_LEFT_TOP, 1278
 FL_ALIGN_LEFT, 1278
 FL_ALIGN_NOWRAP, 1278
 FL_ALIGN_POSITION_MASK, 1279
 FL_ALIGN_RIGHT_BOTTOM, 1279
 FL_ALIGN_RIGHT_TOP, 1279
 FL_ALIGN_RIGHT, 1279
 FL_ALIGN_TEXT_NEXT_TO_IMAGE, 1279
 FL_ALIGN_TEXT_OVER_IMAGE, 1279
 FL_ALIGN_TOP, 1280
 FL_ALIGN_WRAP, 1280
 FL_API_VERSION, 1263
 FL_IMAGE_LABEL, 1263
 FL_MAJOR_VERSION, 1264
 FL_MINOR_VERSION, 1264
 FL_MULTI_LABEL, 1264
 FL_NORMAL_SIZE, 1280
 FL_PATCH_VERSION, 1264
 FL_SYMBOL_LABEL, 1264
 FL_VERSION, 1265
 Fl_Align, 1265
 Fl_Boxtype, 1266
 Fl_Cursor, 1268
 Fl_Damage, 1269
 Fl_Event, 1269
 Fl_Font, 1265
 Fl_Fontsize, 1266
 Fl_Labeltype, 1272
 Fl_When, 1273
 fl_box, 1274
 fl_color_cube, 1274
 fl_darker, 1274
 fl_define_FL_EMBOSSDED_LABEL, 1274
 fl_define_FL_ENGRAVED_LABEL, 1274
 fl_define_FL_ICON_LABEL, 1275
 fl_define_FL_IMAGE_LABEL, 1275
 fl_define_FL_MULTI_LABEL, 1275
 fl_define_FL_SHADOW_LABEL, 1275
 fl_down, 1275
 fl_frame, 1275
 fl_gray_ramp, 1276
 fl_lighter, 1276
 fl_rgb_color, 1276
 errmsg
 Fl_File_Browser, 530
 Fl_Native_File_Chooser, 736
 error
 Common Dialogs classes and functions, 358
 errorcolor
 Fl_File_Input, 554
 event
 Events handling functions, 243
 event_alt
 Events handling functions, 243
 event_button
 Events handling functions, 243
 event_button1
 Events handling functions, 243
 event_button2
 Events handling functions, 243
 event_button3
 Events handling functions, 243
 event_buttons
 Events handling functions, 244
 event_clicks
 Events handling functions, 244, 245
 event_clipboard
 Events handling functions, 245
 event_clipboard_type
 Events handling functions, 245
 event_command
 Events handling functions, 245
 event_ctrl
 Events handling functions, 245
 event_dispatch
 Events handling functions, 246
 event_dx
 Events handling functions, 246
 event_dy
 Events handling functions, 246
 event_inside
 Events handling functions, 247
 event_is_click
 Events handling functions, 248
 event_key
 Events handling functions, 248
 event_length
 Events handling functions, 249
 event_original_key
 Events handling functions, 249
 event_shift
 Events handling functions, 249
 event_state
 Events handling functions, 250
 event_text
 Events handling functions, 250
 event_x_root
 Events handling functions, 251
 event_y_root
 Events handling functions, 251
 Events handling functions, 238
 add_handler, 240
 add_system_handler, 240
 belowmouse, 241
 compose, 241
 compose_reset, 242
 disable_im, 242
 enable_im, 242
 event, 243
 event_alt, 243
 event_button, 243
 event_button1, 243
 event_button2, 244
 event_button3, 244
 event_buttons, 244

event_clicks, 244, 245
event_clipboard, 245
event_clipboard_type, 245
event_command, 245
event_ctrl, 245
event_dispatch, 246
event_dx, 246
event_dy, 246
event_inside, 247
event_is_click, 248
event_key, 248
event_length, 249
event_original_key, 249
event_shift, 249
event_state, 250
event_text, 250
event_x_root, 251
event_y_root, 251
fl_eventnames, 255
fl_fontnames, 255
focus, 251
get_key, 252
get_mouse, 252
handle, 252
handle_, 253
pushed, 253
remove_handler, 254
remove_system_handler, 254
test_shortcut, 254
exists
 FI_Widget_Tracker, 1212
extend_range_for_styles
 FI_Text_Display, 981
extend_selection
 FI_Tree, 1066
extend_selection_dir
 FI_Tree, 1067

FL_ABI_VERSION
 Enumerations.H, 1263
FL_ALIGN_BOTTOM
 Enumerations.H, 1277
FL_ALIGN_CENTER
 Enumerations.H, 1277
FL_ALIGN_CLIP
 Enumerations.H, 1277
FL_ALIGN_IMAGE_BACKDROP
 Enumerations.H, 1277
FL_ALIGN_IMAGE_MASK
 Enumerations.H, 1277
FL_ALIGN_IMAGE_NEXT_TO_TEXT
 Enumerations.H, 1277
FL_ALIGN_IMAGE_OVER_TEXT
 Enumerations.H, 1278
FL_ALIGN_INSIDE
 Enumerations.H, 1278
FL_ALIGN_LEFT_BOTTOM
 Enumerations.H, 1278
FL_ALIGN_LEFT_TOP
 Enumerations.H, 1278
FL_ALIGN_LEFT
 Enumerations.H, 1278
FL_ALIGN_NOWRAP
 Enumerations.H, 1278
FL_ALIGN_POSITION_MASK
 Enumerations.H, 1279
FL_ALIGN_RIGHT_BOTTOM
 Enumerations.H, 1279
FL_ALIGN_RIGHT_TOP
 Enumerations.H, 1279
FL_ALIGN_RIGHT
 Enumerations.H, 1279
FL_ALIGN_TEXT_NEXT_TO_IMAGE
 Enumerations.H, 1279
FL_ALIGN_TEXT_OVER_IMAGE
 Enumerations.H, 1279
FL_ALIGN_TOP
 Enumerations.H, 1280
FL_ALIGN_WRAP
 Enumerations.H, 1280
FL_API_VERSION
 Enumerations.H, 1263
FL_CHART_ENTRY, 477
 col, 478
 str, 478
 val, 478
FL_IMAGE_LABEL
 Enumerations.H, 1263
FL_MAJOR_VERSION
 Enumerations.H, 1264
FL_MINOR_VERSION
 Enumerations.H, 1264
FL_MULTI_LABEL
 Enumerations.H, 1264
FL_NORMAL_SIZE
 Enumerations.H, 1280
FL_PATCH_VERSION
 Enumerations.H, 1264
FL_RESERVED_TYPE
 FI_Widget.H, 1314
FL_SYMBOL_LABEL
 Enumerations.H, 1264
FL_VERSION
 Enumerations.H, 1265
fail
 FI_Image, 627
fatal
 Common Dialogs classes and functions, 358
File names and URI utility functions, 359
 FI_File_Sort_F, 360
 fl_decode_uri, 360
 fl_filename_absolute, 360
 fl_filename_expand, 361
 fl_filename_ext, 361
 fl_filename_free_list, 362
 fl_filename_isdir, 362
 fl_filename_list, 362

fl_filename_match, 363
 fl_filename_name, 364
 fl_filename_relative, 364
 fl_filename_setext, 365
 fl_open_uri, 365
file_access
 FI_Preferences, 785, 786
file_encoding_warning_message
 FI_Text_Buffer, 961
filename
 FI_Help_View, 610
 FI_Native_File_Chooser, 736
filename.H, 1280
filetype
 FI_File_Browser, 530, 531
filter
 FI_File_Browser, 531
 FI_File_Chooser, 539
 FI_Native_File_Chooser, 736
filter_value
 FI_File_Chooser, 540
 FI_Native_File_Chooser, 737
find
 FI_File_Icon, 548
 FI_Group, 592
 FI_Help_View, 610
 FI_Shared_Image, 849
find_cell
 FI_Table, 919
find_child
 FI_Tree_Item, 1113
find_child_item
 FI_Tree_Item, 1113, 1114
find_clicked
 FI_Tree, 1068
 FI_Tree_Item, 1114
find_index
 FI_Menu_, 686, 687
find_item
 FI_Browser_, 448
 FI_Menu_, 688
 FI_Tree, 1068
 FI_Tree_Item, 1115
find_line
 FI_Browser, 423
find_line_end
 FI_Text_Display, 981
find_shortcut
 FI_Menu_Item, 713
find_wrap_range
 FI_Text_Display, 982
find_x
 FI_Text_Display, 982
findchar_backward
 FI_Text_Buffer, 948
findchar_forward
 FI_Text_Buffer, 949
first
 FI_File_Icon, 548
 FI_Menu_Item, 713, 714
 FI_Tree, 1069
first_selected_item
 FI_Tree, 1069
first_visible
 FI_Tree, 1069
first_visible_item
 FI_Tree, 1070
first_window
 Windows handling functions, 235, 236
FI, 369
 abi_check, 380
 abi_version, 380
 add_awake_handler_, 380
 add_check, 380
 add_fd, 381
 add_idle, 382
 add_timeout, 382
 api_version, 382
 arg, 383
 args, 383, 384
 background, 384
 background2, 385
 box_border_radius_max, 385
 box_color, 385
 box_dh, 386
 box_dw, 386
 box_dx, 386
 box_dy, 386
 box_shadow_width, 387
 check, 387
 damage, 388
 display, 388
 dnd_text_ops, 388
 draw_GL_text_with_textures, 389
 draw_box_active, 388
 FI_Option, 379
 flush, 389
 foreground, 390
 get_awake_handler_, 390
 get_boxtype, 390
 get_system_colors, 390
 gl_visual, 390
 help, 403
 idle, 403
 insertion_point_location, 391
 is_scheme, 391
 menu_linespacing, 392
 option, 392, 393
 own_colormap, 394
 program_should_quit, 394
 readqueue, 394
 ready, 395
 release, 395
 reload_scheme, 395
 remove_check, 396
 remove_fd, 396

remove_timeout, 396
repeat_timeout, 396
reset_marked_text, 397
run, 397
scheme, 397
scrollbar_size, 398
set_box_color, 399
set_boxtype, 399
set_idle, 399
set_labeltype, 400
use_high_res_GL, 400
version, 401
visible_focus, 401
visual, 401
wait, 402
Fl.cxx, 1281
 fl_open_display, 1282
Fl.H, 1283
Fl_Adjuster, 404
 draw, 405
 Fl_Adjuster, 405
 handle, 405
 soft, 406
Fl_Align
 Enumerations.H, 1265
Fl_BMP_Image, 410
 Fl_BMP_Image, 410, 411
Fl_Beep
 fl_ask.H, 1287
Fl_Bitmap, 406
 copy, 408
 draw, 408
 Fl_Bitmap, 407, 408
 label, 409
 uncache, 409
Fl_Box, 411
 draw, 413
 Fl_Box, 412
 handle, 413
Fl_Boxtype
 Enumerations.H, 1266
Fl_Browser, 414
 _remove, 418
 add, 419
 bottomline, 419
 clear, 420
 column_char, 420
 column_widths, 420, 421
 data, 421, 422
 display, 422
 displayed, 422
 find_line, 423
 Fl_Browser, 418
 format_char, 423, 424
 full_height, 424
 hide, 425
 icon, 425, 426
 incr_height, 426
 insert, 426, 427
 item_at, 427
 item_draw, 428
 item_first, 428
 item_height, 429
 item_last, 429
 item_next, 430
 item_prev, 430
 item_select, 431
 item_selected, 431
 item_swap, 431
 item_text, 432
 item_width, 432
 lineno, 433
 lineposition, 433
 load, 434
 make_visible, 434
 middleline, 435
 move, 435
 remove, 435
 remove_icon, 436
 replace, 436
 select, 436
 selected, 437
 show, 437
 size, 438
 swap, 438
 text, 439
 textsize, 439
 topline, 440
 value, 440, 441
 visible, 441
Fl_Browser_, 442
 bbox, 447
 deleting, 447
 deselect, 447
 display, 448
 displayed, 448
 find_item, 448
 Fl_Browser_, 446
 full_height, 449
 full_width, 449
 handle, 449
 has_scrollbar, 450
 hposition, 450
 hscrollbar, 463
 incr_height, 451
 inserting, 451
 item_at, 451
 item_first, 453
 item_height, 453
 item_last, 453
 item_next, 454
 item_prev, 454
 item_quick_height, 454
 item_select, 455
 item_selected, 455
 item_swap, 455

item_text, 456
 item_width, 456
 leftedge, 457
 new_list, 457
 position, 457
 redraw_line, 458
 redraw_lines, 458
 replacing, 458
 resize, 459
 scrollbar, 463
 scrollbar_left, 459
 scrollbar_right, 459
 scrollbar_size, 459
 scrollbar_width, 460
 select, 460
 select_only, 461
 selection, 461
 sort, 461
 swapping, 462
 textfont, 462
Fl_Button, 463
 clear, 466
 down_box, 466
 draw, 467
Fl_Button, 465
 handle, 467
 set, 468
 shortcut, 468
 value, 469
Fl_Cairo_State, 469
 cc, 470
Fl_Cairo_Window, 470
 set_draw_cb, 471
Fl_Chart, 472
 add, 474
 autosize, 474
 bounds, 475
 draw, 475
Fl_Chart, 473
 insert, 475
 maxsize, 476
 replace, 476
 textcolor, 477
 textfont, 477
 textsize, 477
Fl_Check_Browser, 478
 \sim **Fl_Check_Browser**, 480
 add, 481
 check_all, 481
 check_none, 481
 checked, 481
 clear, 482
Fl_Check_Browser, 480
 handle, 482
 item_at, 482
 item_first, 483
 item_height, 483
 item_next, 483
 item_prev, 484
 item_select, 484
 item_selected, 484
 item_swap, 485
 item_text, 485
 item_width, 486
 nchecked, 486
 nitems, 486
 remove, 486
 set_checked, 486
 text, 487
 value, 487
Fl_Check_Button, 487
Fl_Check_Button, 488
Fl.Choice, 489
 draw, 491
Fl.Choice, 491
 handle, 491
 value, 492, 493
Fl.Clock, 493
Fl.Clock, 495
 handle, 496
Fl.Clock_Output, 496
 draw, 499
Fl.Clock_Output, 498
 hour, 499
 minute, 499
 second, 499
 shadow, 500
 value, 500, 501
Fl.Color_Chooser, 502
 b, 504
Fl.Color_Chooser, 504
 g, 504
 handle, 504
 hsv, 505
 hsv2rgb, 506
 hue, 506
 mode, 506
 r, 507
 rgb, 507
 rgb2hsv, 507
 saturation, 508
 value, 508
Fl.Color_Chooser.H, 1291
Fl.Copy_Surface, 508
 draw_decorated_window, 510
Fl.Copy_Surface, 510
 origin, 510, 511
 printable_rect, 511
 set_current, 511
 translate, 512
Fl.Counter, 512
 draw, 514
Fl.Counter, 514
 handle, 515
 lstep, 515
 step, 516

Fl_Cursor
 Enumerations.H, 1268

Fl_Damage
 Enumerations.H, 1269

Fl_Device.H, 1292

Fl_Device_Plugin, 516
 rectangle_capture, 517

Fl_Dial, 518
 angle1, 520
 draw, 520
 Fl_Dial, 519
 handle, 520

Fl_Display_Device, 521

Fl_Double_Window, 521
 ~Fl_Double_Window, 522
 flush, 523
 hide, 523
 resize, 523
 show, 523

Fl_Double_Windowcxx, 1292

Fl_EPS_File_Surface, 525
 ~Fl_EPS_File_Surface, 526
 close, 527
 driver, 527
 Fl_EPS_File_Surface, 526
 origin, 527
 printable_rect, 528
 translate, 528

Fl_End, 524

Fl_Event
 Enumerations.H, 1269

Fl_Event_Dispatch
 Callback function typedefs, 230

Fl_File_Browser, 529
 errmsg, 530
 filetype, 530, 531
 filter, 531
 Fl_File_Browser, 530
 iconsize, 531
 load, 531

Fl_File_Chooser, 532
 ~Fl_File_Chooser, 537
 add_extra, 538
 color, 538
 count, 539
 directory, 539
 filter, 539
 filter_value, 540
 Fl_File_Chooser, 537
 hide, 540
 iconsize, 540
 label, 540, 541
 preview, 541
 rescan, 541
 show, 541
 showHiddenButton, 544
 shown, 541
 textcolor, 542
 textfont, 542
 textsize, 542, 543
 type, 543
 user_data, 543
 value, 543
 visible, 544

Fl_File_Icon, 544
 add, 546
 add_color, 547
 add_vertex, 547
 clear, 548
 draw, 548
 find, 548
 first, 548
 Fl_File_Icon, 546
 label, 549
 labeltype, 549
 load, 549
 load_fti, 550
 load_image, 550
 load_system_icons, 550
 next, 550
 pattern, 551
 size, 551
 type, 551
 value, 551

Fl_File_Input, 552
 down_box, 553, 554
 errorcolor, 554
 Fl_File_Input, 553
 handle, 554
 value, 555

Fl_File_Sort_F
 File names and URI utility functions, 360

Fl_Fill_Dial, 555
 Fl_Fill_Dial, 556

Fl_Fill_Slider, 557
 Fl_Fill_Slider, 557

Fl_Float_Input, 558
 Fl_Float_Input, 558

Fl_Font
 Enumerations.H, 1265

Fl_Fontsize
 Enumerations.H, 1266

Fl_FormsBitmap, 559
 bitmap, 559
 draw, 560
 set, 560

Fl_FormsPixmap, 560
 draw, 562
 Fl_FormsPixmap, 561
 Pixmap, 562
 set, 562

Fl_FormsText, 563
 draw, 563

Fl_Free, 564
 draw, 565

Fl_Free, 565

handle, 566
Fl_GIF_Image, 567
 Fl_GIF_Image, 567, 568
Fl_Gl_Choice, 568
Fl_Gl_Window, 569
 as_gl_window, 572
 can_do, 572, 573
 can_do_overlay, 573
 context, 573
 context_valid, 574
 draw, 574
Fl_Gl_Window, 571, 572
 flush, 575
 hide_overlay, 575
 make_current, 575
 make_overlay_current, 575
 mode, 575, 576
 ortho, 577
 pixel_h, 577
 pixel_w, 577
 pixels_per_unit, 577
 redraw_overlay, 578
 resize, 578
 show, 578
 swap_buffers, 579
 valid, 579
Fl_Glut_Bitmap_Font, 580
Fl_Glut_StrokeChar, 580
Fl_Glut_StrokeFont, 580
Fl_Glut_StrokeStrip, 581
Fl_Glut_StrokeVertex, 581
Fl_Glut_Window, 581
 draw, 583
 draw_overlay, 584
Fl_Glut_Window, 582, 583
Fl_Group, 584
 ~Fl_Group, 587
 array, 588
 as_group, 588
 begin, 589
 bounds, 589
 child, 590
 clear, 590
 clip_children, 590
 current, 591
 draw, 591
 draw_child, 591
 draw_children, 592
 draw_outside_label, 592
 end, 592
 find, 592
Fl_Group, 587
 focus, 592
 handle, 593
 init_sizes, 593
 insert, 594
 remove, 594, 595
 resizable, 595, 596
 resize, 597
 sizes, 597
 update_child, 598
Fl_Help_Block, 598
Fl_Help_Dialog, 599
 Fl_Help_Dialog, 600
 h, 600
 hide, 600
 load, 601
 position, 601
 resize, 601
 show, 601
 textszie, 602
 value, 602
 visible, 602
 w, 602
 x, 603
 y, 603
Fl_Help_Font_Stack, 603
 count, 604
Fl_Help_Font_Stack, 604
 top, 604
Fl_Help_Font_Style, 604
Fl_Help_Link, 605
Fl_Help_Target, 606
Fl_Help_View, 606
 ~Fl_Help_View, 610
 clear_selection, 610
 directory, 610
 draw, 610
 filename, 610
 find, 610
 handle, 611
 leftline, 611
 link, 611
 load, 612
 resize, 612
 scrollbar_size, 613
 select_all, 614
 size, 614
 textcolor, 614
 textfont, 614
 textszie, 615
 title, 615
 topline, 615, 616
 value, 616
Fl_Hold_Browser, 617
 Fl_Hold_Browser, 617
Fl_Hor_Fill_Slider, 618
Fl_Hor_Nice_Slider, 619
Fl_Hor_Slider, 619
Fl_Hor_Value_Slider, 620
Fl_Image, 621
 as_shared_image, 624
 color_average, 624
 copy, 624, 625
 count, 625
 d, 625

data, 625
desaturate, 625
draw, 625, 626
draw_empty, 626
draw_scaled, 626
fail, 627
Fl_Image, 623
h, 627, 628
inactive, 628
label, 628
Id, 628, 629
RGB_scaling, 629, 630
release, 629
scale, 630
scaling_algorithm, 631
uncache, 631
w, 631
Fl_Image.H, 1298
Fl_RGB_Scaling, 1299
Fl_Image_Reader, 632
Fl_Image_Surface, 632
~Fl_Image_Surface, 634
Fl_Image_Surface, 634
highres_image, 635
image, 635
offscreen, 635
origin, 635, 636
printable_rect, 636
rescale, 636
set_current, 637
translate, 637
Fl_Input, 638
draw, 640
Fl_Input, 640
handle, 640
Fl_Input_, 641
~Fl_Input_, 645
append, 645
copy, 645
copy_cuts, 646
cursor_color, 646
cut, 647
drawtext, 648
Fl_Input_, 644
handle_mouse, 648
handletext, 648
index, 649
input_type, 649
insert, 650
line_end, 650
line_start, 650
mark, 651
maximum_size, 652
position, 652, 653
readonly, 653
replace, 654
resize, 655
shortcut, 655
size, 656
static_value, 656, 657
tab_nav, 657, 658
textcolor, 658, 659
textfont, 659
textsize, 660
undo, 660
up_down_position, 660
value, 661, 662
word_end, 662
word_start, 663
wrap, 663
Fl_Input_Choice, 664
add, 666
changed, 667
clear, 667
clear_changed, 667
Fl_Input_Choice, 666
input, 667
menu, 667, 668
menubutton, 668
update_menubutton, 668
value, 669
Fl_Int_Input, 670
Fl_Int_Input, 670
Fl_JPEG_Image, 671
Fl_JPEG_Image, 671, 672
Fl_Label, 672
draw, 673
measure, 673
type, 674
Fl_Labeltype
Enumerations.H, 1272
Fl_Light_Button, 674
draw, 676
Fl_Light_Button, 675
handle, 676
Fl_Line_Dial, 677
Fl_Mac_App_Menu, 677
custom_application_menu_items, 678
print, 678
Fl_Menu_, 679
add, 682, 684
clear, 685
clear_submenu, 685
copy, 686
down_box, 686
find_index, 686, 687
find_item, 688
Fl_Menu_, 682
global, 689
insert, 689
item_pathname, 690
menu, 690, 691
menu_end, 691
mode, 692
mvalue, 692
picked, 693

remove, 693
 replace, 693
 setonly, 694
 shortcut, 694
 size, 694
 test_shortcut, 694
 text, 694, 695
 textcolor, 695
 textfont, 695
 textsize, 695, 696
 value, 696
Fl_Menu_Bar, 697
 draw, 699
Fl_Menu_Bar, 698
 handle, 699
 update, 700
Fl_Menu_Button, 700
 draw, 702
Fl_Menu_Button, 702
 handle, 703
 popup, 703
 popup_buttons, 702
Fl_Menu_Item, 704
 activate, 708
 active, 708
 activevisible, 709
 add, 709
 argument, 709
 callback, 710, 711
 check, 711
 checkbox, 711
 checked, 711
 clear, 712
 deactivate, 712
 do_callback, 712
 draw, 713
 find_shortcut, 713
 first, 713, 714
 hide, 714
 image, 714
 insert, 714
 label, 715
 labelcolor, 715
 labelfont, 715, 716
 labelsize, 716
 labeltype, 716
 measure, 717
 next, 717
 popup, 717
 pulldown, 718
 radio, 718
 set, 718
 setonly, 718
 shortcut, 719
 show, 719
 size, 719
 submenu, 720
 test_shortcut, 720
 uncheck, 720
 value, 720
 visible, 721
Fl_Menu_Item.H, 1299
Fl_Menu_Window, 721
 ~**Fl_Menu_Window**, 722
 clear_overlay, 723
Fl_Menu_Window, 722, 723
 flush, 723
 hide, 723
 set_overlay, 724
 show, 724
Fl_Multi_Browser, 725
Fl_Multi_Browser, 725
Fl_Multi_Label, 726
 label, 727, 728
 labela, 728
 labelb, 728
 typea, 728
 typeb, 728
Fl_Multiline_Input, 729
Fl_Multiline_Input, 730
Fl_Multiline_Output, 730
Fl_Multiline_Output, 731
Fl_Native_File_Chooser, 731
 ~**Fl_Native_File_Chooser**, 735
 count, 735
 directory, 735
 errmsg, 736
 filename, 736
 filter, 736
 filter_value, 737
Fl_Native_File_Chooser, 735
 Option, 734
 options, 737
 preset_file, 737
 show, 738
 title, 738
 Type, 734
Fl_Native_File_Chooser.H, 1300
Fl_Nice_Slider, 739
Fl_Option
 Fl, 379
Fl_Output, 739
Fl_Output, 740
Fl_Overlay_Window, 741
 draw_overlay, 743
Fl_Overlay_Window, 742
 flush, 743
 hide, 743
 redraw_overlay, 743
 resize, 743
 show, 744
Fl_PNG_Image, 760
Fl_PNG_Image, 761
Fl_PNM_Image, 762
Fl_PNM_Image, 762
Fl_Pack, 745

draw, 746
FI_Pack, 746
horizontal, 747
FI_Paged_Device, 747
begin_job, 750
begin_page, 750
end_page, 750
margins, 751
Page_Format, 749
Page_Layout, 749
rotate, 751
scale, 752
start_job, 752
start_page, 752
FI_Paged_Device.hxx, 1301
FI_Paged_Device.H, 1301
FI_Paged_Device::page_format, 1247
FI_Pixmap, 753
color_average, 755
copy, 755
desaturate, 755
draw, 756
FI_Pixmap, 754
label, 756
uncache, 756
FI_Plugin, 757
FI_Plugin, 758
FI_Plugin_Manager, 758
~FI_Plugin_Manager, 759
addPlugin, 759
load, 759
removePlugin, 760
FI_Positioner, 763
draw, 765
FI_Positioner, 764
handle, 765
value, 766
xbounds, 766
xstep, 766
xvalue, 766, 767
ybounds, 767
ystep, 767
yvalue, 767
FI_PostScript.H, 1301
FI_PostScript_File_Device, 768
begin_job, 770, 771
begin_page, 771
end_page, 771
margins, 772
origin, 772, 773
printable_rect, 773
rotate, 773
scale, 774
start_job, 774
translate, 774
FI_Preferences, 775
~FI_Preferences, 783
CORE_READ_OK, 795
CORE_WRITE_OK, 796
deleteEntry, 783
deleteGroup, 784
entries, 784
entry, 784
entryExists, 785
file_access, 785, 786
FI_Preferences, 780–783
flush, 786
get, 786–789
getUserdataPath, 790
group, 791
groupExists, 791
groups, 791
ID, 779
NONE, 796
newUUID, 792
Root, 779
set, 792–795
size, 795
FI_Preferences::Entry, 369
FI_Preferences::Name, 1245
Name, 1246
FI_Preferences::Node, 1247
FI_Preferences::RootNode, 1248
FI_Printer, 796
begin_job, 799
begin_page, 800
draw_decorated_window, 800
end_page, 800
margins, 800
origin, 801
printable_rect, 801
rotate, 802
scale, 802
set_current, 803
translate, 803
FI_Printer.H, 1302
FI_Progress, 803
draw, 805
FI_Progress, 804
maximum, 805
minimum, 805
value, 805, 806
FI_RGB_Image, 816
array, 822
as_svg_image, 819
color_average, 819
copy, 819
desaturate, 820
draw, 820
FI_RGB_Image, 818
label, 820, 821
max_size, 821
normalize, 821
uncache, 822
FI_RGB_Scaling
FI_Image.H, 1299

FI_Radio_Button, 806
 FI_Radio_Button, 806
FI_Radio_Light_Button, 807
FI_Radio_Round_Button, 807
 FI_Radio_Round_Button, 808
FI_Rect, 808
 b, 810
 FI_Rect, 809, 810
 r, 810
FI_Repeat_Button, 812
 FI_Repeat_Button, 813
 handle, 813
FI_Return_Button, 814
 draw, 815
 FI_Return_Button, 815
 handle, 815
FI_Roller, 823
 draw, 824
 FI_Roller, 824
 handle, 824
FI_Round_Button, 825
 FI_Round_Button, 826
FI_Round_Clock, 827
 FI_Round_Clock, 827
FI_SVG_File_Surface, 883
 ~FI_SVG_File_Surface, 886
 close, 886
 FI_SVG_File_Surface, 884
 origin, 886
 printable_rect, 887
 translate, 887
FI_SVG_Image, 888
 ~FI_SVG_Image, 891
 as_svg_image, 891
 color_average, 891
 copy, 891
 desaturate, 892
 draw, 892
 FI_SVG_Image, 890
 normalize, 892
 proportional, 893
 resize, 892
FI_Scroll, 828
 bbox, 831
 clear, 831
 draw, 831
 FI_Scroll, 831
 handle, 832
 recalc_scrollbars, 832
 resize, 833
 scroll_to, 833
 scrollbar_size, 834
 xposition, 835
 yposition, 835
FI_Scroll::ScrollInfo, 1248
FI_Scroll::ScrollInfo::FI_Region_LRTB, 811
FI_Scroll::ScrollInfo::FI_Region_XYWH, 811
FI_Scroll::ScrollInfo::FI_Scrollbar_Data, 839
FI_Scrollbar, 835
 ~FI_Scrollbar, 837
 draw, 837
 FI_Scrollbar, 837
 handle, 837
 linesize, 838
 value, 838
FI_Secret_Input, 840
 FI_Secret_Input, 840
 handle, 841
FI_Select_Browser, 842
 FI_Select_Browser, 842
FI_Shared_Image, 843
 ~FI_Shared_Image, 846
 add, 846
 as_shared_image, 846
 color_average, 846
 compare, 847
 copy, 848
 desaturate, 848
 draw, 848
 find, 849
 FI_Shared_Image, 845
 get, 849, 850
 num_images, 850
 original, 850
 refcount, 851
 release, 851
 reload, 851
 remove_handler, 851
 uncache, 851
FI_Shared_Image.H, 1303
 fl_register_images, 1303
FI_Shortcut
 fl_types.h, 1310
FI_Simple_Counter, 852
FI_Simple_Terminal, 853
 ansi, 856, 857
 append, 857
 clear, 858
 current_style_index, 858
 draw, 859
 enforce_history_lines, 859
 enforce_stay_at_bottom, 859
 history_lines, 859, 860
 normal_style_index, 860
 printf, 860
 remove_lines, 861
 stay_at_bottom, 861, 862
 style_table, 862, 863
 style_table_size, 863
 text, 864
 vprintf, 864
FI_Single_Window, 865
 show, 866
FI_Slider, 866
 bounds, 868
 draw, 868

Fl_Slider, 868
handle, 869
scrollvalue, 870
slider, 870
slider_size, 870
Fl_Spinner, 871
 Fl_Spinner, 873
 format, 873
 handle, 873
 maximum, 874
 minimum, 874, 875
 range, 875
 resize, 875
 step, 875, 876
 textcolor, 876
 textfont, 876
 textsize, 877
 type, 877
 value, 877, 878
 wrap, 878
Fl_Spinner::Fl_Spinner_Input, 879
 handle, 879
Fl_Surface_Device, 880
 ~Fl_Surface_Device, 882
 driver, 882
 Fl_Surface_Device, 881
 pop_current, 882
 push_current, 882
 set_current, 883
 surface, 883
Fl_Sys_Menu_Bar, 893
 about, 897
 add, 897, 899
 clear, 899
 clear_submenu, 900
 create_window_menu, 900
 draw, 900
 Fl_Sys_Menu_Bar, 897
 insert, 901
 menu, 902
 mode, 902
 remove, 902
 replace, 903
 setonly, 903
 update, 903
 window_menu_style, 903
 window_menu_style_enum, 896
Fl_Sys_Menu_Bar.H, 1304
Fl_Table, 904
 ~Fl_Table, 912
 array, 913
 callback, 913
 callback_col, 914
 callback_context, 914
 callback_row, 914
 child, 914
 children, 915
 clear, 915
 col_header, 915
 col_resize, 916
 col_resize_min, 916
 col_width, 916
 col_width_all, 916
 cursor2rowcol, 916
 damage_zone, 917
 do_callback, 917
 draw, 917
 draw_cell, 917
 find_cell, 919
 Fl_Table, 912
 get_selection, 919
 init_sizes, 920
 insert, 920
 is_interactive_resize, 920
 is_selected, 920
 move_cursor, 921
 recalc_dimensions, 921
 redraw_range, 921
 resize, 922
 row_col_clamp, 922
 row_header, 922
 row_height, 922
 row_height_all, 923
 row_resize, 923
 row_resize_min, 923
 scrollbar_size, 923
 set_selection, 924
 tab_cell_nav, 924, 925
 table_box, 925
 table_resized, 925
 table_scrolled, 925
 TableContext, 912
 top_row, 926
 visible_cells, 926
 when, 926
Fl_Table_Row, 927
 ~Fl_Table_Row, 928
 clear, 929
 Fl_Table_Row, 928
 row_selected, 929
 select_all_rows, 929
 select_row, 929
 type, 930
Fl_Tabs, 930
 client_area, 935
 draw, 936
 Fl_Tabs, 935
 handle, 937
 push, 937
 tab_align, 938
 value, 938
 which, 939
Fl_Text_Buffer, 939
 add_modify_callback, 945
 address, 945, 946
 append, 946

appendfile, 946
byte_at, 946
char_at, 947
copy, 947
count_displayed_characters, 948
count_lines, 948
file_encoding_warning_message, 961
findchar_backward, 948
findchar_forward, 949
FI_Text_Buffer, 945
highlight_text, 949
insert, 949
insert_, 950
insertfile, 950
is_word_separator, 950
length, 951
line_end, 951
line_start, 951
line_text, 952
loadfile, 952
mPredeleteProcs, 961
mTabDist, 962
next_char, 952
outputfile, 952
prev_char, 953
printf, 953
remove, 954
remove_, 954
replace, 954
rewind_lines, 956
savefile, 956
search_backward, 956
search_forward, 957
secondary_selection_text, 957
selection_text, 957
skip_displayed_characters, 958
tab_distance, 958
text, 958
text_range, 960
transcoding_warning_action, 962
vprintf, 960
word_end, 961
word_start, 961
FI_Text_Display, 962
~FI_Text_Display, 971
absolute_top_line_number, 972
buffer, 972, 973
buffer_modified_cb, 973
buffer_predelete_cb, 974
calc_last_char, 974
calc_line_starts, 974
clear_rect, 976
col_to_x, 976
count_lines, 976
cursor_color, 977
cursor_style, 977
display_insert, 978
draw, 978
draw_cursor, 978
draw_line_numbers, 979
draw_range, 979
draw_string, 979
draw_text, 980
draw_vline, 980
empty_vlines, 981
extend_range_for_styles, 981
find_line_end, 981
find_wrap_range, 982
find_x, 982
FI_Text_Display, 971
get_absolute_top_line_number, 983
handle_vline, 983
highlight_data, 984
in_selection, 985
insert, 985
insert_position, 986
line_end, 986
line_start, 987
linenumber_align, 987
linenumber_bgcolor, 987
linenumber_fgcolor, 988
linenumber_font, 988
linenumber_format, 988
linenumber_size, 988
linenumber_width, 989
longest_vline, 989
maintain_absolute_top_line_number, 989
maintaining_absolute_top_line_number, 990
measure_deleted_lines, 990
measure_proportional_character, 990
measure_vline, 991
move_down, 991
move_left, 991
move_right, 991
move_up, 992
offset_line_starts, 992
overstrike, 992
position_style, 993
position_to_line, 993
position_to_linecol, 995
position_to_xy, 995
redisplay_range, 996
reset_absolute_top_line_number, 996
resize, 996
rewind_lines, 997
scroll, 997
scroll_, 998
scroll_timer_cb, 998
scrollbar_align, 998
scrollbar_size, 999
scrollbar_width, 1000
shortcut, 1000
show_cursor, 1001
show_insert_position, 1001
skip_lines, 1001
string_width, 1002

textcolor, 1002
textfont, 1003
textsize, 1003
update_h_scrollbar, 1004
update_line_starts, 1004
update_v_scrollbar, 1004
vline_length, 1004
word_end, 1005
word_start, 1005
wrap_mode, 1006
wrap_uses_character, 1006
wrapped_column, 1007
wrapped_line_counter, 1007
wrapped_row, 1008
x_to_col, 1009
xy_to_position, 1009
xy_to_rowcol, 1009

Fl_Text_Display::Style_Table_Entry, 1249

Fl_Text_Editor, 1010

- add_default_key_bindings, 1013
- add_key_binding, 1014
- bound_key_function, 1014
- default_key_function, 1014

Fl_Text_Editor, 1013

global_key_bindings, 1023

insert_mode, 1015

Key_Func, 1013

- kf_backspace, 1015
- kf_c_s_move, 1015
- kf_copy, 1015
- kf_ctrl_move, 1016
- kf_cut, 1016
- kf_default, 1016
- kf_delete, 1016
- kf_down, 1017
- kf_end, 1017
- kf_enter, 1017
- kf_home, 1017
- kf_ignore, 1017
- kf_insert, 1018
- kf_left, 1018
- kf_m_s_move, 1018
- kf_meta_move, 1018
- kf_move, 1019
- kf_page_down, 1019
- kf_page_up, 1019
- kf_paste, 1019
- kf_right, 1020
- kf_select_all, 1020
- kf_shift_move, 1020
- kf_undo, 1020
- kf_up, 1020

remove_all_key_bindings, 1021
remove_key_binding, 1021
tab_nav, 1021, 1023

Fl_Text_Editor::Key_Binding, 1245

Fl_Text_Selection, 1024

end, 1025

includes, 1026
length, 1026
position, 1026
selected, 1027
set, 1027
start, 1028
update, 1028

Fl_Tile, 1029

- Fl_Tile, 1031
- handle, 1031
- position, 1032
- resize, 1032

Fl_Tiled_Image, 1033

- color_average, 1034
- copy, 1034
- desaturate, 1035
- draw, 1035

Fl_Tiled_Image, 1034

Fl_Timer, 1036

- direction, 1037
- draw, 1038
- Fl_Timer, 1037
- handle, 1038
- suspended, 1039

Fl_Toggle_Button, 1039

- Fl_Toggle_Button, 1040

Fl_Tooltip, 1041

- color, 1042, 1043
- current, 1043
- delay, 1043
- disable, 1043
- enable, 1043
- enabled, 1044
- enter_area, 1044
- font, 1044
- hidedelay, 1044, 1045
- hoverdelay, 1045
- margin_height, 1045
- margin_width, 1045, 1046
- size, 1046
- textcolor, 1046
- wrap_width, 1046, 1047

Fl_Tree, 1047

- add, 1058, 1059
- calc_dimensions, 1060
- calc_tree, 1060
- callback_item, 1060, 1061
- callback_reason, 1061
- clear, 1061
- clear_children, 1062
- close, 1062
- closeicon, 1063
- connectorstyle, 1064
- deselect, 1064
- deselect_all, 1065
- display, 1066
- displayed, 1066
- extend_selection, 1066

extend_selection_dir, 1067
find_clicked, 1068
find_item, 1068
first, 1069
first_selected_item, 1069
first_visible, 1069
first_visible_item, 1070
get_selected_items, 1070
handle, 1071
hposition, 1071
insert, 1072
insert_above, 1073
is_close, 1073, 1074
is_hscroll_visible, 1074
is_open, 1074, 1075
is_scrollbar, 1075
is_selected, 1076
is_vscroll_visible, 1077
item_clicked, 1077
item_draw_mode, 1078
item_labelbgcolor, 1079
item_labelfgcolor, 1079
item_labelfont, 1079
item_labelsize, 1079
item_pathname, 1080
item_reselect_mode, 1080
labelmarginleft, 1081
last, 1081
last_selected_item, 1081
last_visible, 1082
last_visible_item, 1082
load, 1082
next, 1083
next_item, 1083
next_selected_item, 1084
next_visible_item, 1085
open, 1086, 1087
open_toggle, 1087
openicon, 1088
prev, 1089
recalc_tree, 1089
remove, 1089
resize, 1089
root, 1090
root_label, 1090
scrollbar_size, 1090, 1091
select, 1091, 1092
select_all, 1093
select_only, 1093
select_toggle, 1094
selectbox, 1094
selectmode, 1095
set_item_focus, 1095
show_item, 1095, 1096
show_item_bottom, 1096
show_item_middle, 1096
show_item_top, 1097
show_self, 1097
showcollapse, 1097
showroot, 1098
sortorder, 1098
usericon, 1098
usericonmarginleft, 1099
vposition, 1099
widgetmarginleft, 1100
FI_Tree.H, 1305
 FI_Tree_Reason, 1305
FI_Tree_Connector
 FI_Tree_Prefs.H, 1308
FI_Tree_Item, 1100
 activate, 1107
 add, 1107, 1108
 calc_item_height, 1108
 child, 1108
 deactivate, 1109
 deparent, 1109
 depth, 1109
 deselect_all, 1109
 draw, 1109
 draw_horizontal_connector, 1110
 draw_item_content, 1110
 draw_vertical_connector, 1112
 drawbgcolor, 1112
 drawfgcolor, 1112
 find_child, 1113
 find_child_item, 1113, 1114
 find_clicked, 1114
 find_item, 1115
 FI_Tree_Item, 1106
 hide_widgets, 1115
 insert, 1115
 insert_above, 1115
 label, 1116
 label_h, 1116
 label_w, 1116
 label_x, 1116
 label_y, 1117
 labelbgcolor, 1117
 move, 1117, 1118
 move_above, 1118
 move_below, 1119
 move_into, 1119
 next, 1119
 next_displayed, 1120
 next_sibling, 1120
 next_visible, 1120
 parent, 1120
 prefs, 1121
 prev, 1121
 prev_displayed, 1121
 prev_sibling, 1121
 prev_visible, 1122
 recalc_tree, 1122
 remove_child, 1122, 1123
 reparent, 1123
 replace, 1123

replace_child, 1124
select, 1125
select_all, 1125
show_self, 1125
show_widgets, 1125
swap_children, 1125, 1126
tree, 1126
update_prev_next, 1126
userdeicon, 1127
usericon, 1127
visible_r, 1128
w, 1128

Fl_Tree_Item.H, 1306
Fl_Tree_Item_Array, 1128
add, 1130
clear, 1130
deparent, 1130
Fl_Tree_Item_Array, 1129
insert, 1130
manage_item_destroy, 1131
move, 1131
remove, 1131, 1132
reparent, 1132
replace, 1132

Fl_Tree_Item_Array.H, 1306
Fl_Tree_Item_Draw_Mode
 Fl_Tree_Prefs.H, 1308
Fl_Tree_Item_Reselect_Mode
 Fl_Tree_Prefs.H, 1308

Fl_Tree_Prefs, 1132
closedeicon, 1135
closeicon, 1136
item_draw_mode, 1136
item_labelbgcolor, 1136
marginbottom, 1136, 1137
opendeicon, 1137
openicon, 1137
selectmode, 1137
showcollapse, 1138
showroot, 1138
sortorder, 1138
userdeicon, 1138

Fl_Tree_Prefs.H, 1307
 Fl_Tree_Connector, 1308
 Fl_Tree_Item_Draw_Mode, 1308
 Fl_Tree_Item_Reselect_Mode, 1308
 Fl_Tree_Select, 1309
 Fl_Tree_Sort, 1309

Fl_Tree_Reason
 Fl_Tree.H, 1305

Fl_Tree_Select
 Fl_Tree_Prefs.H, 1309

Fl_Tree_Sort
 Fl_Tree_Prefs.H, 1309

Fl_Valuator, 1139
bounds, 1142
clamp, 1142
Fl_Valuator, 1141

format, 1142
handle_drag, 1142
handle_release, 1142
increment, 1143
maximum, 1143
minimum, 1143
precision, 1143
range, 1144
round, 1144
set_value, 1144
step, 1144
value, 1145

Fl_Value_Input, 1145
cursor_color, 1147, 1148
draw, 1148
Fl_Value_Input, 1147
handle, 1148
resize, 1149
shortcut, 1149, 1150
soft, 1150
textcolor, 1150
textfont, 1150, 1151
textsize, 1151

Fl_Value_Output, 1151
draw, 1153
Fl_Value_Output, 1152
handle, 1153
soft, 1154
textcolor, 1154
textfont, 1154, 1155
textsize, 1155

Fl_Value_Slider, 1155
draw, 1157
Fl_Value_Slider, 1156
handle, 1157
textcolor, 1158
textfont, 1158
textsize, 1158, 1159

Fl_When
 Enumerations.H, 1273

Fl_Widget, 1159
~Fl_Widget, 1167
activate, 1167
active, 1167
active_r, 1168
align, 1168
argument, 1169
as_gl_window, 1169
as_group, 1170
as_window, 1170
box, 1171
callback, 1172, 1173
changed, 1173
clear_active, 1174
clear_changed, 1174
clear_damage, 1174
clear_output, 1175
clear_visible, 1175

clear_visible_focus, 1175
color, 1175, 1176
color2, 1177
contains, 1177
copy_label, 1178
copy_tooltip, 1178
damage, 1178, 1179
damage_resize, 1180
deactivate, 1180
default_callback, 1180
deimage, 1181, 1182
do_callback, 1182, 1183
draw, 1183
draw_box, 1184
draw_focus, 1184
draw_label, 1184, 1185
FI_Widget, 1166
h, 1185
handle, 1185
hide, 1186
image, 1186, 1187
inside, 1188
is_label_copied, 1188
label, 1188, 1189
label_shortcut, 1189
labelcolor, 1190
labelfont, 1190, 1191
labelsize, 1191
labeltype, 1192
measure_label, 1192
output, 1193
parent, 1193, 1194
position, 1194
redraw, 1194
redraw_label, 1194
resize, 1194
selection_color, 1195
set_active, 1196
set_changed, 1196
set_output, 1196
set_visible, 1196
set_visible_focus, 1197
shortcut_label, 1197
show, 1197
size, 1198
take_focus, 1198
takesevents, 1199
test_shortcut, 1199
tooltip, 1200
top_window, 1201
top_window_offset, 1201
type, 1201, 1202
user_data, 1202
visible, 1202
visible_focus, 1203
visible_r, 1203
w, 1204
when, 1204
window, 1205
x, 1206
y, 1206
FI_Widget.H, 1313
FL_RESERVED_TYPE, 1314
FI_Widget_Surface, 1207
draw, 1208
draw_decorated_window, 1208
FI_Widget_Surface, 1208
origin, 1209
print_window_part, 1210
printable_rect, 1210
translate, 1210
FI_Widget_Tracker, 1211
deleted, 1212
exists, 1212
widget, 1213
FI_Window, 1213
~FI_Window, 1218
as_window, 1219
border, 1219
clear_border, 1219
clear_modal_states, 1219
current, 1220
current_, 1239
cursor, 1220, 1221
decorated_h, 1221
decorated_w, 1222
default_cursor, 1222
default_icon, 1222
default_icons, 1223
default_xclass, 1223, 1224
draw, 1224
FI_Window, 1217, 1218
flush, 1224
force_position, 1225
free_icons, 1225
free_position, 1225
fullscreen, 1226
fullscreen_screens, 1226
handle, 1226
hide, 1228
hotspot, 1228
icon, 1228, 1230
iconize, 1230
iconlabel, 1230
icons, 1231
label, 1231
make_current, 1231
menu_window, 1232
modal, 1232
non_modal, 1232
override, 1232
resize, 1232
screen_num, 1233
set_menu_window, 1233
set_modal, 1233
set_non_modal, 1234

set_tooltip_window, 1234
shape, 1234, 1235
show, 1236
shown, 1237
size_range, 1237
tooltip_window, 1238
wait_for_expose, 1238
xclass, 1238, 1239
Fl_Window.H, 1314
Fl_Wizard, 1240
 draw, 1241
 Fl_Wizard, 1241
 next, 1241
 prev, 1242
 value, 1242
Fl_XBM_Image, 1242
 Fl_XBM_Image, 1243
Fl_XColor, 1243
Fl_XPM_Image, 1244
 Fl_XPM_Image, 1244
fl_access
 Unicode and UTF-8 functions, 324
fl_add_symbol
 Drawing functions, 283
fl_alert
 Common Dialogs classes and functions, 345
fl_arc
 Drawing functions, 283, 284
fl_arc.cxx, 1284
fl_ask
 Common Dialogs classes and functions, 346
fl_ask.cxx, 1284
fl_ask.H, 1286
 Fl_Beep, 1287
 fl_message_position, 1288
fl_beep
 Common Dialogs classes and functions, 346
fl_begin_complex_polygon
 Drawing functions, 285
fl_begin_offscreen
 Drawing functions, 285
fl_begin_points
 Drawing functions, 286
fl_box
 Enumerations.H, 1274
fl_boxtype.cxx, 1288
 fl_internal_boxtype, 1290
 fl_rectbound, 1290
fl_can_do_alpha_blending
 Drawing functions, 286
fl_capture_window_part
 Drawing functions, 286
fl_casenumericssort
 numericsort.c, 1320
fl_chdir
 Unicode and UTF-8 functions, 324
fl_chmod
 Unicode and UTF-8 functions, 325
fl_choice
 Common Dialogs classes and functions, 347
fl_circle
 Drawing functions, 287
fl_clip
 Drawing functions, 282
fl_clip_box
 Drawing functions, 287
fl_clip_region
 Drawing functions, 288, 289
fl_color
 Color & Font functions, 268, 269
fl_color.cxx, 1290
fl_color_average
 Color & Font functions, 269
fl_color_chooser
 Common Dialogs classes and functions, 348, 349
fl_color_cube
 Enumerations.H, 1274
Fl_compose.cxx, 1291
fl_contrast
 Color & Font functions, 270
fl_copy_offscreen
 Drawing functions, 289
fl_create_offscreen
 Drawing functions, 289
fl_cursor
 Drawing functions, 290
fl_curve
 Drawing functions, 290
fl_curve.cxx, 1292
fl_darker
 Enumerations.H, 1274
fl_decode_uri
 File names and URI utility functions, 360
fl_define_FL_EMBOSSSED_LABEL
 Enumerations.H, 1274
fl_define_FL_ENGRAVED_LABEL
 Enumerations.H, 1274
fl_define_FL_ICON_LABEL
 Enumerations.H, 1275
fl_define_FL_IMAGE_LABEL
 Enumerations.H, 1275
fl_define_FL_MULTI_LABEL
 Enumerations.H, 1275
fl_define_FL_SHADOW_LABEL
 Enumerations.H, 1275
fl_delete_offscreen
 Drawing functions, 291
fl_dir_chooser
 Common Dialogs classes and functions, 350
fl_down
 Enumerations.H, 1275
fl_draw
 Drawing functions, 291, 292
fl_draw.H, 1293
fl_draw_box
 Drawing functions, 292

fl_draw_image
 Drawing functions, 293
fl_draw_image_mono
 Drawing functions, 294
fl_drawPixmap
 Drawing functions, 295
fl_draw_symbol
 Drawing functions, 296
fl_eventnames
 Events handling functions, 255
fl_expand_text
 Drawing functions, 296
fl_file_chooser
 Common Dialogs classes and functions, 351
fl_file_chooser_callback
 Common Dialogs classes and functions, 352
fl_file_chooser_ok_label
 Common Dialogs classes and functions, 352
fl_filename_absolute
 File names and URI utility functions, 360
fl_filename_expand
 File names and URI utility functions, 361
fl_filename_ext
 File names and URI utility functions, 361
fl_filename_free_list
 File names and URI utility functions, 362
fl_filename_isdir
 File names and URI utility functions, 362
fl_filename_list
 File names and URI utility functions, 362
fl_filename_match
 File names and URI utility functions, 363
fl_filename_name
 File names and URI utility functions, 364
fl_filename_relative
 File names and URI utility functions, 364
fl_filename_setext
 File names and URI utility functions, 365
fl_font
 Color & Font functions, 270
fl_fontnames
 Events handling functions, 255
fl_fopen
 Unicode and UTF-8 functions, 325
fl_frame
 Drawing functions, 296
 Enumerations.H, 1275
fl_frame2
 Drawing functions, 297
fl_gap
 Drawing functions, 297
fl_getcwd
 Unicode and UTF-8 functions, 326
fl_getenv
 Unicode and UTF-8 functions, 326
fl_gray_ramp
 Enumerations.H, 1276
fl_height
 Color & Font functions, 270, 271
fl_input
 Common Dialogs classes and functions, 352
fl_internal_boxtyle
 fl_boxtyle.cxx, 1290
fl_intptr_t
 platform_types.h, 1323
fl_latin1_to_local
 Color & Font functions, 271
fl_lighter
 Enumerations.H, 1276
fl_line_style
 Drawing functions, 297
fl_local_to_latin1
 Color & Font functions, 271
fl_local_to_mac_roman
 Color & Font functions, 272
fl_mac_os_version
 Mac OS X-specific symbols, 343
fl_mac_roman_to_local
 Color & Font functions, 272
fl_mac_set_about
 Mac OS X-specific symbols, 342
fl_make_path
 Unicode and UTF-8 functions, 327
fl_make_path_for_file
 Unicode and UTF-8 functions, 327
fl_measure
 Drawing functions, 298
fl_measurePixmap
 Drawing functions, 299
fl_message
 Common Dialogs classes and functions, 353
fl_message_hotspot
 Common Dialogs classes and functions, 353, 354
fl_message_icon
 Common Dialogs classes and functions, 354
fl_message_position
 Common Dialogs classes and functions, 354, 355
 fl_ask.H, 1288
fl_message_title
 Common Dialogs classes and functions, 356
fl_message_title_default
 Common Dialogs classes and functions, 356
fl_mkdir
 Unicode and UTF-8 functions, 327
fl_mult_matrix
 Drawing functions, 299
fl_nonspacing
 Unicode and UTF-8 functions, 328
fl_not_clipped
 Drawing functions, 300
fl_numericsort
 numericsort.c, 1321
fl_old_shortcut
 Drawing functions, 300
fl_open
 Unicode and UTF-8 functions, 328

fl_open_callback
 Mac OS X-specific symbols, 342

fl_open_display
 Fl.hxx, 1282

fl_open_ext
 Unicode and UTF-8 functions, 329

fl_open_uri
 File names and URI utility functions, 365

fl_password
 Common Dialogs classes and functions, 357

fl_pie
 Drawing functions, 301

fl_polygon
 Drawing functions, 302

fl_pop_clip
 Drawing functions, 303

fl_push_clip
 Drawing functions, 303

fl_push_matrix
 Drawing functions, 303

fl_putenv
 Unicode and UTF-8 functions, 329

fl_read_image
 Drawing functions, 303

fl_rect
 Drawing functions, 304

fl_rect.cxx, 1302

fl_rectbound
 fl_boxtype.hxx, 1290

fl_rectf
 Drawing functions, 304

fl_register_images
 Fl_Shared_Image.H, 1303

fl_rename
 Unicode and UTF-8 functions, 330

fl_rescale_offscreen
 Drawing functions, 305

fl_reset_spot
 Drawing functions, 305

fl_rgb_color
 Enumerations.H, 1276

fl_rmdir
 Unicode and UTF-8 functions, 330

fl_rotate
 Drawing functions, 305

fl_scale
 Drawing functions, 306

fl_scroll
 Drawing functions, 306

fl_set_spot
 Drawing functions, 307

fl_set_status
 Drawing functions, 307

fl_shortcut_label
 Drawing functions, 307, 309

fl_show_colormap
 Color & Font functions, 273

fl_show_colormap.H, 1303

fl_size
 Color & Font functions, 273

fl_stat
 Unicode and UTF-8 functions, 331

fl_strdup
 Fl_string, 367

Fl_string, 367
 fl_strdup, 367

fl_string.h, 1304

fl_system
 Unicode and UTF-8 functions, 331

fl_text_extents
 Color & Font functions, 274

fl_transform_dx
 Drawing functions, 309

fl_transform_dy
 Drawing functions, 309

fl_transform_x
 Drawing functions, 311

fl_transform_y
 Drawing functions, 311

fl_transformed_vertex
 Drawing functions, 311

fl_translate
 Drawing functions, 311

fl_types.h, 1309
 Fl_Shortcut, 1310

fl_ucs_to_Utf16
 Unicode and UTF-8 functions, 332

fl_uintptr_t
 platform_types.h, 1323

fl_unlink
 Unicode and UTF-8 functions, 332

fl_utf8.h, 1310

fl_utf8back
 Unicode and UTF-8 functions, 333

fl_utf8bytes
 Unicode and UTF-8 functions, 333

fl_utf8decode
 Unicode and UTF-8 functions, 333

fl_utf8encode
 Unicode and UTF-8 functions, 334

fl_utf8from_mb
 Unicode and UTF-8 functions, 334

fl_utf8fromma
 Unicode and UTF-8 functions, 334

fl_utf8fromwc
 Unicode and UTF-8 functions, 335

fl_utf8fwd
 Unicode and UTF-8 functions, 335

fl_utf8len
 Unicode and UTF-8 functions, 336

fl_utf8len1
 Unicode and UTF-8 functions, 336

fl_utf8locale
 Unicode and UTF-8 functions, 336

fl_utf8test
 Unicode and UTF-8 functions, 337

fl_utf8to_mb
 Unicode and UTF-8 functions, 337
fl_utf8toUtf16
 Unicode and UTF-8 functions, 338
fl_utf8toa
 Unicode and UTF-8 functions, 337
fl_utf8towc
 Unicode and UTF-8 functions, 338
fl_utf_strcasecmp
 Unicode and UTF-8 functions, 339
fl_utf_strncasecmp
 Unicode and UTF-8 functions, 339
fl_utf_tolower
 Unicode and UTF-8 functions, 340
fl_utf_toupper
 Unicode and UTF-8 functions, 340
fl_vertex
 Drawing functions, 312
fl_vertex.cxx, 1313
fl_wcwidth
 Unicode and UTF-8 functions, 340
fl_wcwidth_
 Unicode and UTF-8 functions, 341
fl_width
 Color & Font functions, 274, 275
flush
 Fl, 389
 Fl_Double_Window, 523
 Fl_GI_Window, 575
 Fl_Menu_Window, 723
 Fl_Overlay_Window, 743
 Fl_Preferences, 786
 Fl_Window, 1224
focus
 Events handling functions, 251
 Fl_Group, 592
font
 Fl_Tooltip, 1044
force_position
 Fl_Window, 1225
foreground
 Fl, 390
format
 Fl_Spinner, 873
 Fl_Valuator, 1142
format_char
 Fl_Browser, 423, 424
free_color
 Color & Font functions, 275
free_icons
 Fl_Window, 1225
free_position
 Fl_Window, 1225
full_height
 Fl_Browser, 424
 Fl_Browser_, 449
full_width
 Fl_Browser_, 449

fullscreen
 Fl_Window, 1226
fullscreen_screens
 Fl_Window, 1226
g
 Fl_Color_Chooser, 504
get
 Fl_Preferences, 786–789
 Fl_Shared_Image, 849, 850
get_absolute_top_line_number
 Fl_Text_Display, 983
get_awake_handler_
 Fl, 390
get_boxtype
 Fl, 390
get_color
 Color & Font functions, 275
get_font
 Color & Font functions, 276
get_font_name
 Color & Font functions, 276
get_font_sizes
 Color & Font functions, 276
get_key
 Events handling functions, 252
get_mouse
 Events handling functions, 252
get_selected_items
 Fl_Tree, 1070
get_selection
 Fl_Table, 919
get_system_colors
 Fl, 390
getUserdataPath
 Fl_Preferences, 790
gl.h, 1315
 gl_color, 1316
 gl_draw, 1316–1318
 gl_font, 1318
 gl_rect, 1318
 gl_rectf, 1318
 gl_texture_pile_height, 1319
gl_color
 gl.h, 1316
gl_draw
 gl.h, 1316–1318
gl_font
 gl.h, 1318
gl_rect
 gl.h, 1318
gl_rectf
 gl.h, 1318
gl_texture_pile_height
 gl.h, 1319
gl_visual
 Fl, 390
global
 Fl_Menu_, 689

global_key_bindings
 Fl_Text_Editor, 1023

grab
 Windows handling functions, 236

group
 Fl_Preferences, 791

groupExists
 Fl_Preferences, 791

groups
 Fl_Preferences, 791

h
 Fl_Help_Dialog, 600
 Fl_Image, 627, 628
 Fl_Widget, 1185
 Screen functions, 261

handle
 Events handling functions, 252
 Fl_Adjuster, 405
 Fl_Box, 413
 Fl_Browser_, 449
 Fl_Button, 467
 Fl_Check_Browser, 482
 Fl_Choice, 491
 Fl_Clock, 496
 Fl_Color_Chooser, 504
 Fl_Counter, 515
 Fl_Dial, 520
 Fl_File_Input, 554
 Fl_Free, 566
 Fl_Group, 593
 Fl_Help_View, 611
 Fl_Input, 640
 Fl_Light_Button, 676
 Fl_Menu_Bar, 699
 Fl_Menu_Button, 703
 Fl_Positioner, 765
 Fl_Repeat_Button, 813
 Fl_Return_Button, 815
 Fl_Roller, 824
 Fl_Scroll, 832
 Fl_Scrollbar, 837
 Fl_Secret_Input, 841
 Fl_Slider, 869
 Fl_Spinner, 873
 Fl_Spinner::Fl_Spinner_Input, 879
 Fl_Tabs, 937
 Fl_Tile, 1031
 Fl_Timer, 1038
 Fl_Tree, 1071
 Fl_Value_Input, 1148
 Fl_Value_Output, 1153
 Fl_Value_Slider, 1157
 Fl_Widget, 1185
 Fl_Window, 1226

handle_
 Events handling functions, 253

handle_drag
 Fl_Valuator, 1142

handle_mouse
 Fl_Input_, 648

handle_release
 Fl_Valuator, 1142

handle_vline
 Fl_Text_Display, 983

handletext
 Fl_Input_, 648

has_scrollbar
 Fl_Browser_, 450

help
 Fl, 403

hide
 Fl_Browser, 425
 Fl_Double_Window, 523
 Fl_File_Chooser, 540
 Fl_Help_Dialog, 600
 Fl_Menu_Item, 714
 Fl_Menu_Window, 723
 Fl_Overlay_Window, 743
 Fl_Widget, 1186
 Fl_Window, 1228

hide_overlay
 Fl_Gl_Window, 575

hide_widgets
 Fl_Tree_Item, 1115

hidedelay
 Fl_Tooltip, 1044, 1045

highlight_data
 Fl_Text_Display, 984

highlight_text
 Fl_Text_Buffer, 949

highres_image
 Fl_Image_Surface, 635

history_lines
 Fl_Simple_Terminal, 859, 860

horizontal
 Fl_Pack, 747

hotspot
 Fl_Window, 1228

hour
 Fl_Clock_Output, 499

hoverdelay
 Fl_Tooltip, 1045

hposition
 Fl_Browser_, 450
 Fl_Tree, 1071

hscrollbar
 Fl_Browser_, 463

hsv
 Fl_Color_Chooser, 505

hsv2rgb
 Fl_Color_Chooser, 506

hue
 Fl_Color_Chooser, 506

icon
 Fl_Browser, 425, 426
 Fl_Window, 1228, 1230

iconize
 Fl_Window, 1230
iconlabel
 Fl_Window, 1230
icons
 Fl_Window, 1231
iconsize
 Fl_File_Browser, 531
 Fl_File_Chooser, 540
ID
 Fl_Preferences, 779
idle
 Fl, 403
image
 Fl_Image_Surface, 635
 Fl_Menu_Item, 714
 Fl_Widget, 1186, 1187
in_selection
 Fl_Text_Display, 985
inactive
 Fl_Image, 628
includes
 Fl_Text_Selection, 1026
incr_height
 Fl_Browser, 426
 Fl_Browser_, 451
increment
 Fl_Valuator, 1143
index
 Fl_Input_, 649
init_sizes
 Fl_Group, 593
 Fl_Table, 920
input
 Fl_Input_Choice, 667
input_type
 Fl_Input_, 649
insert
 Fl_Browser, 426, 427
 Fl_Chart, 475
 Fl_Group, 594
 Fl_Input_, 650
 Fl_Menu_, 689
 Fl_Menu_Item, 714
 Fl_Sys_Menu_Bar, 901
 Fl_Table, 920
 Fl_Text_Buffer, 949
 Fl_Text_Display, 985
 Fl_Tree, 1072
 Fl_Tree_Item, 1115
 Fl_Tree_Item_Array, 1130
insert_
 Fl_Text_Buffer, 950
insert_above
 Fl_Tree, 1073
 Fl_Tree_Item, 1115
insert_mode
 Fl_Text_Editor, 1015
 insert_position
 Fl_Text_Display, 986
insertfile
 Fl_Text_Buffer, 950
inserting
 Fl_Browser_, 451
insertion_point_location
 Fl, 391
inside
 Fl_Widget, 1188
is_close
 Fl_Tree, 1073, 1074
is_hscroll_visible
 Fl_Tree, 1074
is_interactive_resize
 Fl_Table, 920
is_label_copied
 Fl_Widget, 1188
is_open
 Fl_Tree, 1074, 1075
is_scheme
 Fl, 391
is_scrollbar
 Fl_Tree, 1075
is_selected
 Fl_Table, 920
 Fl_Tree, 1076
is_vscroll_visible
 Fl_Tree, 1077
is_word_separator
 Fl_Text_Buffer, 950
item_at
 Fl_Browser, 427
 Fl_Browser_, 451
 Fl_Check_Browser, 482
item_clicked
 Fl_Tree, 1077
item_draw
 Fl_Browser, 428
item_draw_mode
 Fl_Tree, 1078
 Fl_Tree_Prefs, 1136
item_first
 Fl_Browser, 428
 Fl_Browser_, 453
 Fl_Check_Browser, 483
item_height
 Fl_Browser, 429
 Fl_Browser_, 453
 Fl_Check_Browser, 483
item_labelbgcolor
 Fl_Tree, 1079
 Fl_Tree_Prefs, 1136
item_labelfgcolor
 Fl_Tree, 1079
item_labelfont
 Fl_Tree, 1079
item_labelsize

FI_Tree, 1079
item_last
 FI_Browser, 429
 FI_Browser_, 453
item_next
 FI_Browser, 430
 FI_Browser_, 454
 FI_Check_Browser, 483
item_pathname
 FI_Menu_, 690
 FI_Tree, 1080
item_prev
 FI_Browser, 430
 FI_Browser_, 454
 FI_Check_Browser, 484
item_quick_height
 FI_Browser_, 454
item_reselect_mode
 FI_Tree, 1080
item_select
 FI_Browser, 431
 FI_Browser_, 455
 FI_Check_Browser, 484
item_selected
 FI_Browser, 431
 FI_Browser_, 455
 FI_Check_Browser, 484
item_swap
 FI_Browser, 431
 FI_Browser_, 455
 FI_Check_Browser, 485
item_text
 FI_Browser, 432
 FI_Browser_, 456
 FI_Check_Browser, 485
item_width
 FI_Browser, 432
 FI_Browser_, 456
 FI_Check_Browser, 486

Key_Func
 FI_Text_Editor, 1013
keyboard_screen_scaling
 Screen functions, 261
kf_backspace
 FI_Text_Editor, 1015
kf_c_s_move
 FI_Text_Editor, 1015
kf_copy
 FI_Text_Editor, 1015
kf_ctrl_move
 FI_Text_Editor, 1016
kf_cut
 FI_Text_Editor, 1016
kf_default
 FI_Text_Editor, 1016
kf_delete
 FI_Text_Editor, 1016
kf_down

 FI_Text_Editor, 1017
kf_end
 FI_Text_Editor, 1017
kf_enter
 FI_Text_Editor, 1017
kf_home
 FI_Text_Editor, 1017
kf_ignore
 FI_Text_Editor, 1017
kf_insert
 FI_Text_Editor, 1018
kf_left
 FI_Text_Editor, 1018
kf_m_s_move
 FI_Text_Editor, 1018
kf_meta_move
 FI_Text_Editor, 1018
kf_move
 FI_Text_Editor, 1019
kf_page_down
 FI_Text_Editor, 1019
kf_page_up
 FI_Text_Editor, 1019
kf_paste
 FI_Text_Editor, 1019
kf_right
 FI_Text_Editor, 1020
kf_select_all
 FI_Text_Editor, 1020
kf_shift_move
 FI_Text_Editor, 1020
kf_undo
 FI_Text_Editor, 1020
kf_up
 FI_Text_Editor, 1020

label
 FI_Bitmap, 409
 FI_File_Chooser, 540, 541
 FI_File_Icon, 549
 FI_Image, 628
 FI_Menu_Item, 715
 FI_Multi_Label, 727, 728
 FI_Pixmap, 756
 FI_RGB_Image, 820, 821
 FI_Tree_Item, 1116
 FI_Widget, 1188, 1189
 FI_Window, 1231

label_h
 FI_Tree_Item, 1116

label_shortcut
 FI_Widget, 1189

label_w
 FI_Tree_Item, 1116

label_x
 FI_Tree_Item, 1116

label_y
 FI_Tree_Item, 1117

labela

Fl_Multi_Label, 728
labelb
Fl_Multi_Label, 728
labelbgcolor
Fl_Tree_Item, 1117
labelcolor
Fl_Menu_Item, 715
Fl_Widget, 1190
labelfont
Fl_Menu_Item, 715, 716
Fl_Widget, 1190, 1191
labelmarginleft
Fl_Tree, 1081
labelsize
Fl_Menu_Item, 716
Fl_Widget, 1191
labeltype
Fl_File_Icon, 549
Fl_Menu_Item, 716
Fl_Widget, 1192
last
Fl_Tree, 1081
last_selected_item
Fl_Tree, 1081
last_visible
Fl_Tree, 1082
last_visible_item
Fl_Tree, 1082
ld
Fl_Image, 628, 629
leftedge
Fl_Browser_, 457
leftline
Fl_Help_View, 611
length
Fl_Text_Buffer, 951
Fl_Text_Selection, 1026
line_end
Fl_Input_, 650
Fl_Text_Buffer, 951
Fl_Text_Display, 986
line_start
Fl_Input_, 650
Fl_Text_Buffer, 951
Fl_Text_Display, 987
line_text
Fl_Text_Buffer, 952
lineno
Fl_Browser, 433
linenumber_align
Fl_Text_Display, 987
linenumber_bgcolor
Fl_Text_Display, 987
linenumber_fgcolor
Fl_Text_Display, 988
linenumber_font
Fl_Text_Display, 988
linenumber_format
Fl_Text_Display, 988
linenumber_size
Fl_Text_Display, 988
linenumber_width
Fl_Text_Display, 989
lineposition
Fl_Browser, 433
linesize
Fl_Scrollbar, 838
link
Fl_Help_View, 611
load
Fl_Browser, 434
Fl_File_Browser, 531
Fl_File_Icon, 549
Fl_Help_Dialog, 601
Fl_Help_View, 612
Fl_Plugin_Manager, 759
Fl_Tree, 1082
load_fti
Fl_File_Icon, 550
load_image
Fl_File_Icon, 550
load_system_icons
Fl_File_Icon, 550
loadfile
Fl_Text_Buffer, 952
lock
Multithreading support functions, 314
longest_vline
Fl_Text_Display, 989
lstep
Fl_Counter, 515
mPredeleteProcs
Fl_Text_Buffer, 961
mTabDist
Fl_Text_Buffer, 962
Mac OS X-specific symbols, 342
fl_mac_os_version, 343
fl_mac_set_about, 342
fl_open_callback, 342
mac.H, 1319
maintain_absolute_top_line_number
Fl_Text_Display, 989
maintaining_absolute_top_line_number
Fl_Text_Display, 990
make_current
Fl_GI_Window, 575
Fl_Window, 1231
make_overlay_current
Fl_GI_Window, 575
make_visible
Fl_Browser, 434
manage_item_destroy
Fl_Tree_Item_Array, 1131
margin_height
Fl_Tooltip, 1045
margin_width

FI_Tooltip, 1045, 1046
marginbottom
 FI_Tree_Prefs, 1136, 1137
margins
 FI_Paged_Device, 751
 FI_PostScript_File_Device, 772
 FI_Printer, 800
mark
 FI_Input_, 651
max_size
 FI_RGB_Image, 821
maximum
 FI_Progress, 805
 FI_Spinner, 874
 FI_Valuator, 1143
maximum_size
 FI_Input_, 652
maxsize
 FI_Chart, 476
measure
 FI_Label, 673
 FI_Menu_Item, 717
measure_deleted_lines
 FI_Text_Display, 990
measure_label
 FI_Widget, 1192
measure_proportional_character
 FI_Text_Display, 990
measure_vline
 FI_Text_Display, 991
menu
 FI_Input_Choice, 667, 668
 FI_Menu_, 690, 691
 FI_Sys_Menu_Bar, 902
menu_end
 FI_Menu_, 691
menu_linespacing
 FI, 392
menu_window
 FI_Window, 1232
menubutton
 FI_Input_Choice, 668
middleline
 FI_Browser, 435
minimum
 FI_Progress, 805
 FI_Spinner, 874, 875
 FI_Valuator, 1143
minute
 FI_Clock_Output, 499
modal
 FI_Window, 1232
 Windows handling functions, 236
mode
 FI_Color_Chooser, 506
 FI_GI_Window, 575, 576
 FI_Menu_, 692
 FI_Sys_Menu_Bar, 902
move
 FI_Browser, 435
 FI_Tree_Item, 1117, 1118
 FI_Tree_Item_Array, 1131
move_above
 FI_Tree_Item, 1118
move_below
 FI_Tree_Item, 1119
move_cursor
 FI_Table, 921
move_down
 FI_Text_Display, 991
move_into
 FI_Tree_Item, 1119
move_left
 FI_Text_Display, 991
move_right
 FI_Text_Display, 991
move_up
 FI_Text_Display, 992
Multithreading support functions, 313
 awake, 313
 lock, 314
 thread_message, 314
 unlock, 314
mvalue
 FI_Menu_, 692
NONE
 FI_Preferences, 796
Name
 FI_Preferences::Name, 1246
nchecked
 FI_Check_Browser, 486
new_list
 FI_Browser_, 457
newUUID
 FI_Preferences, 792
next
 FI_File_Icon, 550
 FI_Menu_Item, 717
 FI_Tree, 1083
 FI_Tree_Item, 1119
 FI_Wizard, 1241
next_char
 FI_Text_Buffer, 952
next_displayed
 FI_Tree_Item, 1120
next_item
 FI_Tree, 1083
next_selected_item
 FI_Tree, 1084
next_sibling
 FI_Tree_Item, 1120
next_visible
 FI_Tree_Item, 1120
next_visible_item
 FI_Tree, 1085
next_window

Windows handling functions, 237
 nitems
 FI_Check_Browser, 486
 non_modal
 FI_Window, 1232
 normal_style_index
 FI_Simple_Terminal, 860
 normalize
 FI_RGB_Image, 821
 FI_SVG_Image, 892
 num_images
 FI_Shared_Image, 850
 numericsort.c, 1320
 fl_casenumericssort, 1320
 fl_numericssort, 1321
 offscreen
 FI_Image_Surface, 635
 offset_line_starts
 FI_Text_Display, 992
 open
 FI_Tree, 1086, 1087
 open_toggle
 FI_Tree, 1087
 opendeicon
 FI_Tree_Prefs, 1137
 openicon
 FI_Tree, 1088
 FI_Tree_Prefs, 1137
 Option
 FI_Native_File_Chooser, 734
 option
 FI, 392, 393
 options
 FI_Native_File_Chooser, 737
 origin
 FI_Copy_Surface, 510, 511
 FI_EPS_File_Surface, 527
 FI_Image_Surface, 635, 636
 FI_PostScript_File_Device, 772, 773
 FI_Printer, 801
 FI_SVG_File_Surface, 886
 FI_Widget_Surface, 1209
 original
 FI_Shared_Image, 850
 ortho
 FI_GI_Window, 577
 output
 FI_Widget, 1193
 outputfile
 FI_Text_Buffer, 952
 override
 FI_Window, 1232
 overstrike
 FI_Text_Display, 992
 own_colormap
 FI, 394
 Page_Format
 FI_Paged_Device, 749
 Page_Layout
 FI_Paged_Device, 749
 parent
 FI_Tree_Item, 1120
 FI_Widget, 1193, 1194
 paste
 Selection & Clipboard functions, 258
 pattern
 FI_File_Icon, 551
 picked
 FI_Menu_, 693
 pixel_h
 FI_GI_Window, 577
 pixel_w
 FI_GI_Window, 577
 pixels_per_unit
 FI_GI_Window, 577
 Pixmap
 FI_FormsPixmap, 562
 platform_types.h, 1322
 fl_intptr_t, 1323
 fl_uintptr_t, 1323
 pop_current
 FI_Surface_Device, 882
 popup
 FI_Menu_Button, 703
 FI_Menu_Item, 717
 popup_buttons
 FI_Menu_Button, 702
 position
 FI_Browser_, 457
 FI_Help_Dialog, 601
 FI_Input_, 652, 653
 FI_Text_Selection, 1026
 FI_Tile, 1032
 FI_Widget, 1194
 position_style
 FI_Text_Display, 993
 position_to_line
 FI_Text_Display, 993
 position_to_linecol
 FI_Text_Display, 995
 position_to_xy
 FI_Text_Display, 995
 precision
 FI_Valuator, 1143
 prefs
 FI_Tree_Item, 1121
 preset_file
 FI_Native_File_Chooser, 737
 prev
 FI_Tree, 1089
 FI_Tree_Item, 1121
 FI_Wizard, 1242
 prev_char
 FI_Text_Buffer, 953
 prev_displayed

FI_Tree_Item, 1121
prev_sibling
 FI_Tree_Item, 1121
prev_visible
 FI_Tree_Item, 1122
preview
 FI_File_Chooser, 541
print
 FI_Mac_App_Menu, 678
print_window_part
 FI_Widget_Surface, 1210
printable_rect
 FI_Copy_Surface, 511
 FI_EPS_File_Surface, 528
 FI_Image_Surface, 636
 FI_PostScript_File_Device, 773
 FI_Printer, 801
 FI_SVG_File_Surface, 887
 FI_Widget_Surface, 1210
printf
 FI_Simple_Terminal, 860
 FI_Text_Buffer, 953
program_should_quit
 FI, 394
proportional
 FI_SVG_Image, 893
pulldown
 FI_Menu_Item, 718
push
 FI_Tabs, 937
push_current
 FI_Surface_Device, 882
pushed
 Events handling functions, 253

r
 FI_Color_Chooser, 507
 FI_Rect, 810
RGB_scaling
 FI_Image, 629, 630
radio
 FI_Menu_Item, 718
range
 FI_Spinner, 875
 FI_Valuator, 1144
readonly
 FI_Input_, 653
readqueue
 FI, 394
ready
 FI, 395
recalc_dimensions
 FI_Table, 921
recalc_scrollbars
 FI_Scroll, 832
recalc_tree
 FI_Tree, 1089
 FI_Tree_Item, 1122
rectangle_capture

FI_Device_Plugin, 517
redisplay_range
 FI_Text_Display, 996
redraw
 FI_Widget, 1194
redraw_label
 FI_Widget, 1194
redraw_line
 FI_Browser_, 458
redraw_lines
 FI_Browser_, 458
redraw_overlay
 FI_GI_Window, 578
 FI_Overlay_Window, 743
redraw_range
 FI_Table, 921
refcount
 FI_Shared_Image, 851
release
 FI, 395
 FI_Image, 629
 FI_Shared_Image, 851
release_widget_pointer
 Safe widget deletion support functions, 317
reload
 FI_Shared_Image, 851
reload_scheme
 FI, 395
remove
 FI_Browser, 435
 FI_Check_Browser, 486
 FI_Group, 594, 595
 FI_Menu_, 693
 FI_Sys_Menu_Bar, 902
 FI_Text_Buffer, 954
 FI_Tree, 1089
 FI_Tree_Item_Array, 1131, 1132
remove_
 FI_Text_Buffer, 954
remove_all_key_bindings
 FI_Text_Editor, 1021
remove_check
 FI, 396
remove_child
 FI_Tree_Item, 1122, 1123
remove_fd
 FI, 396
remove_handler
 Events handling functions, 254
 FI_Shared_Image, 851
remove_icon
 FI_Browser, 436
remove_key_binding
 FI_Text_Editor, 1021
remove_lines
 FI_Simple_Terminal, 861
remove_system_handler
 Events handling functions, 254

remove_timeout
 Fl, 396
 removePlugin
 Fl_Plugin_Manager, 760
 reparent
 Fl_Tree_Item, 1123
 Fl_Tree_Item_Array, 1132
 repeat_timeout
 Fl, 396
 replace
 Fl_Browser, 436
 Fl_Chart, 476
 Fl_Input, 654
 Fl_Menu, 693
 Fl_Sys_Menu_Bar, 903
 Fl_Text_Buffer, 954
 Fl_Tree_Item, 1123
 Fl_Tree_Item_Array, 1132
 replace_child
 Fl_Tree_Item, 1124
 replacing
 Fl_Browser, 458
 rescale
 Fl_Image_Surface, 636
 rescan
 Fl_File_Chooser, 541
 reset_absolute_top_line_number
 Fl_Text_Display, 996
 reset_marked_text
 Fl, 397
 resizable
 Fl_Group, 595, 596
 resize
 Fl_Browser, 459
 Fl_Double_Window, 523
 Fl_Gl_Window, 578
 Fl_Group, 597
 Fl_Help_Dialog, 601
 Fl_Help_View, 612
 Fl_Input, 655
 Fl_Overlay_Window, 743
 Fl_SVG_Image, 892
 Fl_Scroll, 833
 Fl_Spinner, 875
 Fl_Table, 922
 Fl_Text_Display, 996
 Fl_Tile, 1032
 Fl_Tree, 1089
 Fl_Value_Input, 1149
 Fl_Widget, 1194
 Fl_Window, 1232
 rewind_lines
 Fl_Text_Buffer, 956
 Fl_Text_Display, 997
 rgb
 Fl_Color_Chooser, 507
 rgb2hsv
 Fl_Color_Chooser, 507
 Root
 Fl_Preferences, 779
 root
 Fl_Tree, 1090
 root_label
 Fl_Tree, 1090
 rotate
 Fl_Paged_Device, 751
 Fl_PostScript_File_Device, 773
 Fl_Printer, 802
 round
 Fl_Valuator, 1144
 row_col_clamp
 Fl_Table, 922
 row_header
 Fl_Table, 922
 row_height
 Fl_Table, 922
 row_height_all
 Fl_Table, 923
 row_resize
 Fl_Table, 923
 row_resize_min
 Fl_Table, 923
 row_selected
 Fl_Table_Row, 929
 run
 Fl, 397
 runtime graphics driver configuration, 231
 runtime printer driver configuration, 232
 runtime system configuration, 234
 runtime window and event manager configuration, 233
 STRICT_RFC3629
 Unicode and UTF-8 functions, 323
 Safe widget deletion support functions, 315
 clear_widget_pointer, 316
 delete_widget, 316
 do_widget_deletion, 316
 release_widget_pointer, 317
 watch_widget_pointer, 317
 saturation
 Fl_Color_Chooser, 508
 savefile
 Fl_Text_Buffer, 956
 scale
 Fl_Image, 630
 Fl_Paged_Device, 752
 Fl_PostScript_File_Device, 774
 Fl_Printer, 802
 scaling_algorithm
 Fl_Image, 631
 scheme
 Fl, 397
 Screen functions, 260
 h, 261
 keyboard_screen_scaling, 261
 screen_dpi, 261
 screen_num, 262

screen_scale, 262
screen_scaling_supported, 263
screen_work_area, 263, 264
screen_xywh, 264–266
w, 266
x, 266
y, 266
screen_dpi
 Screen functions, 261
screen_num
 Fl_Window, 1233
 Screen functions, 262
screen_scale
 Screen functions, 262
screen_scaling_supported
 Screen functions, 263
screen_work_area
 Screen functions, 263, 264
screen_xywh
 Screen functions, 264–266
scroll
 Fl_Text_Display, 997
scroll_
 Fl_Text_Display, 998
scroll_timer_cb
 Fl_Text_Display, 998
scroll_to
 Fl_Scroll, 833
Scrollbar
 Fl_Browser_, 463
Scrollbar_align
 Fl_Text_Display, 998
Scrollbar_left
 Fl_Browser_, 459
Scrollbar_right
 Fl_Browser_, 459
Scrollbar_size
 Fl, 398
 Fl_Browser_, 459
 Fl_Help_View, 613
 Fl_Scroll, 834
 Fl_Table, 923
 Fl_Text_Display, 999
 Fl_Tree, 1090, 1091
Scrollbar_width
 Fl_Browser_, 460
 Fl_Text_Display, 1000
scrollvalue
 Fl_Slider, 870
search_backward
 Fl_Text_Buffer, 956
search_forward
 Fl_Text_Buffer, 957
second
 Fl_Clock_Output, 499
secondary_selection_text
 Fl_Text_Buffer, 957
select
 Fl_Browser, 436
 Fl_Browser_, 460
 Fl_Tree, 1091, 1092
 Fl_Tree_Item, 1125
select_all
 Fl_Help_View, 614
 Fl_Tree, 1093
 Fl_Tree_Item, 1125
select_all_rows
 Fl_Table_Row, 929
select_only
 Fl_Browser_, 461
 Fl_Tree, 1093
select_row
 Fl_Table_Row, 929
select_toggle
 Fl_Tree, 1094
selectbox
 Fl_Tree, 1094
selected
 Fl_Browser, 437
 Fl_Text_Selection, 1027
selection
 Fl_Browser_, 461
 Selection & Clipboard functions, 259
Selection & Clipboard functions, 256
 add_clipboard_notify, 256
 clipboard_contains, 257
 copy, 257
 dnd, 257
 paste, 258
 selection, 259
 selection_owner, 259
selection_color
 Fl_Widget, 1195
selection_owner
 Selection & Clipboard functions, 259
selection_text
 Fl_Text_Buffer, 957
selectmode
 Fl_Tree, 1095
 Fl_Tree_Prefs, 1137
set
 Fl_Button, 468
 Fl_FormsBitmap, 560
 Fl_FormsPixmap, 562
 Fl_Menu_Item, 718
 Fl_Preferences, 792–795
 Fl_Text_Selection, 1027
set_active
 Fl_Widget, 1196
set_atclose
 Windows handling functions, 237
set_box_color
 Fl, 399
set_boxtype
 Fl, 399
set_changed

FI_Widget, 1196
set_checked
 FI_Check_Browser, 486
set_color
 Color & Font functions, 276, 277
set_current
 FI_Copy_Surface, 511
 FI_Image_Surface, 637
 FI_Printer, 803
 FI_Surface_Device, 883
set_draw_cb
 FI_Cairo_Window, 471
set_font
 Color & Font functions, 277
set_fonts
 Color & Font functions, 277
set_idle
 FI, 399
set_item_focus
 FI_Tree, 1095
set_labeltype
 FI, 400
set_menu_window
 FI_Window, 1233
set_modal
 FI_Window, 1233
set_non_modal
 FI_Window, 1234
set_output
 FI_Widget, 1196
set_overlay
 FI_Menu_Window, 724
set_selection
 FI_Table, 924
set_tooltip_window
 FI_Window, 1234
set_value
 FI_Valuator, 1144
set_visible
 FI_Widget, 1196
set_visible_focus
 FI_Widget, 1197
setonly
 FI_Menu, 694
 FI_Menu_Item, 718
 FI_Sys_Menu_Bar, 903
shadow
 FI_Clock_Output, 500
shape
 FI_Window, 1234, 1235
shortcut
 FI_Button, 468
 FI_Input, 655
 FI_Menu, 694
 FI_Menu_Item, 719
 FI_Text_Display, 1000
 FI_Value_Input, 1149, 1150
shortcut_label
 FI_Widget, 1197
show
 FI_Browser, 437
 FI_Double_Window, 523
 FI_File_Chooser, 541
 FI_Gl_Window, 578
 FI_Help_Dialog, 601
 FI_Menu_Item, 719
 FI_Menu_Window, 724
 FI_Native_File_Chooser, 738
 FI_Overlay_Window, 744
 FI_Single_Window, 866
 FI_Widget, 1197
 FI_Window, 1236
show_cursor
 FI_Text_Display, 1001
show_insert_position
 FI_Text_Display, 1001
show_item
 FI_Tree, 1095, 1096
show_item_bottom
 FI_Tree, 1096
show_item_middle
 FI_Tree, 1096
show_item_top
 FI_Tree, 1097
show_self
 FI_Tree, 1097
 FI_Tree_Item, 1125
show_widgets
 FI_Tree_Item, 1125
showHiddenButton
 FI_File_Chooser, 544
showcollapse
 FI_Tree, 1097
 FI_Tree_Prefs, 1138
shown
 FI_File_Chooser, 541
 FI_Window, 1237
showroot
 FI_Tree, 1098
 FI_Tree_Prefs, 1138
size
 FI_Browser, 438
 FI_File_Icon, 551
 FI_Help_View, 614
 FI_Input, 656
 FI_Menu, 694
 FI_Menu_Item, 719
 FI_Preferences, 795
 FI_Tooltip, 1046
 FI_Widget, 1198
size_range
 FI_Window, 1237
sizes
 FI_Group, 597
skip_displayed_characters
 FI_Text_Buffer, 958

skip_lines
 Fl_Text_Display, 1001

slider
 Fl_Slider, 870

slider_size
 Fl_Slider, 870

soft
 Fl_Adjuster, 406
 Fl_Value_Input, 1150
 Fl_Value_Output, 1154

sort
 Fl_Browser_, 461

sortorder
 Fl_Tree, 1098
 Fl_Tree_Prefs, 1138

start
 Fl_Text_Selection, 1028

start_job
 Fl_Paged_Device, 752
 Fl_PostScript_File_Device, 774

start_page
 Fl_Paged_Device, 752

static_value
 Fl_Input_, 656, 657

stay_at_bottom
 Fl_Simple_Terminal, 861, 862

step
 Fl_Counter, 516
 Fl_Spinner, 875, 876
 Fl_Valuator, 1144

str
 FL_CHART_ENTRY, 478

string_width
 Fl_Text_Display, 1002

style_table
 Fl_Simple_Terminal, 862, 863

style_table_size
 Fl_Simple_Terminal, 863

submenu
 Fl_Menu_Item, 720

surface
 Fl_Surface_Device, 883

suspended
 Fl_Timer, 1039

swap
 Fl_Browser, 438

swap_buffers
 Fl_GI_Window, 579

swap_children
 Fl_Tree_Item, 1125, 1126

swapping
 Fl_Browser_, 462

tab_align
 Fl_Tabs, 938

tab_cell_nav
 Fl_Table, 924, 925

tab_distance
 Fl_Text_Buffer, 958

tab_nav
 Fl_Input_, 657, 658
 Fl_Text_Editor, 1021, 1023

table_box
 Fl_Table, 925

table_resized
 Fl_Table, 925

table_scrolled
 Fl_Table, 925

TableContext
 Fl_Table, 912

take_focus
 Fl_Widget, 1198

takesevents
 Fl_Widget, 1199

test_shortcut
 Events handling functions, 254
 Fl_Menu_, 694
 Fl_Menu_Item, 720
 Fl_Widget, 1199

text
 Fl_Browser, 439
 Fl_Check_Browser, 487
 Fl_Menu_, 694, 695
 Fl_Simple_Terminal, 864
 Fl_Text_Buffer, 958

text_range
 Fl_Text_Buffer, 960

textcolor
 Fl_Chart, 477
 Fl_File_Chooser, 542
 Fl_Help_View, 614
 Fl_Input_, 658, 659
 Fl_Menu_, 695
 Fl_Spinner, 876
 Fl_Text_Display, 1002
 Fl_Tooltip, 1046
 Fl_Value_Input, 1150
 Fl_Value_Output, 1154
 Fl_Value_Slider, 1158

textfont
 Fl_Browser_, 462
 Fl_Chart, 477
 Fl_File_Chooser, 542
 Fl_Help_View, 614
 Fl_Input_, 659
 Fl_Menu_, 695
 Fl_Spinner, 876
 Fl_Text_Display, 1003
 Fl_Value_Input, 1150, 1151
 Fl_Value_Output, 1154, 1155
 Fl_Value_Slider, 1158

textsize
 Fl_Browser, 439
 Fl_Chart, 477
 Fl_File_Chooser, 542, 543
 Fl_Help_Dialog, 602
 Fl_Help_View, 615

FI_Input_, 660
FI_Menu_, 695, 696
FI_Spinner, 877
FI_Text_Display, 1003
FI_Value_Input, 1151
FI_Value_Output, 1155
FI_Value_Slider, 1158, 1159
thread_message
 Multithreading support functions, 314
title
FI_Help_View, 615
FI_Native_File_Chooser, 738
tooltip
FI_Widget, 1200
tooltip_window
FI_Window, 1238
top
FI_Help_Font_Stack, 604
top_row
FI_Table, 926
top_window
FI_Widget, 1201
top_window_offset
FI_Widget, 1201
topline
FI_Browser, 440
FI_Help_View, 615, 616
transcoding_warning_action
FI_Text_Buffer, 962
translate
FI_Copy_Surface, 512
FI_EPS_File_Surface, 528
FI_Image_Surface, 637
FI_PostScript_File_Device, 774
FI_Printer, 803
FI_SVG_File_Surface, 887
FI_Widget_Surface, 1210
tree
FI_Tree_Item, 1126
Type
FI_Native_File_Chooser, 734
type
FI_File_Chooser, 543
FI_File_Icon, 551
FI_Label, 674
FI_Spinner, 877
FI_Table_Row, 930
FI_Widget, 1201, 1202
typea
FI_Multi_Label, 728
typeb
FI_Multi_Label, 728
uncache
FI_Bitmap, 409
FI_Image, 631
FI_Pixmap, 756
FI_RGB_Image, 822
FI_Shared_Image, 851
uncheck
FI_Menu_Item, 720
undo
FI_Input_, 660
Unicode and UTF-8 functions, 321
ERRORS_TO_CP1252, 323
ERRORS_TO_ISO8859_1, 323
fl_access, 324
fl_chdir, 324
fl_chmod, 325
fl_fopen, 325
fl_getcwd, 326
fl_getenv, 326
fl_make_path, 327
fl_make_path_for_file, 327
fl_mkdir, 327
fl_nonspacing, 328
fl_open, 328
fl_open_ext, 329
fl_putenv, 329
fl_rename, 330
fl_rmdir, 330
fl_stat, 331
fl_system, 331
fl_ucs_to_Utf16, 332
fl_unlink, 332
fl_utf8back, 333
fl_utf8bytes, 333
fl_utf8decode, 333
fl_utf8encode, 334
fl_utf8from_mb, 334
fl_utf8froma, 334
fl_utf8fromwc, 335
fl_utf8fwd, 335
fl_utf8len, 336
fl_utf8len1, 336
fl_utf8locale, 336
fl_utf8test, 337
fl_utf8to_mb, 337
fl_utf8toUtf16, 338
fl_utf8toa, 337
fl_utf8towc, 338
fl_utf_strcasecmp, 339
fl_utf_strncasecmp, 339
fl_utf_tolower, 340
fl_utf_toupper, 340
fl_wcwidth, 340
fl_wcwidth_, 341
STRICT RFC3629, 323
unlock
 Multithreading support functions, 314
up_down_position
FI_Input_, 660
update
FI_Menu_Bar, 700
FI_Sys_Menu_Bar, 903
FI_Text_Selection, 1028
update_child

FI_Group, 598
update_h_scrollbar
 FI_Text_Display, 1004
update_line_starts
 FI_Text_Display, 1004
update_menubutton
 FI_Input_Choice, 668
update_prev_next
 FI_Tree_Item, 1126
update_v_scrollbar
 FI_Text_Display, 1004
use_high_res_GL
 FI, 400
user_data
 FI_File_Chooser, 543
 FI_Widget, 1202
userdeicon
 FI_Tree_Item, 1127
 FI_Tree_Prefs, 1138
usericon
 FI_Tree, 1098
 FI_Tree_Item, 1127
usericonmarginleft
 FI_Tree, 1099

val
 FL_CHART_ENTRY, 478
valid
 FI_GI_Window, 579
value
 FI_Browser, 440, 441
 FI_Button, 469
 FI_Check_Browser, 487
 FI.Choice, 492, 493
 FI_Clock_Output, 500, 501
 FI_Color_Chooser, 508
 FI_File_Chooser, 543
 FI_File_Icon, 551
 FI_File_Input, 555
 FI_Help_Dialog, 602
 FI_Help_View, 616
 FI_Input_, 661, 662
 FI_Input_Choice, 669
 FI_Menu_, 696
 FI_Menu_Item, 720
 FI_Positioner, 766
 FI_Progress, 805, 806
 FI_Scrollbar, 838
 FI_Spinner, 877, 878
 FI_Tabs, 938
 FI_Valuator, 1145
 FI_Wizard, 1242
version
 FI, 401
visible
 FI_Browser, 441
 FI_File_Chooser, 544
 FI_Help_Dialog, 602
 FI_Menu_Item, 721
 FI_Widget, 1202
visible_cells
 FI_Table, 926
visible_focus
 FI, 401
 FI_Widget, 1203
visible_r
 FI_Tree_Item, 1128
 FI_Widget, 1203
visual
 FI, 401
vline_length
 FI_Text_Display, 1004
vposition
 FI_Tree, 1099
vprintf
 FI_Simple_Terminal, 864
 FI_Text_Buffer, 960

w
 FI_Help_Dialog, 602
 FI_Image, 631
 FI_Tree_Item, 1128
 FI_Widget, 1204
 Screen functions, 266
wait
 FI, 402
wait_for_expose
 FI_Window, 1238
warning
 Common Dialogs classes and functions, 358
watch_widget_pointer
 Safe widget deletion support functions, 317
when
 FI_Table, 926
 FI_Widget, 1204
which
 FI_Tabs, 939
widget
 FI_Widget_Tracker, 1213
widgetmarginleft
 FI_Tree, 1100
window
 FI_Widget, 1205
window_menu_style
 FI_Sys_Menu_Bar, 903
window_menu_style_enum
 FI_Sys_Menu_Bar, 896
Windows handling functions, 235
 atclose, 237
 default_atclose, 235
 first_window, 235, 236
 grab, 236
 modal, 236
 next_window, 237
 set_atclose, 237
word_end
 FI_Input_, 662
 FI_Text_Buffer, 961

FI_Text_Display, 1005
word_start
 FI_Input_, 663
 FI_Text_Buffer, 961
 FI_Text_Display, 1005
wrap
 FI_Input_, 663
 FI_Spinner, 878
wrap_mode
 FI_Text_Display, 1006
wrap_uses_character
 FI_Text_Display, 1006
wrap_width
 FI_Tooltip, 1046, 1047
wrapped_column
 FI_Text_Display, 1007
wrapped_line_counter
 FI_Text_Display, 1007
wrapped_row
 FI_Text_Display, 1008

x
 FI_Help_Dialog, 603
 FI_Widget, 1206
 Screen functions, 266

x_to_col
 FI_Text_Display, 1009

xbounds
 FI_Positioner, 766

xclass
 FI_Window, 1238, 1239

xposition
 FI_Scroll, 835

xstep
 FI_Positioner, 766

xvalue
 FI_Positioner, 766, 767

xy_to_position
 FI_Text_Display, 1009

xy_to_rowcol
 FI_Text_Display, 1009

y
 FI_Help_Dialog, 603
 FI_Widget, 1206
 Screen functions, 266

ybounds
 FI_Positioner, 767

yposition
 FI_Scroll, 835

ystep
 FI_Positioner, 767

yvalue
 FI_Positioner, 767