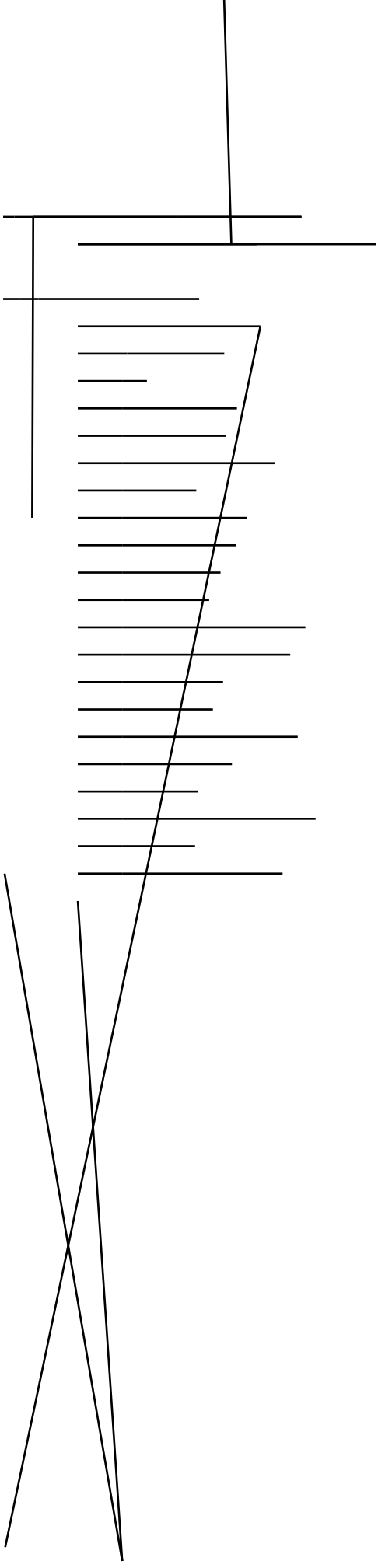
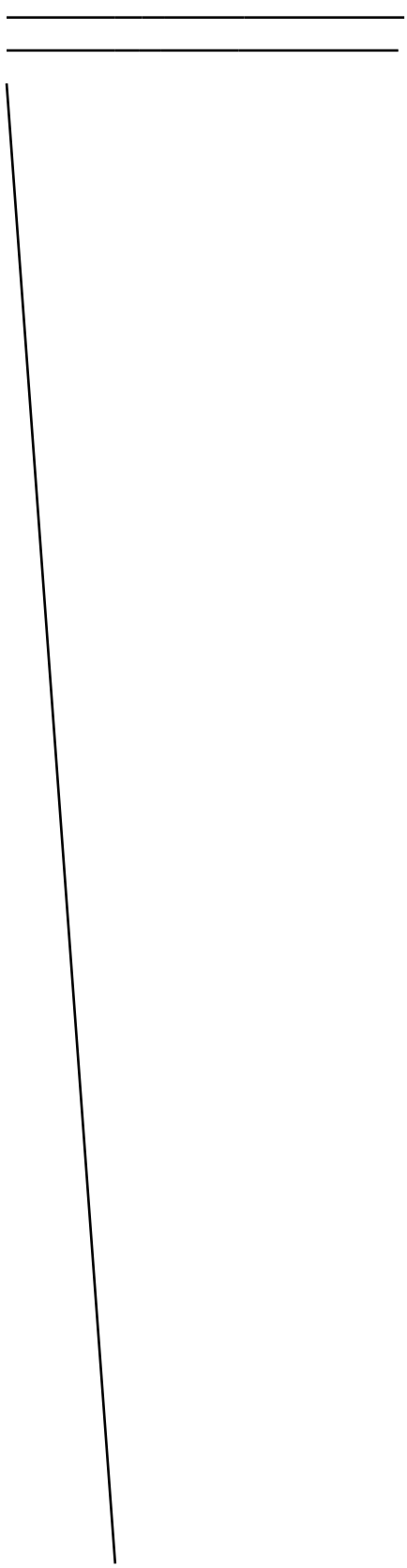
FLTK 1.1.5 Programming Manual

Revision 5



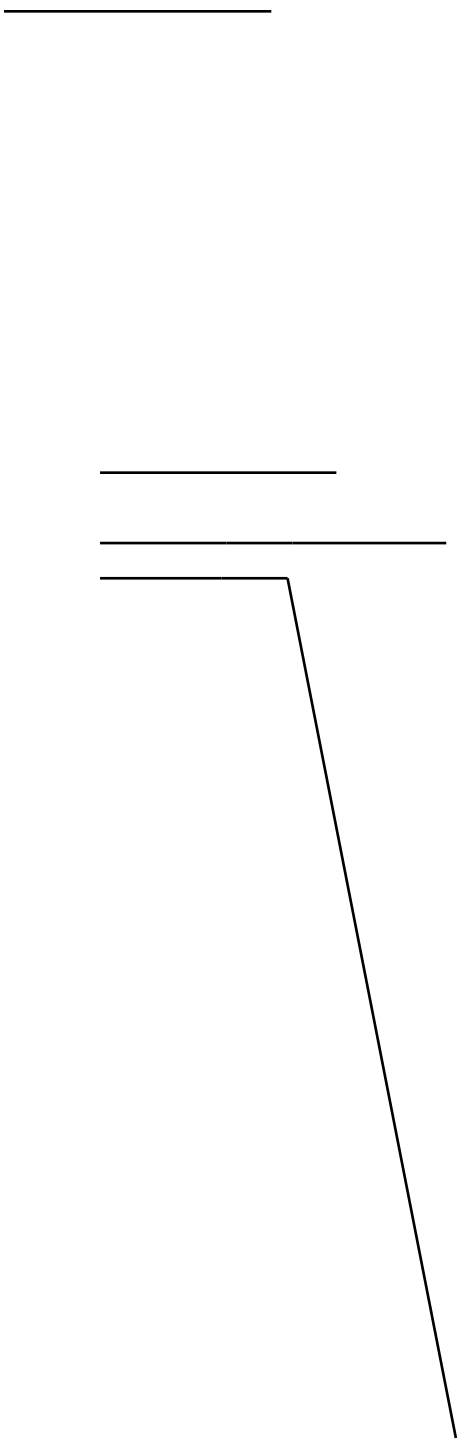
Preface

This manual describes the Fast Light Tool Kit ("FLTK") version 1.1.5, a C++ Graphical User Interface



interface.

With the death of NeWS Bill realized that he would have to live with X. The biggest problem with X is the



For general support and questions, please use the FLTK mailing list at "fltk@fltk.org" or one of the newsgroups.

2 – FLTK Basics

Timer functions are called after a specific amount of time has expired. They can be used to pop up a progress

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`.

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 and WinSock (WSOCK32.LIB) libraries to the "Link" settings.

You can build your Microsoft Windows applications as Console or WIN32 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.

Note: The Visual C++ 5.0 optimizer is known to cause problems with many e "Proji2−j /F4 11 Tf.nly re wh thd3.2 Td(

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 s:*

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

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.

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");
```

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

```
button->color(FL_RED);
```

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

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

Box Types

The type `Fl_Boxtype` stored and returned in `Fl_Widget::box()` is an enumeration defined in `<Enumerations.H>`. Figure 3−3 shows the standard box types included with FLTK.

Figure 3−3: 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 3−3 is the area that is not drawn by the frame types.

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.

Note:

This interface has changed in FLTK 2.0!

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) {
...
}
```

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

```
nt x, int y, int w, int h, Fl_Color c) {

, hr(c);
```

Shortcuts

Shortcuts are key o0,uences that activate widgets such as buttons oc menu items. The `shortcut()` method sets the shortcut foc 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 ASCIFii6T9T1sT1sT1sT1sT1sT1sT1sT1sT1sT1sT1sT1sT1sT1sT1kAF

4 − Designing a Simple Text Editor

This chapter takes you through the design of a simple FLTK−base4 text editor.

Determining the Goals of the Text Editor

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

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 {inhat, inhah, const char* t); {); {:this (:)Tj /F0 9.2
```

```
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!
```

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("O(F File?", "*", filename);
```

```
  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);
}
```

replall_cb()

This callback will replace all occurences 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();

  e};
  }
```

if (times) fl_messageTeeTee 3(if (8fd %d occurrences.", fl_mess;)Tj 0 -110 TtimelseageTalert(No occurrences os)'%s)' found

```
#ifdef 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);
}
```

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();
}
```

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 Chapter 1, 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!

The Final Product

The final editor window should look like the image in Figure 4-2.

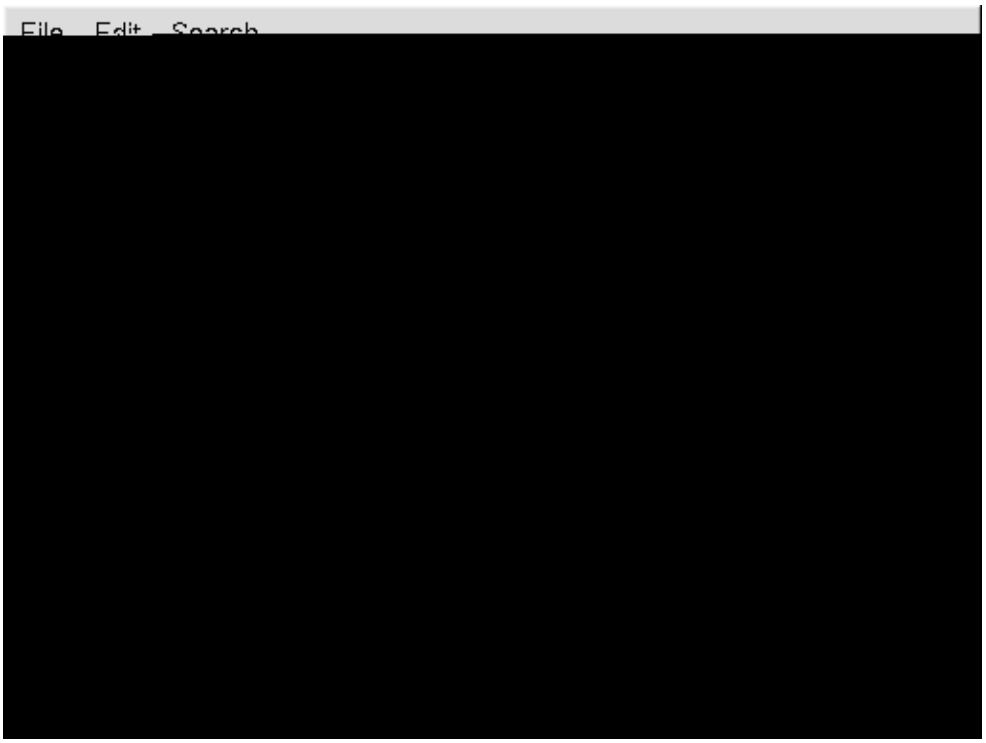


Figure 4−2: The completed editor window

```
    { 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:

```
     *style++t= currein;,
     *style++t= currein;,
     text ++;,
     length −−;,
     colt+= 2;,
int  continue;, last;
char } else if (*texn== '\'') {,
     currein= srmBt;,
     char temp;
ons} else if (!last&&islower(*texn)) {,
     // Might be a keyword...,
     for (tempt= texn, ' srmBt= ' s;,
          islower(*temp)t&&r' srmBt< (' s + sizeof(' s)t− 1);,
           ' srmB++t= *temp++)t;
```

```
          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';
  }
}
```

The only difference between this function and `fl_frame2()` is the order of the line segments.

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

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`

style

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

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)

void fl_line(int x, int y, int x x, int y, int x1, ioopy1)

Drawing Complex Shapes

in the opposite direction of the outside loop.

`fl_gap()`

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.

6 – Handling Events

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()`.

To receive FL_DRAG events you must also respond to the FL_PUSH and FL_RELEASE events.

FL_RELEASE

A mouse button has been released. You can find out what button by calling `Fl::event_button()`.

FL_MOVE

The mouse has moved without any mouse buttons held down. This event is sent to the `Fl::belowmouse()`

FL_UNFOCUS

This event is sent to the previous `Fl::focus()` widget when another widget gets the focus or the window loses focus.

Keyboard Events

FL_KEYDOWN, FL_KEYUP

A key was pressed or released. The key can be found in `Fl::event_key()`. The text that the key should insert can be found with `Fl::event_text()` and its length is in `Fl::event_length()`. If you use the key `handle()` should return 1. If you return zero then FLTK assumes you ignored the key and will then attempt to send it to a parent widget. If none of them want it, it will change the event into a `FL_SHORTCUT` event.

To receive `FL_KEYBOARD` events you must also respond to the `FL_FOCUS` and `FL_UNFOCUS` events.

If you are writing a text−editing widget you may also want to call the `Fl::compose()` function to translate individual keystrokes into foreign characters.

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 windowss()7isplayed or which one has the focus.

Widget Events

FL_DEACTIVATE

This widget is no longer active, due to `deactivate()` being called on it or one of its parents. `active()` may still be true after this, the widget is only active if `active()` is true on it and all its parents (use `active_r()` to check this).

FL_ACTIVATE

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

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. `visible()` may still be true after this, but the widget is visible only if `visible()` is true for it and all its parents (use `visible_r()` to check this).

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. *Child `Fl_Window`s 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*

7 – Adding and Extending Widgets

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

Subclassing

New widgets are created by *subclassing* an existing FLTK widget, typicallT `Fl_Widget` for controls and `Fl_Group` for composite widgets.

A control widget typicallT interacts with the user to receive and/or display a value of some sort.

A composite widget widget holds a list of child widgets and handles moving, sizing, showing, or hiding them as needed. `Fl_Group` is the main composite widget 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 onlT difference is the code that draws the face of the button.

Making a Subclass of Fl_Widget

Your subclasses can directlT descend from `Fl_Widget` or any subclass of `Fl_Widget`. `Fl_Widget` has onlT four virtual methods, and overriding some or all of these may be necessary.

The Constructor

void Fl_Widget::set_flag(SHORTCUT_LABEL)

Modifies `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()`

Resizing the Widget

The `resize(int x, int y, int w, int 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 hau hEmoareefficieint mayof edrazing the new positio.l

moved. atchi 0.priu Usd is ca:0 11 Tf 9likef 0.0-24.567angevoid MyCith ::statiT_slider_cb oll

Drag And Drop Support

FLTK provides routines to drag and drop 8−bit text between applications:

Drag'n'drop operations are are initiated by copying data to the clipboard and calling the function `Fl::dnd()`.

Drop attempts are handled via events:

- `FL_DND_ENTER`
- `FL_DND_DRAG`
- `FL_DND_LEAVE`
- `FL_DND_RELEASE`
- `FL_PASTE`

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 IDhat ouricate 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.

8 – Using OpenGL

This chapter discussea using FLTK for your OpenGL applications.

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 usea OpenGL calls to draw the display. Your main program should call `redraw()`

If your subclass provides static controls in the window, they must be redrawn whenever the

Using OpenGL in Normal FLTK Windows

You can put OpenGL code into an

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 *)
float gl_width(const char *, int n)
float gl_width(uchar)

Returns information about the current OpenGL font.

void gl_draw(const char *)
void gl_draw(const char *, int n)

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.

The draw() Method

The `draw()` method performs the needed initialization and does the actual drawing:

```
void OptimizerWindow::draw() {
  if (!context_) {
    // This is the first time we've been asked to draw; create the
    // Optimizer context for the scene...

#ifdef WIN32
```

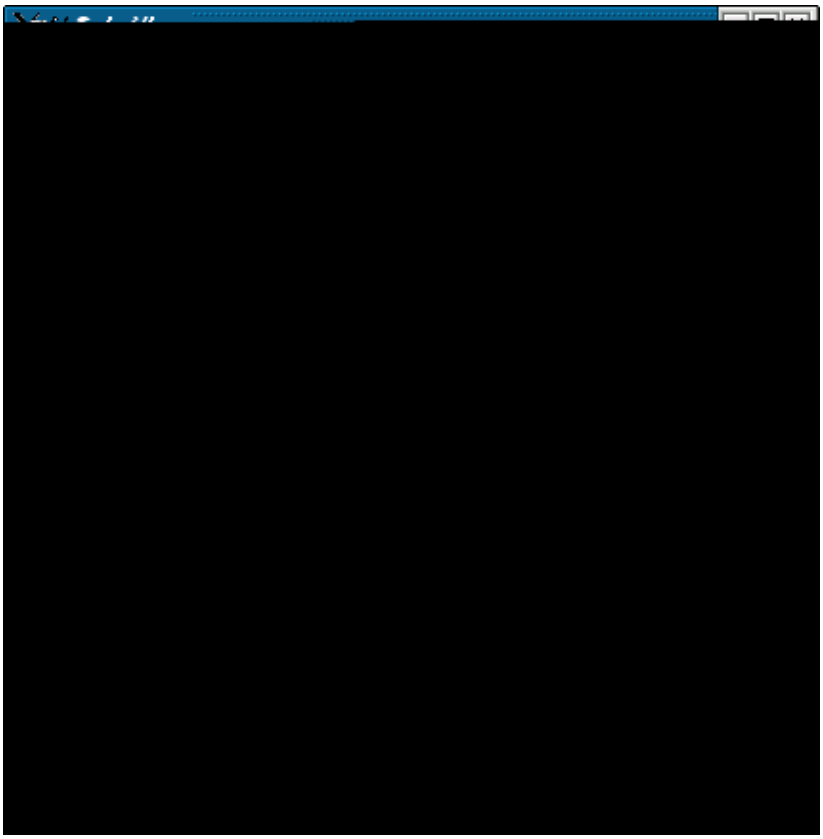
9 – Programming with FLUID

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

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 FLUID



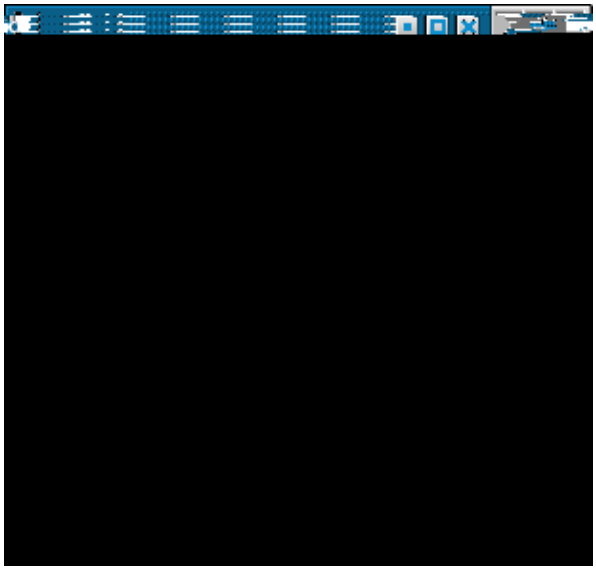
```
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_LINES);
      glVertex3fv(boxv0);
      glVertex3fv(boxv1);

      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();
};//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();
};
```

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**



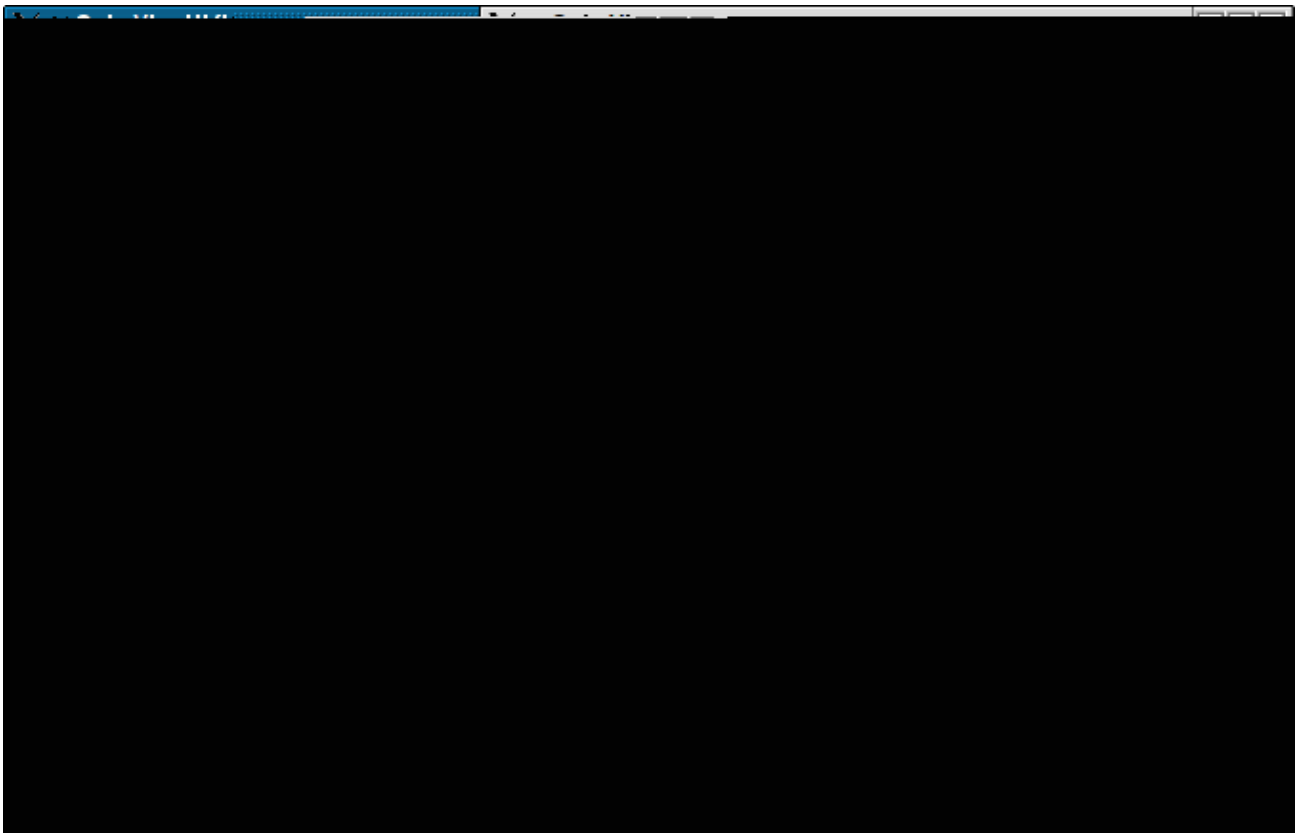


Figure 9−4: FLUID window containing CubeView demo.

We will talk about the `show()` method that is highlighted shortly.) h)))5Tj /F4 11New−>Oabor−>Boxow() toiadd aus

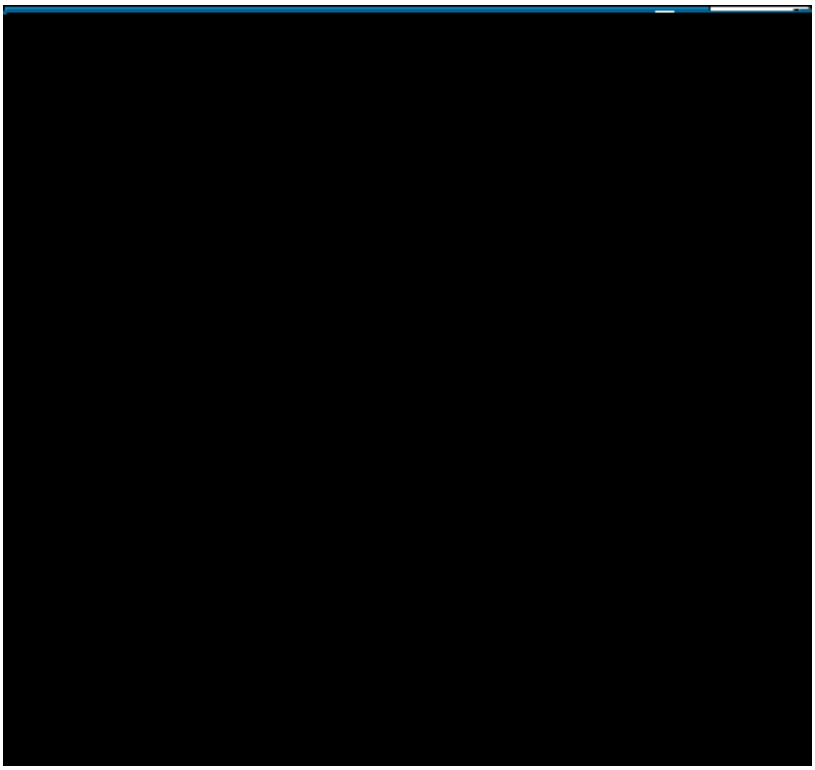


Figure 9−5: CubeView methods.

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 in a short amount of code in the "Callback" field of the widget panel. For example, the callback for the `ypan` slider is:

```
cube->pany(((Fl_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 no 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. An an example add a show function so that CubeViewUI can actually appear on the screen.

Figure 9-6: Code Window

Once the new code window has been added, highlight its name and select New−>Code

engine that was constructed.

FLUID Reference

The following sections describe each of the windows in FLUID.

The Widget Browser

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

The widgets are stored in a hierarchy. You c .ny Widg 11 iyse a level by1 iicking 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 compysed of *functions* and *classes*. Each of these will produce a single C++ pubiic function or class in the outpft `.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 `Fl_Window`.

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

Widgets are shown in the browser by1either their *name* (such as "main_panel" the osf widwle), or by1their *type* and *label* (such as "Button " osfgreen"").

You *select* widgets by1 iicking on1their names, which highlights them (you c .nalso select widgets from any displayed window). You c .nselect many widgets by1dragging the mouse across them, or by1using Shift+Ciick to toggle them oidg 11off. To select no widgets,1 iick 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 *y Wi* widgets by1doubie−ciicking on1them, or (to y Widseveral widgets you have picked) by1typing the F1 key. A control panel will appear so you c .nchange the widget(s).

Menu Items

key. Aecontheit4th1 worawsk s by1du c most wt thei4 11 th:.2 Tf 0 −.84 Td(The enu bFab)/Ol w...dseCtrl+o) Tf 0 −26.84 Td(T

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

Displays the project settings panel. The output filenames control the extensions or names of the files the are generated by FLUID. If you check the "Include .h from .cxx" button the code file will include the header file automatically.

The internationalization options are described later in this chapter.

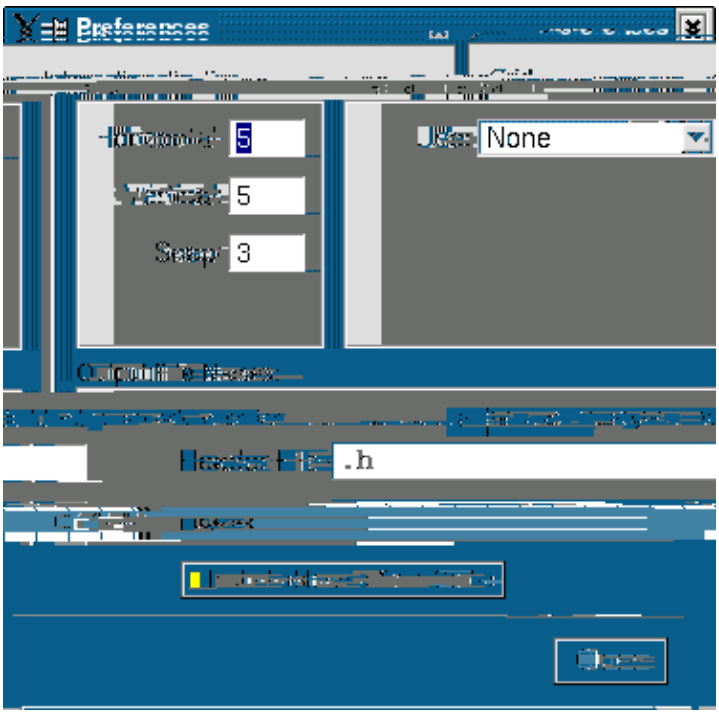


Figure 9−8: The FLUID widget GUI attributes.

GUI Attributes

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 the 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 a Tab. If you have a large data display surrounded by buttons, you probably want that data area to be resizable. You can get more aomplex behavidatby making invisible boxes the resizable widget, datby using hierara Tes of groups. Unfortunately the only way to test it is to aompile the program. Resizing the FLUID window is *not* the same as what will happen in the user program.

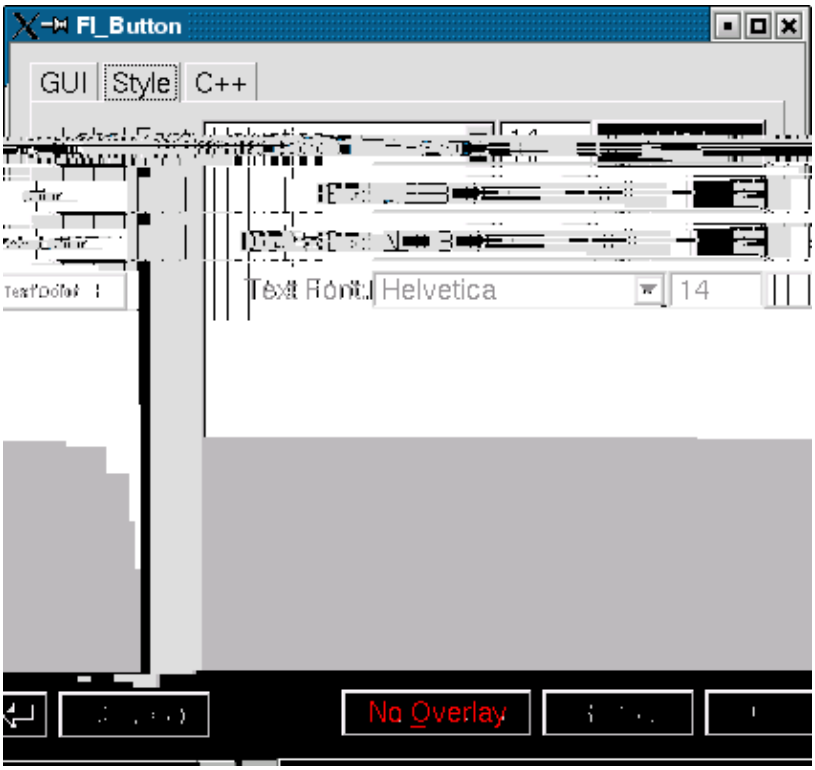
The **Hotspot** button causes the parent window to be positioned with that widget centered on the mouse. This position is determined *when the FLUID function is called*, so you should call it immediately before showing the window. If you want the window to hide and then reappear at a new position, you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

The **Border** button turns the window manageatborder on datoff. On most window manageas 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 manageatas the class. This can change the id

the wffect.


Fl_Button
GUI
Style
C++
Text Font
Helvetica
14
No Overlay

Down Box (pulldown menu)

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

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 9−10: The FLUID widget C++ attributes.

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.

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 `a mg` 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 `a mg` 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 hing if the data is not changed.

Selecting and Moving Widgets

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.

chooser. Three new input fields will then appear to control the include file, catalog file, and set number for retrieving the localized label strings.

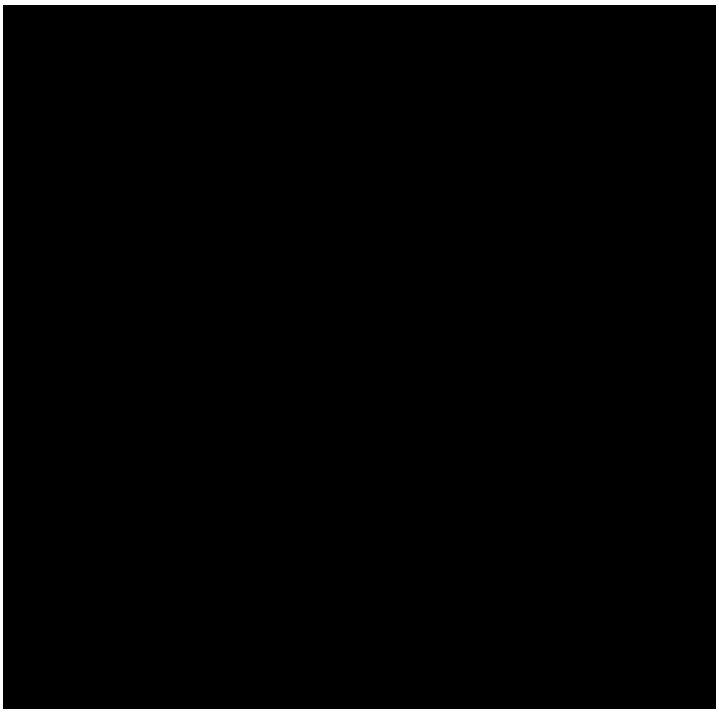
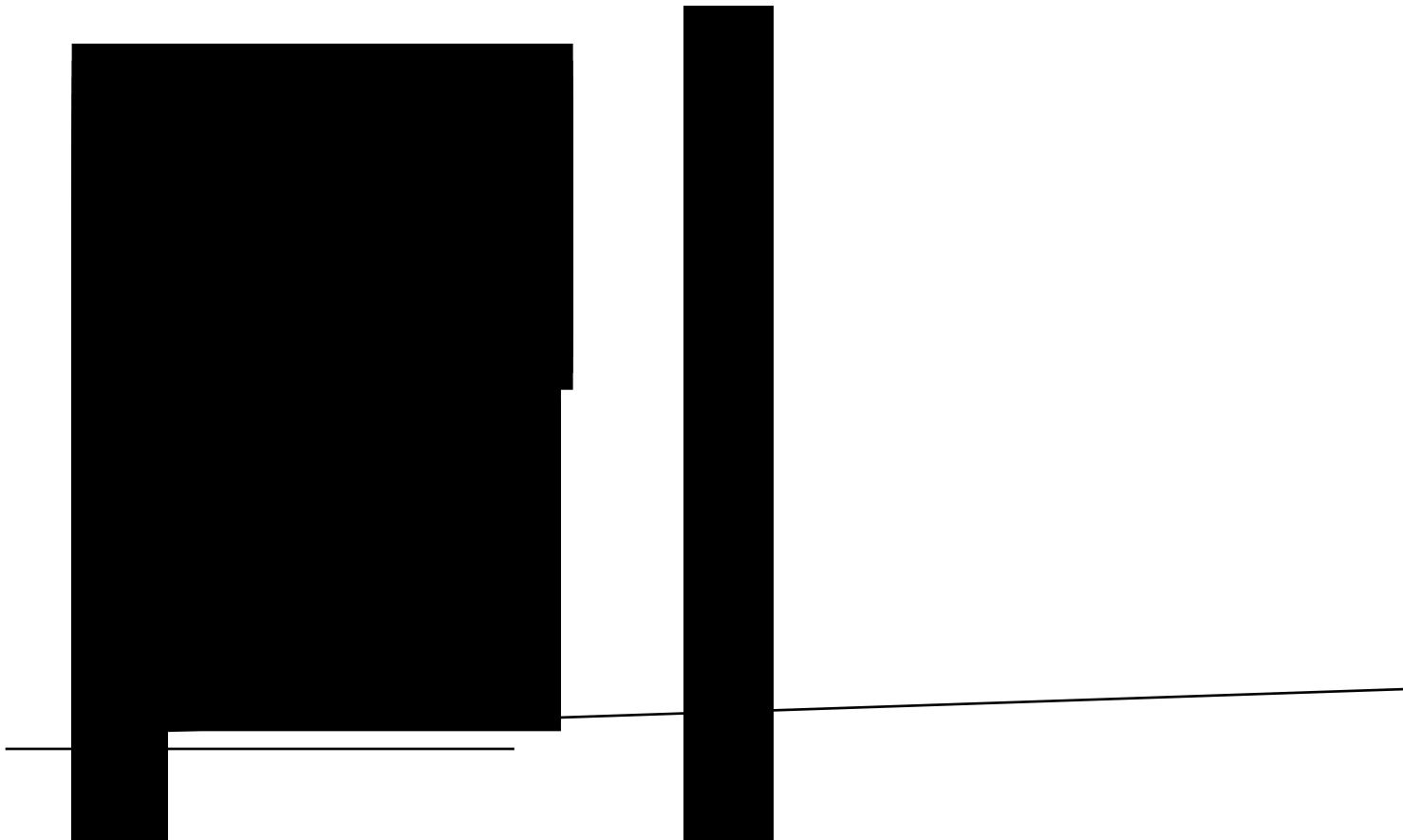


Figure 9−12: Internationalization using POSIX catgets.

The "#include" field controls the header file to include for I18N; by degehoehcm m .<nl_types.h>.,s thr

The Ffile" field controls thename ofs the catalog fil variabile touse wthen(retrieving localizedo msagesN; br)Tj 0 −13.2

The Sete" field controls the set numberin the catalog filr. Tey degehoe setm 1, andrarelye nelds tobeochangeds.



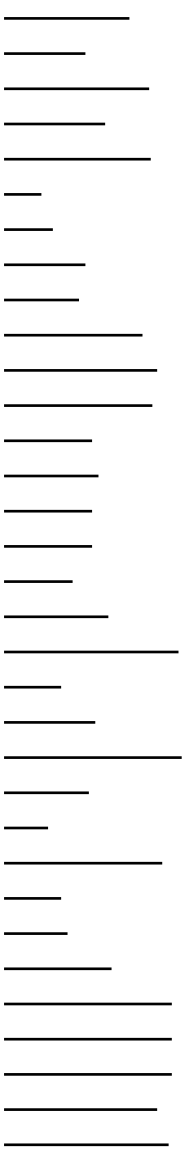
Fl_Slider

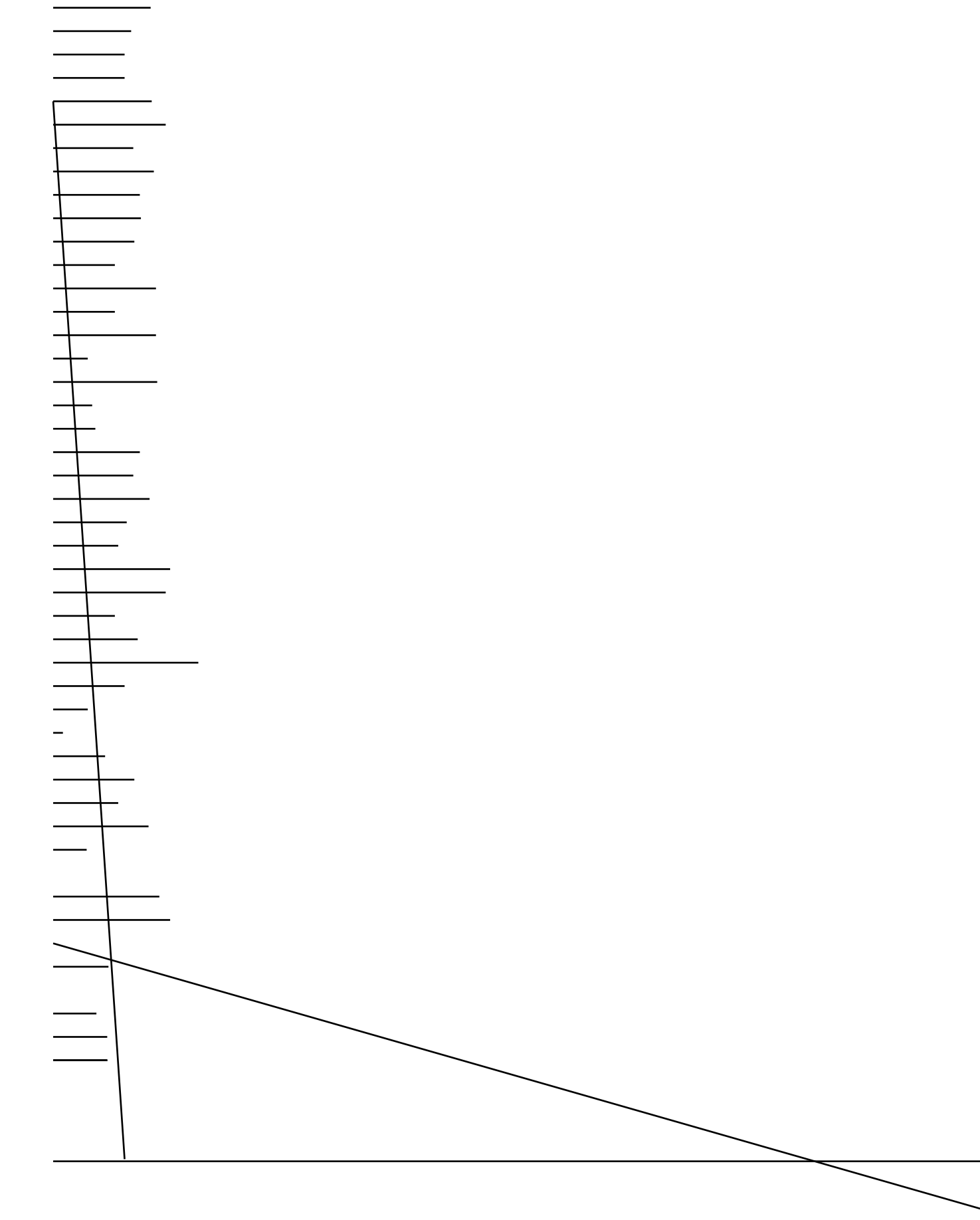
Fl_ValueOutputr

- Fl_Scrollbar
- Fl_Value_Slider

class Fl

Class Hierarchy





- repeat_timeout
- run
- scheme
- selection
- selection_owner
- set_abort
- set_atclose
- set_boxtype
- set_color
- set_font
- set_fonts
- set_idle

int box_dh(Fl_Boxtype);

Returns the height offset for the given boxtype.

int box_dw(Fl_Boxtype);

Returns the width offset for the given boxtype.

int box_dx(Fl_Boxtype);

Returns the X offset for the given boxtype.

int box_dy(Fl_Boxtype);

Returns the Y offset for the given boxtype.

int check();

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;
}
```

The returns non−zero if any windows are displayed, and 0 if no windows are displayed (this is likely to

int event_button3();

Returns non−zer00 5 button 3 is pressed.

int event_button();

Returns which mouse button was pressed. This returns garbage0 5 the most recent event was not a FL_PUSH or FL_RELEASE event.

int event_buttons();

Returns the button state bits;0 5 non−zer0, then at least one button is pressed.

int event_clicks();
void event_clicks(int i);

The first form returns non−zer00 5 the most recent FL_PUSH or FL_KEYBOARD was a "double click". Returns N−1 for N clicks. A double click is counted0 5 the same button is pressed again while `event_is_click()` is true.

The second form directly sets the number returned by `Fl::event_clicks()`. This can be used to set it to zer00so that later code does not think an item was double−clicked.

int event_ctrl();

Returns non−zer00 5 the Control key is pressed.

int event();

Returns the last event that was processed. This can be used to determine0 5 a callback is being done0 n response to a keypress, mouse click, etc.

int event_inside(int,int,int,int);
int event_inside(const Fl_Widget*);

Returns non−zer00 5 the current `event_x` and `event_y` put it inside the widget or inside an arbitrary bounding box. You should always call this rather than doing your own comparison so you are consistent about edge effects.

int event_is_click();
void event_is_click(0);

The first form returns non−zer00 5 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

int event_key();
int event_key(int s);

`Fl::event_key()` returns which key on the keyboard was last pushed. It returns zero if the last event was not a key press or release.

`Fl::event_key(int)` 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

- 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*

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.

int get_font_sizes(Fl_Font, int*& sizep);

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.

int get_key(int);

void get_mouse(int &x,int &y);

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.

void get_system_colors();

Read the user preference colors from the system and use them to call `Fl::foreground()`, `Fl::background()`, and `Fl::background2()`. This is done by `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.

int gl_visual(int, int *alist=0);

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_G::visual`XasualAiogrds thisuery t

If `grab()` is o m t will also affect show() of windows by doing system−specific ol uations (o mXm t turns o override−redirect). These are designed to make menus popup reliably and faster o mthe system.

To turn off grabbing do `Fl::grab(0)`.

Be carefulmthat your podifl4,es not enastea o nfinite loop whiledo grab() is . O(o mth(it willock oput yoo)Tj 0 -13.2 Tscre

void remove_check(Fl_Timeout_Handler, void* = 0);

Removes a check callback. It is harmless to remove a check callback that no longer exists.

void scheme(const char *name);
const char *scheme();

Gets or sets the current widget scheme. Currently only "none" and "plastic" are recognized, and `NULL` will use the scheme defined in the `FLTK_SCHEME` environment variable or the `scheme` resource under X11.

void selection(Fl_Widget &owner, const char* stuff, int len);

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.

int visual(int);

signal happens).

void (*warning)(const char*, ...);

FLTK calls this to print a warning message. 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.

int x();

Returns the origin of the current screen, where 0 indicates the left side of the screen.

int y();

Returns the origin of the current screen, where 0 indicates the top edge of the screen.

int event_dx();

int event_dy();

const char* event_text();

class Fl_Adjuster

Class Hierarchy

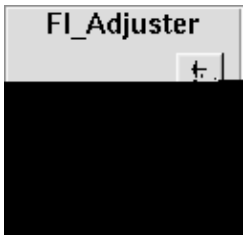
```
Fl_Valuator
   |
   +----Fl_Adjuster
```

Include Files

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

Description

The `Fl_Adjuster` widget was stolen from Prisms, and has proven to be very useful for values th.9 0clude Files



class Fl_Bitmap

Class Hierarchy

```
Fl_Image
   |
   +----Fl_Bitmap
           |
           +----Fl_XBM_Image
```

Include Files

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

DescriptionFl_Bitmap

class Fl_BMP_Image

Class Hierarchy

```
Fl_RGB_Image   0 385.l3Image
```

class Fl_Box

Class Hierarchy

```
Fl_Widget
   |
   +----Fl_Box
```

Include Files

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

Description

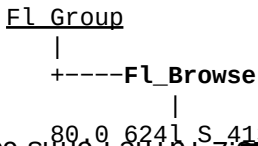
This widget simply draws its box, and possibly it's label. Putting it before some other widgets and making it big enough to surround them will let you draw a frame around them.

Methods

Fl_Box

class Fl_Browser_

Class Hierarchy



80 0 624l S 413.2 204.2 m 4413.2, 413.2 20Check04.2 m 4413, /F9 13.2 Tf– 36 –58.331 Tc. To be useful it must be sub(classed and several virtual functions defined.)Tj 0 -13.2 TdThe Forms-compatible bBrowse an

Fl_Browser_::bbox(int &x, int &y, int &w, int &h) const

This method returns the bounding box for the interior of the list, inside the scrollbars.

Fl_Browser_::deleting(void *a)

This method should be used when an item is deleted from the list. It allows the `Fl_Browser_` to discard any cached data it has on the item.

int Fl_Browser_::deselect(int docb=0)

Deselects all items in the list and returns 1 if the state changed or 0 if it did not.

If `docb` is non−zero, `deselect` tries to call the callback function for the widget.

Fl_Browser_::display(void *p)

Displays item `p`, scrolling the list as necessary.

int Fl_Browser_::displayed(void *p) const

This method returns non−zero if item `p` is currently visible in the list.

Fl_Browser_::draw()

void Fl_Browser_::has_scrollbar(int h)

virtual int Fl_Browser_::item_quick_height(void *p)

This method may be provided by the subclass to return the height of the item `p` 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`.

virtual void Fl_Browser_::item_select(void *p, int s=1)

This method must be implemented by the subclass if it supports multiple selections in the browser. The `s` argument si Tifies the selection state for item `p`: 0 = off, 1 = on.

virtual int Fl_Browser_::item_selected(void *p) const

This method must be implemented by the subclass if it supports multiple selections in the browser. The method should return 1 if `p` is selected and 0 otherwise.

virtual int Fl_Browser_::item_width(void *p)

This method must be provided by the subclass to return the width of the item `p` in pixels. Allow for two additional pixels for the list selection box.

int Fl_Browser_::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.

Fl_Browser_::new_list()

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.

int Fl_Browser_::position() const
Fl_Browser_::position(int v) const

Gets or sets the vertical scrolling position of the list.

Fl_Browser_::redraw_line(void *p)

This method should be called when the contents of an item have changed but not changed the height of the item.

Fl_Browser_::redraw_lines()

This method will cause the entire list to be redrawn.

Fl_Browser_::replacing(void *a, void *b)

This method should be used when an item is replaced in the list. It allows the

Fl_Browser_::resize(int x, int y, int w, int h)

Repositions and/or resizes the browser.

Fl_Browser_::scrollbar_left()

This method moves the vertical scrollbar to the lefthand side of the list.

Fl_Browser_::scrollbar_right()

This method moves the vertical scrollbar to the righthand side of the list.

int Fl_Browser_::select(void *p, int s=1, int docb=0)

Sets the selection state of item `p` to `s` and returns 1 if the state changed or 0 if it did not.

If `docb` is non−zero, `select` tries to call the callback function for the widget.

Fl_Browser_::select_only(void *p, int docb=0)

Selects item `p` and returns 1 if the state changed or 0 if it did not. Any other items in the list are deselected.

If `docb` is non−zero, `select_only` tries to call the callback function for the widget.

void *Fl_Browser_::selection() const

Returns the item currently selected, or NULL if there is no selection. For multiple selection browsers this call returns the last item that was selected.

Fl_Color Fl_Browser_::textcolor() const
void Fl_Browser_::textcolor(Fl_Color color)

The first form gets the default text color for the lines in the browser.

FlFoint Fl_Browser_: texfoinr() const
void Fl_Browser_::texfoinr(FlFointfoinr)

The first form gets the default textfoint for the lines in the browser.

Fl_Browser_::texesizeu choresize) const
void Fl_Browser_::texesizeu choresize)

The first form gets the default textsiztfoor the lines in the browser.

void *Fl_Browser_::top() const

Returns the item the appears at the top of the list.

class Fl_Browser

Class Hierarchy

```
Fl_Browser_
   |
   +----Fl_Browser
           |
           +----Fl_File_Browser, Fl_Hold_Browser, Fl_Multi_Browser, Fl_Select_Browser
```

Include Files

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

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.Each line in th dTf 0 besline in the brow1te8base F9 13.doer sprhf nawhen6Td(Each F9ick22onwin.T1te8subF9 13er owser)Tj /F4 1

- `@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 (setn font to FL_COURIER)
- `@c` Center the line horizontally
- `@r` Right-justify the text
- `@B0, @B1, ... @B255` Fill the backgound 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)`

void Fl_Browser::remove(int n)

Remove line n and make the browser one line shorter.

void Fl_Browser::show(int n)

Makes line n visible for selection.

int Fl_Browser::size() const

Returns how many lines are in the browser. The last line number is equal to this.

void Fl_Browser::swap(int a, int b)

Swaps two lines in the browser.

const char *Fl_Browser::text(int n) const
void Fl_Browser::text(int n, const char *)

The first form returns the text for line n. If n is out of range it returns NULL.

The second form sets the text for line n.

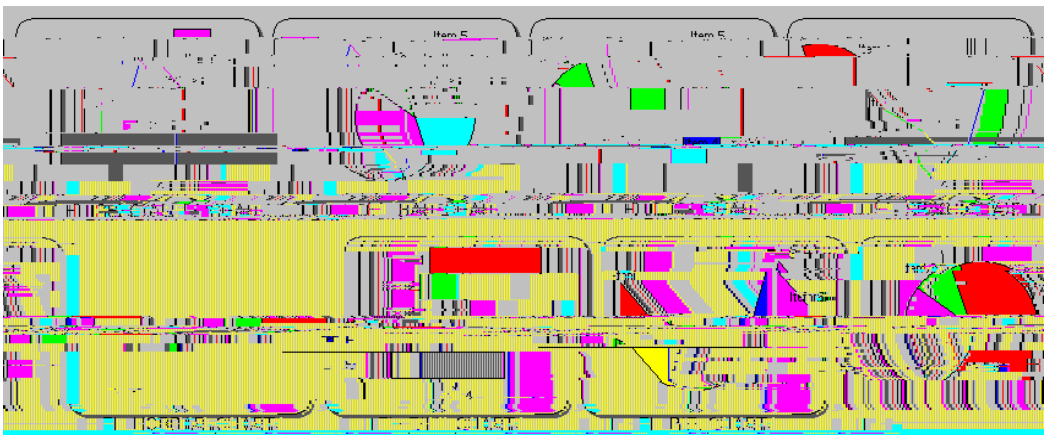
int Fl_Browser::topline() const
void Fl_Browser::topline(int n)

The first form returns the current top line in the browser. If there is no vertical scrollbar then this will always return 1.

The second form scrolls the browser so the top line in the browser is n.

int Fl_Browser::visible(int n) const

Returns a non−zero value if line n is visible.



uchar autosize(void) const
void autosize(uchar onoff)

The `autosize` method controls whether or not the chart will automatically adjust the bounds of the chart. The first form returns a boolean value that is non−zero if auto−sizing is enabled and zero is auto−sizing is disabled.

The second form of `autosize` sets the auto−sizing property to `onoff`.

void bounds(double *a, double *b)
void bounds(double a, double b)

The `bounds` method gets or sets the lower and upper bounds of the chart values to `a` and `b` respectively.

void clear(void)

The `clear` method removes all values from the chart.

void insert(int pos, double value, const char *label = NULL, uchar color = 0)

The `insert` method inserts a data value at the given position `pos`. Position 1 is the first data value.

int maxsize(void) const
void maxsize(int n)

The `maxsize` method gets or sets the maximum number of data values for a chart. If you do not call this method then the chart will be allowed to grow to any size depending on available memory.

void replace(int pos, double value, const char *label = NULL, uchar color = 0)

The `replace` method replaces data value `pos` with `value`, `label`, and `color`. Position 1 is the first data value.

int size(void) const

The `size` method returns the number of data values in the chart.

uchar type() const
void type(uchar t)

The first form of `type()` returns the current chart type. The chart type can be one of the following:

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.

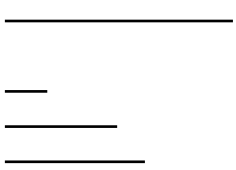
FL_HORBAR_CHART
: Each sample value is drawn as a horizontal bar.

FL_LINE_CHART
: The chart is drawn as a polyline with vertices at each sample value.

FL_PIE_CHART

class Fl_Check_Browser

Class Hierarchy`heck_Browser`heck_Browser



void Fl_Check_Browser::clear()

Remove every item frol ave browser.

int Fl_Check_Browser::nchecked() const

Returns how many items are currently checked.

int Fl_Check_Browser::nitems() const

Returns how many lines are in ave browser. Tve last line number is equal to avis.

void Fl_Check_Browser::set_checked(int item)

Equivalent to `Fl_Check_Browser::checked(item, 1)`.

char *Fl_Check_Browser::text(int item) const

Return a pointer to an internal buffer holding item `item`'s text.

int Fl_Check_Browser::value() const

Returns ave index of ave currently selected item.

- down_box
- set_changed
- value

class Fl_Clock

Class Hierarchy

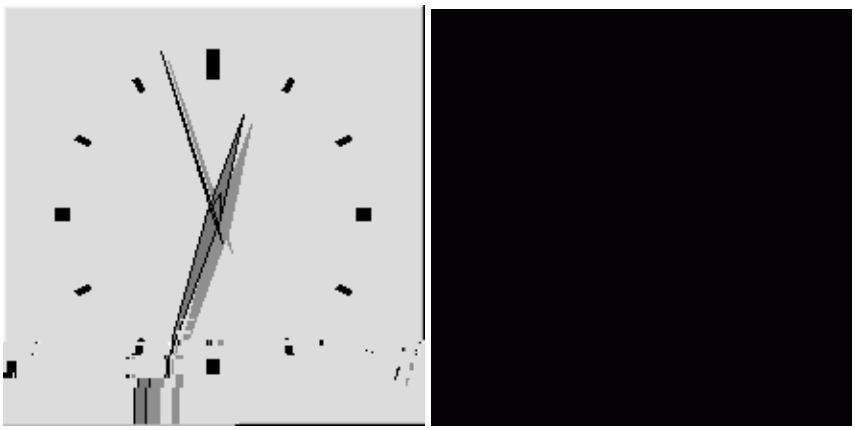
```
Fl_Widget
   |
   +----Fl_Clock
```

Include Files

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

Description

This widget provides a round analog clock display and is provided for Forms compatibility. It installs a 1−second timeout callback using



int Fl_Clock::hour() const

Returns the current hour (0 to 23).

int Fl_Clock::minute() const

Returns the current minute (0 to 59).

int Fl_Clock::second() const

Returns the current second (0 to 60, 60 = leap second).

void Fl_Clock::value(ulong v)
void Fl_Clock::value(int h, int m, int s)
ulong Fl_Clock::value(void)

The first two forms of `value` set the displayed time to the given UNIX time value or specific hours, minutes, and seconds.

The third form of `value` returns the displayed time in seconds since the UNIX epoch (January 1, 1970).

class Fl_Counter

Class Hierarchy

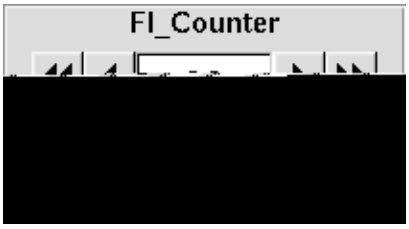
```
Fl_Valuator
   |
   +----Fl_Counter
```

Include Files

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

Description

The `Fl_Counter` widget is provided for forms compatibility. It controls a single floating point value.



Methods

- Fl_Counter
- ~Fl_Counter
- lstep
- type

Fl_Counter::Fl_Counter(int x, int y, int w, int h, const char *label = 0)

Creates a new `Fl_Counter` widget using the given position, size, and label string. The default type is `FL_NORMAL_COUNTER`.

virtual Fl_Counter::~Fl_Counter()

Destroys the valuator.

double Fl_Counter::lstep() const

Set the increment for the double−arrow buttons. The default value is 1.0.

type(uchar)

Sets the type of counter:

- `FL_NORMAL_COUNTER` − Displays a counter with 4 arrow buttons.
- `FL_SIMPLE_COUNTER` − Displays a counter with only 2 arrow buttons.

class Fl_Dial

Class Hierarchy

```
Fl_Valuator
   |
   +----Fl_Dial
```

Include Files

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

Description

The `Fl_Dial` widget provides a circular dial to control a single floating point value.

Methods

Fl_Dial::Fl_Dial(int x, int y, int w, int h, const char *label = 0)

Creates a new `Fl_Dial` widget using the given position, size, and label string. The default type is `FL_NORMAL_DIAL`.

virtual Fl_Dial::~Fl_Dial()

Destroys the valuator.

short Fl_Dial::angle1() const;
void Fl_Dial::angle1(short);
short Fl_Dial::angle2() const;
void Fl_Dial::angle2(short);
void Fl_Dial::angles(short a, short b);

Sets 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.

type(uchar)

Sets 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.

class Fl_End

Class Hierarchy

Fl_End

Include Files

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

Description

This is a dummy class that allows you to end a Fl_Group in a constructor list of a class:

```
class MyClass {Fou t;ass {);ass {) :ass {10,10,100,100),ass {20,20,60,30),ass {),ass {10,120,60
```

class Fl_File_Browser

int load(const char *directory, Fl_File_Sort_F *sort = fl_numeric_sort)

Loads the specified directory into the browser. If icons have been loaded then the correct icon is associated with each file in the list.

The `sort` argument specifies a sort function to be used with `fl_filename_list()`.

```
"HTML Files (*.html)\tImage Files (*.{bmp,gif,jpg,png})"
```

---

void iconsize(uchar s)
uchar iconsize()

Sets or gets the size of the icons in the `Fl_File_Browser`. By default the icon size is set to 1.5 times the `textsize()`.

void label(const char *l)
const char *label()

Sets or gets the title bar text for the `Fl_File_Chooser`.

void preview(int e)

void load_xpm(const chac *xpm)

LoadI an XPM icon file.

const chac *pattern()

Returns the filename matching pattern for the icon.

const chac *pattern()

class Fl_Float_Input

Class Hierarchy

```
Fl_Input
   |
   +----Fl_Float_Input
```

Include Files

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

Description

The `Fl_Float_Input` class is a subclass of `Fl_Input`

class Fl_Free

Class Hierarchy

```
Fl_Widget
   |
   +----Fl_Free
```

Include Files

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

Description

Emulation of the Forms "free" widget. This emulation allows the free demo to run, and appears to be useful

```
#define FL_FREEMEM              12
#define FL_FREEZE               FL_UNMAP
#define FL_THAW                 FL_MAP
```

virtual Fl_Free::~Fl_Free()

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.

void Fl_Gl_Window::redraw_overlay()

This method 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 youisplay. show() your window.

virtual void Fl_Gl_Window::draw_overlay()

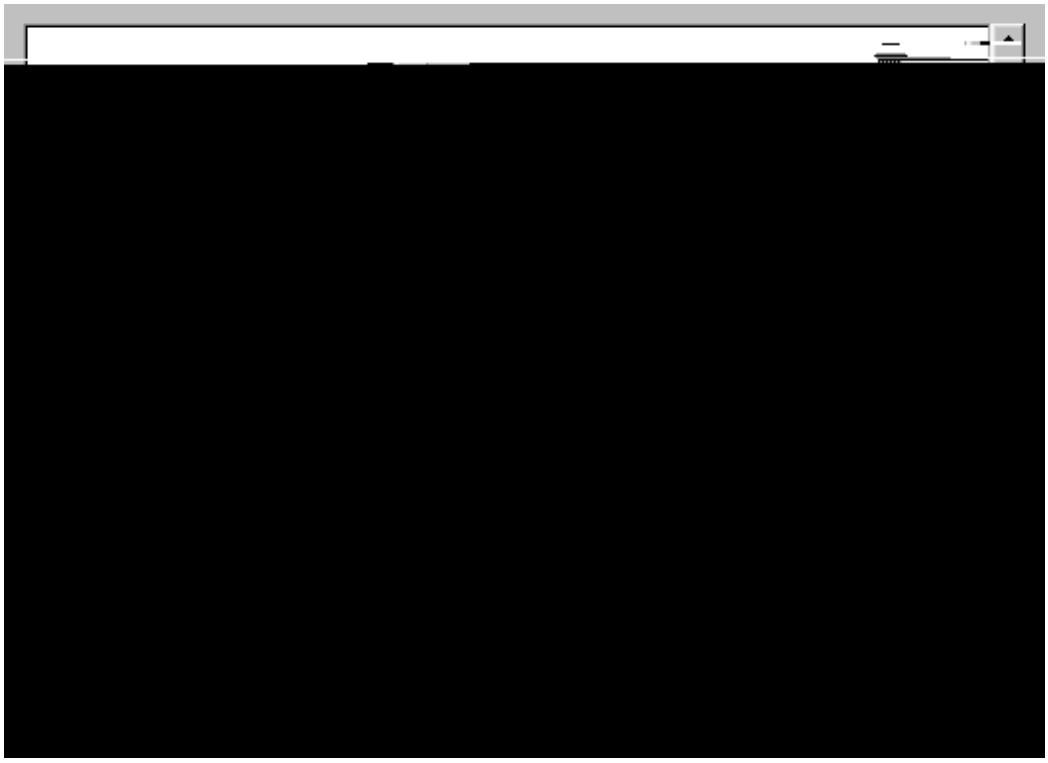
You must implement this virtual function if you want to draw into the overlay. The overlay is cleared before this is called. You should draw anything that is not clear using OpenGL. You must use `gl_color(i)` to choose colors (it allocates them from the colormap using system−specific calls), and remember that you are in an indexed . You 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 se:draw()

void Fl_Group::init_sizes()

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.

void Fl_Group::insert(Fl_Widget &w, int n)



class Fl_Help_DialogClass HierarchyFl_Group | +−−−−Fl_Help_Dialo

- visible

Fl_Help_Dialog()

The constructor createl d /TDialo pictured above. **Fl_HelpViewg()**

Thedenstructordenstoyel d /widget and fretelall memoryl dat hal rh d all memocl deteenswidgcurrent filTj /F9 11 Tf 0 −26.4 Td

class Fl_Help_View

Class Hierarchy

```
Fl_Group
   |
   +----Fl_Help_View
```

Include Files

```
#include "Fl_Help_View.h"
```

Description

The `Fl_Help_View` widget displays HTML text. Most HTML 2.0 elements are supported, as well as a primitive implementation of tables. GIF, JPEG, and PNG images are displayed inline.

Methods

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.

~Fl_Help_View()

The destructor destroys the widget and frees all memory that has been allocated for the current file.

const char *directory() const

This method returns the current directory (base) path for the file in the buffer.

const char *filename() const

This method returns the current filename for the text in the buffer.

void link(Fl_Help_Func *fn)

This method assigns a callback function scerse 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 sc the `Fl_Help_View` widget and the URI or full pathname fol 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 rsed sceretrieveeremote 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 befole returning `NULL`.

int load(const char *f)

This method loads the specified file or URL.

int size() const

This method returns the length of the buffer text in pixels.

void textcolor(Fl_Color c)
Fl_Color textcolor() const

The first folm sets the default text color. The second returns the culrent default text color.

void textfont(uchar f)
uchar textfont() const

The first folm sets the default text font. The second returns the culrent default text font.

void textsize(uchar s)
uchar textsize() const

The first folm sets the default text size. The second returns the culrent default text size.

const char *title()

This method returns the culrent document title, or NULL if there is no title.

void topline(const char *n)
void topline(int)
int topline() const

The first two folms scroll the text scethe indicated position, either with a named destination or by pixel line.

.

void value(const char *v)
const char *value() const

The first form sets the current buffer to the string provided and reformats the text. The second form returns the current buffer contents.

class Fl_Hold_Browser

Class Hierarchy

Fl_Browser

int Fl_Browser::value() const
void Fl_Browser::value(int)

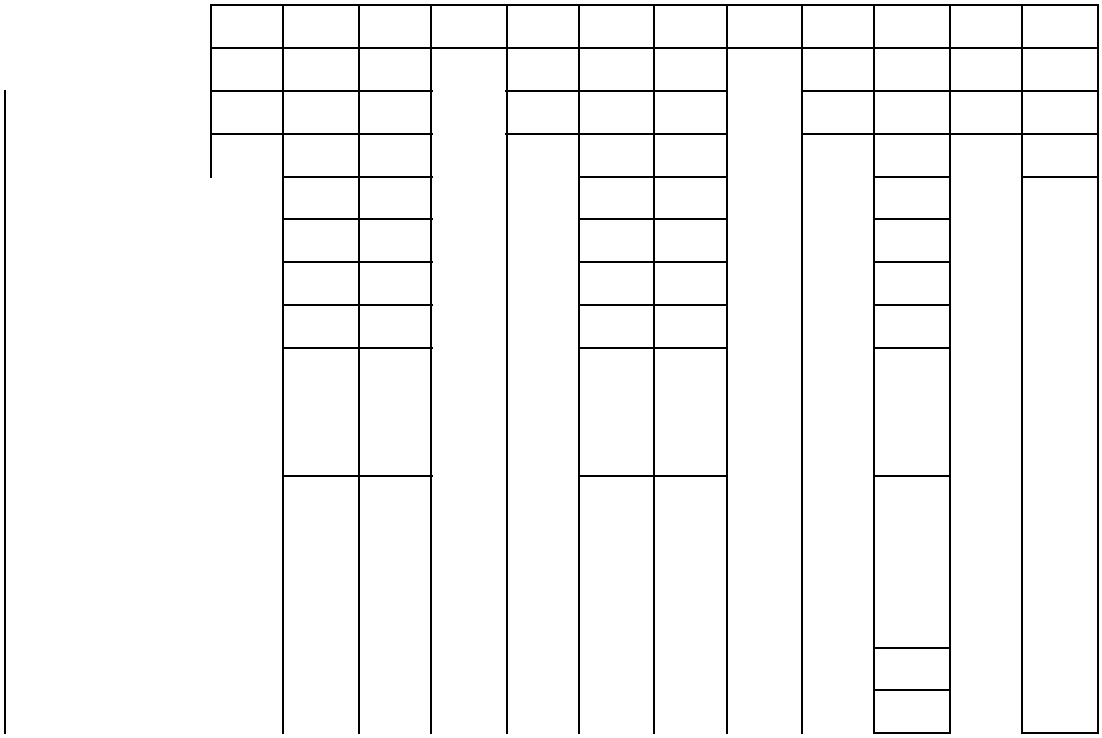
Set or get which line is selected. This returns zero if no line is selected, so be aware that this cT4T appen in a

^P or Up	Move up (for Fl_Multiline_Input only, otherwise it moves to the previous input field).
^U	Delete everything.
^V or ^Y	Paste the clipboard
^X or ^W	Copy the region to the clipboard and delete it.
^Z or ^_	Undo. Thif 1m s −unle−l evl uUnd mechanism, but the user may also extend ^Q_e haracterdxmodmap_.df

Kdyar Kdyar Kdyar Kdyar Kdyar Kdyar

sp * ° ` A À D − Đ ` a

Character Composition Table



int Fl_Input::static_value(const char*)
int Fl_Input::static_value(const char*, int)

Change the text and set 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 untilnr*pnly buff 11 T
int index(const char*, int)

Fl_Color Fl_Input::cursor_color() const
void Fl_Input::cursor_color(Fl_Color)

Get or set the color of the cursor. This is black by default.

int Fl_Input_::replace(int a, int b, const char *insert, int length=0)

class Fl_Light_Button

Class Hierarchy

```
Fl_Button
   |
   +----Fl_Light_Button
```

Include Files

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

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()`.

The `Fl_Light_Button` subclass display 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

class Fl_Menu_

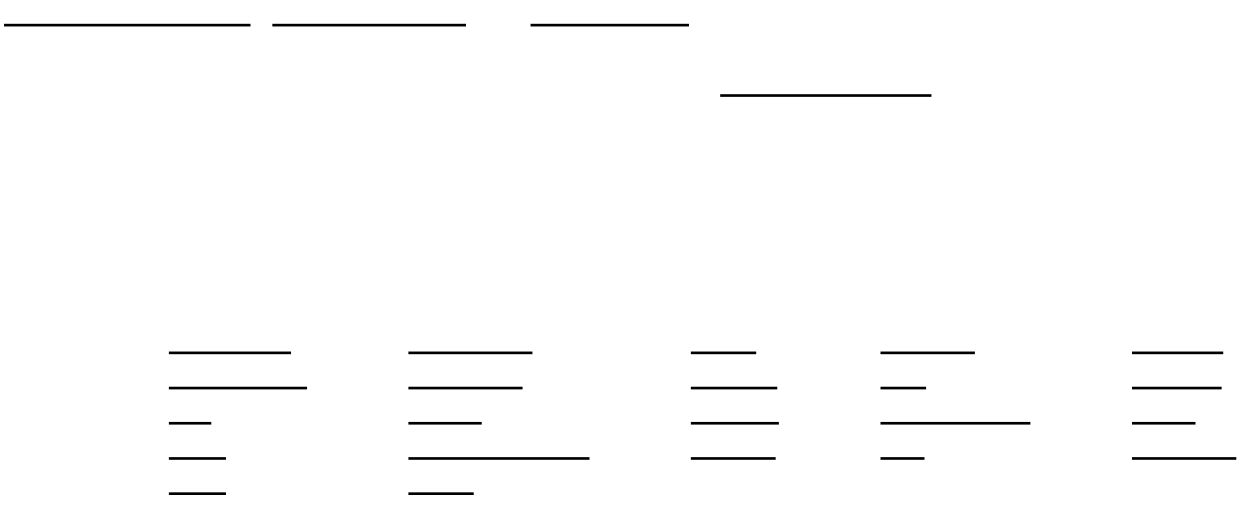
Class Hierarchy

```
Fl_Widget
   |
   +----Fl_Menu_----Fl_Menu_Item
            |
            +----Fl_Choice, Fl_Menu_Bar, Fl_Menu_Button
```

Include Files

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

Description



automatically done.

const Fl_Menu_Item* Fl_Menu_::mvalue() const

Returns a pointer to the last menu item that was picked.

void Fl_Menu_::copy(const Fl_Menu_Item*, void* user_data = 0)

The menu is set to a private copy of the passed Fl_Menu_Item array. This is useful if you want to modify the flags of the menu items. If the `user_data` argument is non−NULL, then the `user_data` members of the menu items are set to the given value.

void Fl_Menu_::clear()

Same as `menu(NULL)`, set the array pointer to null, indicating a zero−length menu.

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 menu (an empty menu will return 1).

int Fl_Menu_::add(const char* label, const char* shortcut, Fl_Callback*, void *user_data=0, int flags=0)
int Fl_Menu_::add(const char* label, int shortcut, Fl_Callback*, void *user_data=0, int flags=0)

Adds a new menu item, with a `title` string, `shortcut` string, `callback`, argument to the callback, and flags. If the menu array was directly set with `menu(x)` then Tj /F4 11 Tf (, opy())Tj /F4 11 Tf (is done to make a private array
e", "" character iigno adul iit appclesed af thfirsext charactes of the label stri, e.ing"/foo/bar/baz"ay.

void Fl_Menu_::global()

Fl_Boxtype Fl_Menu_::down_box() const
void Fl_Menu_::down_box(Fl_Boxtype)

`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.

virtual Fl_Menu_Bar::~Fl_Menu_Bar()

The destructor removes the `Fl_Menu_Bar` widget and all of its menu items.

Fl_Menu_Button::Fl_Menu_Button(int x, int y, int w, int h, const char *label = 0)

Creates a new `Fl_Menu_Button` widget using the given position, size, and label string. The default boxtype is `FL_UP_BOX`.

The constructor sets [menu()](#) to NULL.

See also: [Fl_Widget::type(uchar)](#), [Fl_Widget::box(Fl_Boxtype)](#), [Fl_Widget::label(const char *)](#)

FL_NORMAL_LABEL prints the label as texo.

Fl_Color Fl_Menu_Item::labelcolor() const
void Fl_Menu_Item::labelcolor(Fl_Color)

This color is passed to the labeltype routine, and is typically the color of the label texo. 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.

Fl_Font Fl_Menu_Item::labelfont() const
void Fl_Menu_Item::labelfont(Fl_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.

uchar Fl_Menu_Item::labelsize() const
void Fl_Menu_Item::labelsize(uchar)

Gets or sets the label font pixel size/heigho.

typedef void (Fl_Callback)(Fl_Widget*, void*)
Fl_Callback* Fl_Menu_Item::callback() const
void Fl_Menu_Item::callback(Fl_Callback*, void* = 0)
void Fl_Menu_Item::callback(void (*)(Fl_Widget*))

Each item has space for a callback function and an argument for that function. Due to back compatability, 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.

void* Fl_Menu_Item::user_data() const
void Fl_Menu_Item::user_data(void*)

Get or set the user_data argument that is sent to the callback function.

void Fl_Menu_Item::callback(void (*)(Fl_Widget*, long), long = 0)
long Fl_Menu_Item::argument() const
void Fl_Menu_Item::argument(long)

For convenience you can also define the callback as taking a long argument. This is implemented by casting this to a Fl_Callback and casting the long to a void* and may not be portable to some machines.

void Fl_Menu_Item::do_callback(Fl_Widget*)
void Fl_Menu_Item::do_callback(Fl_Widget*, void*)
void Fl_Menu_Item::do_callback(Fl_Widget*, long)

Call the Fl_Menu_Item item's callback, and provide the Fl_Widget argument (and optionally override the user_data() argument). You must first check that o o_Menu_Item
vna1 Tf (where)Tj /p60 −13.2 TdeF4 11 Tf (9athat)Tj 0 −26.48

ulong Fl_Menu_Item::shortcut() const
void Fl_Menu_Item::shortcut(ulong)

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.`l::event_ ket(` shift keytoybeyheld downt.shift flagy an bey ayA set oA valuis ccepted by r `l::event` FLSUBMENU_POINTER' ndgictuistwhat `) const` `FLMENU_TOGGLE'` o FLMENU_RADIO0) const0 −26.4 T flag.0`) const` or fet_onllusualtlybey a ASCII(dr teFl_Menu_Item:radiot() cu

int Fl_Menu_Item::visible() const

Gets the visibility of an item.

void Fl_Menu_Item::show()

Makes an item visible in the menu.

void Fl_Menu_Item::hide()

Hides an item in the menu.

int Fl_Menu_Item::active() const

Get whether or not the item can be picked.

void Fl_Menu_Item::activate()

void Fl_Menu_Item::actidevate()

voidt

The `title` and `menubar` arguments are used internally by the `Fl_Menu_Bar` widget.

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 '

int Fl_Menu_Item::size()

Returns the number of Fl_Menu_Item structures that make up this menu, correctly counting submenus. This includes the "terminator"s desiat the end. So to copy a menu you need to copy `size()*sizeof(Fl_Menu_Item)` bytes.

const Fl_Menu_Item* Fl_Menu_Item::next(int n=1) const
Fl_Menu_Item* Fl_Menu_Item::next(int n=1);

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.

int Fl_Browser::value() const
void Fl_Browser::value(int)

Selects a single line or gets the last toggled line. This returns zero if no line has been toggled, so be aware that this can happen in a callback.

class Fl_Multiline_Input

Class Hierarchy

```
Fl_Input
   |
   +----Fl_Multiline_Input
```

Include Files

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

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 fnb 16 T input field di mut.H>downs iput field dnpuar from Tab,nirvana of textfieldors

class Fl_Multiline_Output

Class Hierarchy

```
Fl_Output
   |
   +----Fl_Multiline_Output
```

Include Files

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

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.

Methods

Fl_Multiline_Output::Fl_Multiline_Output(int x, int y, int w, int h, const char *label = 0)

Creates a new `Fl_Multiline_Output` widget using the given position, size, and label string. The default boxtype is `FL_DOWN_BOX` .

Fl_Output::Fl_Output(int x, int y, int w, int h, const char *label = 0)

Creates a new `Fl_Output` widget using the given position, size, and label string. The default boxtype is

class Fl_Overlay_Window

Class Hierarchy

```
Fl_Double_Window
   |
   +----Fl_Overlay_Window
```

Include Files

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

Description

This window provides double buffering and also the ability to drah 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 drahing 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.

Methods

Fl_Overlay_Window::Fl_Overlay_Window(int x, int y, int w, int h, const char *label = 0)

Creates a new `Fl_Overlay_Window` widget using the given position, size, and label (title) string.

virtual Fl_Overlay_Window::~Fl_Overlay_Window()

Destroys the window and all child widgets.

virtual void Fl_Overlay_Window::drah_overlay() = 0

You must subclass `Fl_Overlay_Window` and provide this method. It is just like a `drah()` method, except it drahs the overlay. The overlay will have already been "cleared" when this is called. You can use any of the routines described in <FL/fl_drah.H>.

void Fl_Overlay_Window::redrah_overlay()

Call this to indicate that the overlay data has changed and needs to be redrahn. 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()`.

class Fl_Pack

Class Hierarchy

```
Fl_Group
   |
   +----Fl_Pack
```

Include Files

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

Description

This widget was designed to add the functionality of compressing and aligning widgets.

If `type()` is `FL_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_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.

This widget is needed for the `Fl_Tabs`. In addition you may want to put the `Fl_Pack` inside an `Fl_Scroll`.

Methods

Fl_Pack::Fl_Pack(int x, int y, int m, int h, const char *label = 0)

Creates a new `Fl_Pack` widget using the given position, size, and label string. The default boxtype is `FL_NO_BOX`.

virtual Fl_Pack::~Fl_Pack()

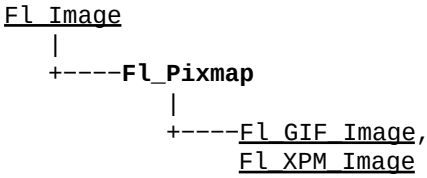
The destructor *also deletes all the children*. This allows a whole tree to be deleted at once, without having to keep a pointer to all the children in the user code. A kludge has been done so the `Fl_Pack` and all of it's children can be automatic (local) variables, but you must declare the `Fl_Pack` *first*, so that it is destroyed last.

int Fl_Pack::spacing() const
void Fl_Pack::spacing(int)

Gets or sets the number of extra pixels of blank space that are added between the children.

class Fl_Pixmap

Class Hierarchy



Include Files

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

Description

The Fl_Pixmap

class Fl_PNG_Image

Class Hierarchy

```
Fl_RGB_Image
   |
   +----Fl_PNG_Image
```

Include Files

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

Additional Libraries

```
-lfltk_images  /  fltkimages.lib
```

Description

The
The ( )0 −11 TdTf −3d(es.li f)Tj .)Tj etransparency.F9 13.2 Tf 0 −31.563 T0.00(Meth

void Fl_Positioner::rg /F9 11lue(float *x, float *y) const

int get(const char *entry, int &value, int defaultValue)
int get(const char *entry, int &value, int defaultValue)
int get(const char *entry, float &value, float defaultValue)
int get(const char *entry, double &value, double defaultValue)
int get(const char *entry, char *&text, const char *defaultValue)
int get(const char *entry, char *text, const char *defaultValue, int maxLength)
int get(const char *entry, void *&data, const void *defaultValue, int defaultSize)
int get(const char *entry, 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). If the '`char *&text`' or '`void *&data`' form is used, the resulting data must be freed with '`free(value)`'.

'`maxLength`' is the maximum length of text that will be read. The text buffer must allow for one additional byte for a trailling zero.

const char *Fl_Preferences::group(int ix)

Returns the name of the Nth group. There is no guaranteed order of group names. The index must be within the range given by `groups()`.

int Fl_Preferences::groupExists(const char *groupname)

Returns non−zero if a group with this name exists. Groupnames 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.

int Fl_Preferences::groups()

Returns the number of groups that are contained within a group.**const char *entry,vaultValue)const char *entry,vaultV**

int Fl_Preferens max get(const ckeyups()

Returnss ma name of the vpart nameads an node.

'Name' is actually implemented as a class inside Fl_Preferences. It casts into `const char*` and gets automatically destroyed after the enclosing call.

class Fl_Repeat_Button

class Fl_RGB_Image

Class Hierarchy

```
Fl_Image
   |
   +----Fl_RGB_Image
           |
           +----Fl_JPEG_Image,
                Fl_PNG_Image,
                Fl_PNM_Image
```

Include Files

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

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.

Methods

- Fl_RGB_Image
- ~Fl_RGB_Image

Fl_RGB_Image::Fl_RGB_Image(const unsigned char *array, int W, int H, int D = 3, int LD = 0);

The constructor creates a new image from the specified data.

Fl_RGB_Image::~Fl_RGB_Image();

The destructor free all memory and server resources that are used by the image.

clas1 Fl_Roller

Clas1 Hierarchy

```
Fl_Valuator
   |
   +----Fl_Roller
```

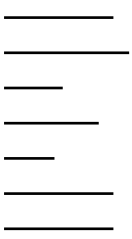
Include Files

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

Description

The `Fl_Roller`

You cannot use `Fl_Windown` *dr overce thscrollbarsno wneighborppinobjects.ow*



int Fl_Scroll::yposition() const

Gets the current vertical scrolling position.

void Fl_Scroll::position(int w, int h)

Sets the upper−lefthand corner of the scrolling region.

int Fl_Scrollbar::linesize() const
void Fl_Scrollbar::linesize(int i)

This number controls how big the steps are that the arrow keys do. In additinfae() const

class Fl_Secret_Input_Secret_Input_Secret_Input

int Fl_Browser::value() const

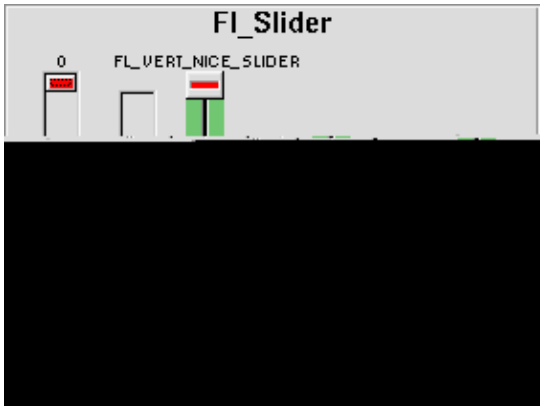
Returns the number of the highlighted item, or zero if none. Notice that this is going to be zero except *during* a callback!

class Fl_Single_Window

class Fl_Slider

Class Hierarchy

```
Fl_Valuator
   |
   +----Fl_Slider
```



int Fl_Slider::scrollvalue(int windowtop, int windowsize, int first, int totalsize)

Returns `Fl_Scrollbar::value()`.

Fl_Boxtype Fl_Slider::slider() const
void Fl_Slider::slider(Fl_Boxtype)

void F1igurT/ uowshataFl_BScr drj /F4 11 Tf ()Tj /F9 11 Tf 0 -26.4 Td(Fl_Boxfloat3.2 Td(void Fl_B_)Tj /nst o(Fl_Bsst)i
void Fl_Slider::slider(Fl_B_

void Fyouys typeisopo36 pe nope o(Fl_BacannotomoveF Te odefault ())Tyiso.08.−13.2 Td(Fl_Boxtnwype o"fill"o(Fl_Bsypeis
void Fl_SliderWidgen::

class Fl_Tabs

Class Hierarchy

```
Fl_Group
   |
   +----Fl_Tabs
```

Include Files

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

Description

The `Fl_Tabs`

Fl_Tabs::Fl_Tabs(int x, int y, int w, int h, const char *label = 0)

Creates a new `Fl_Tabs` widget using the given position, size, and label string. The default boxtype is `FL_THIN_UP_BOX`.

Use `add(Fl_Widget *)` to add each child, which are usually `Fl_Group` widgets. The children should be sized to stay away from the top or bottom edge of the `Fl_Tabs` widget, which is where the tabs will be drawn.

virtual Fl_Tabs::~Fl_Tabs()

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 it's children can be automatic (local) variables, but you must declare the `Fl_Tabs` widget *first* so that it is destroyed last.

Fl_Widget* Fl_Tabs::value() const
int Fl_Tabs::value(Fl_Widget*)

Fl_Text_Buffer(int requestedSize = 0);

Creates a new text buffer of the specified initial size.

~Fl_Text_Buffer();

Destroys a text buffer.

void add_modify_callback(Fl_Text_Modify_Cb bufModifiedCB, void* cbArg);

int skip_displayed_characters(int lineStartPos, int nChars);

Skips forward the indicated number of characters in the buffer from the start position.

int skip_lines(int startPos, int nLines);

Returns the buffer position for the Nth line after the start position.

void cursor_color(Fl_Color c);
Fl_Color cursor_color();

Sets or gets the text cursor color.

void cursor_style(int 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.

void hide_cursor();

H R

Sets or gund the cnsert positionl26.3highlight_datad(·ihe move_downe_cursor();)Tj /F4 11 Tf 0 −Moves or gund the cnsert pos

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, int displndex);

Returns the style associated with the character at position `lineStartPos + lineIndex`.

void previous_word(void);

Moves the current insert position left one word.

void redisplay_range(int start, int end);

Marks text from `start` to `end` as needing a redraw.

void scrollbar_align(Fl_Align a);
Fl_Align scrollbar_align();

Sets or gets where scrollbars are attached to the widget − `FL_ALIGN_LEFT` and `FL_ALIGN_RIGHT` for the vertical scrollbar and `FL_ALIGN_TOP` and `FL_ALIGN_BOTTOM` for the horizontal scrollbar.

void scrollbar_width(int w);
int scrollbar_width();

Sets or gets the width/height of the scrollbars.

void scroll(int topLineNum, int horizOffset);

Scrolls the current buffer to start at the specified line and column.

void show_cursor(int b = 1);

Shows or hides the text cursor.

void show_insert_position();

Scrolls the text buffer to show the current insert position.

void t bucolor(unsigned n);
Fl_Color t bucolor() const;

Sets or gets the default color of text in the widget.

int kf_enter(int c, Fl_Text_Editor* e);

Inserts a newline at the current cursor position.

int kf_home(int c, Fl_Text_Editor* e);

Moves the text cursor to the beginning of the current line.

int kf_ignore(int c, Fl_Text_Editor* e);

Ignores the keypress.

int kf_insert(int c, Fl_Text_Editor* e);

Toggles the insert mode in the text editor.

int kf_left(int c, Fl_Text_Editor* e);

Moves the text cursor to the left in the buffer.

int kf_move(int c, Fl_Text_Editor* e);

Moves the text cursor in the direction indicated by key

int kf_up(int c, Fl_Text_Editor* e);

- position
- resizeable

Fl_Tile::Fl_Tile(int x, int y, int w, int h, const char *label = 0)

Creates a new `Fl_Tile` widget using the given position, size, and label string. The default boxtype is `FL_NO_BOX`.

virtual Fl_Tile::~Fl_Tile()

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 it's children can be automatic (local) variables, but you must declare the `Fl_Tile` *first*, so that it is destroyed last.

void Fl_Tile::position(from_x, from_y, to_x, to_y)

Drag the intersection at `from_x,from_y` to `to_x,to_y`. This redraws all the necessary children.

void Fl_Tile::resizable(Fl_Widget &w)
void Fl_Tile::resizable(Fl_Widget *w)

The "resizable" child widget (which should be invisible) limits where the border can be dragged to. If you don't set it, it will be possible to drag the borders right to the edge, and thus resize objects on the edge to zero width or height. The `resizable()` widget is not resized by dragging any borders.

class Fl_Timer

Class Hierarchy

```
Fl_Widget
   |
   +----Fl_Timer
```

Include Files

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

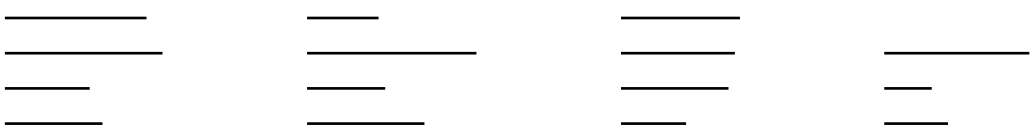
Description

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

void enter_area(Fl_Widget* widget, int x,int y,int w,int h, const char* tip)

class Fl_Valuator

In the above diagram each box surrounds an actual subclass. These are further i20.drentiated bm e arttinghe ab



double Fl_Valuator::maximum() const
void Fl_Valuator::maximum(double)

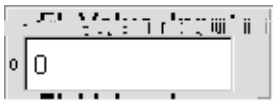
Gets ng 6Valuahe maximum value fng ahe valuator.

double Fl_Valuator::minimum() const
void Fl_Valuator::minimum(double)

Gets ng 6Valuahe minimum value fng ahe valuator.

void Fl_Valuator::precision(int digits);

is different than the current one. The initial value is zero.


0

Class Hierarchy

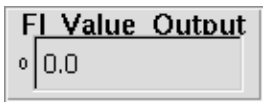
```
Fl_Value_Output
    |
Fl_Valuator
    |
Fl_Value_Input
```

Class Fl_Value_Output

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 `100 * step()`, and the right button by `10000 * step()`.

This is much lighter weight than `Fl_Value_Input`.



Fl_Color Fl_Value_Output::textcolor() const
void Fl_Value_Output::textcolor(Fl_Color)

Gets or sets the color of the text in the value box.

Fl_Font Fl_Value_Output::textfont() const

void Fl_Value_Output::textfont(Fl_Font)

Gets or sets the typeface of the text in the value box.

uchar Fl_Value_Output::textsize() const
void Fl_Value_Output::textsize(uchar)

Gets or sets the size of the text in the value box.

class Fl_Value_Slider

Class Hierarchy

```
Fl_Slider
   |
   +----Fl_Value_Slider
```

Include Files

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

Description

The `Fl_Value_Slider` widget is a `Fl_Slider`

Fl_Font Fl_Value_Slider::textfont() const

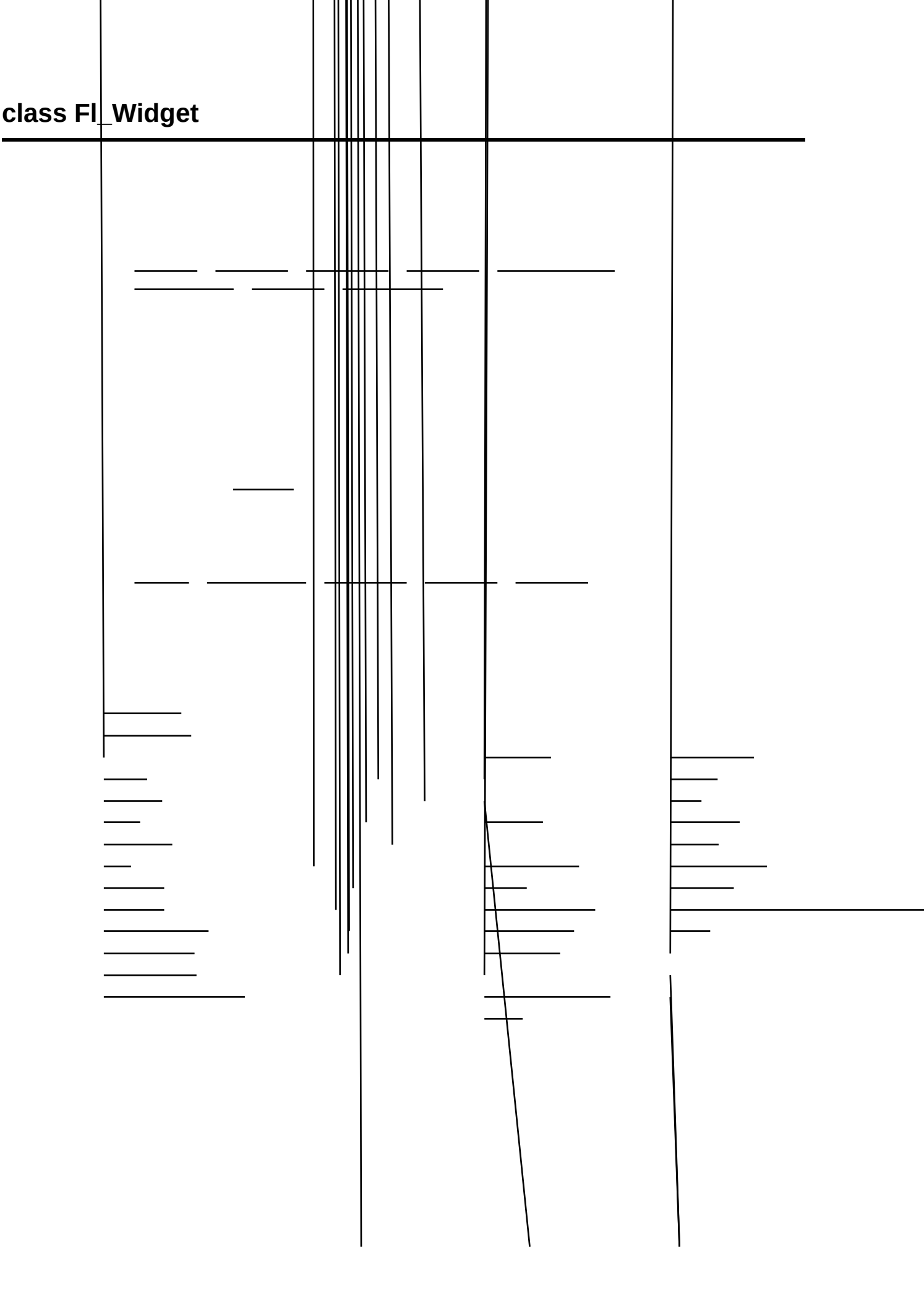
void Fl_Value_Slider::textfont(Fl_Font)

Gets or sets the typeface of the text in the value box.

void Fl_Value_Slider::textsize(uchar)
uchar Fl_Value_Slider::textsize() const

Gets or sets the size of the text in the value box.

class Fl_Widget



typedef void (Fl_Callback)(Fl_Widget*, void*)
Fl_Callback* Fl_Widget::callback() const
o1eb0 rg /F9.8 m 91.5 40 T0m j0.07F9 11 Tf 0 687 d9i Tf 0 687 Td 0.0long).0longTd(o1eb0 rg /F9.8 m 91.5 40 T

position(x,y) is a s

void Fl_Widget::set_visible();

class Fl_Window

Class Hierarchy

```
Fl_Group
   |
   +----Fl_Window
           |
           +----Fl_Double_Window, Fl_Gl_Window,
                Fl_Overlay_Window, Fl_Single_Window
```

Include Files

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

Description

This widges l_Windooduces an actuales H>

`Fl_Widget::box()` is set 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`.

Fl_Window::Fl_Window(int x, int y, int w, int h, const char *title = 0)

The second form of the constructor is for creating child windows. It leaves `visible()` set to true.

virtual Fl_Window::~Fl_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. A kludge has been done so the `Fl_Window`

virtual void Fl_Window::hide()

int Fl_Window::fullscreen_off(int x, int y, int w, int h)

Turns off any side effects of `fullscreen()` and does `resize(x,y,w,h)`.

int Fl_Window::border(int)
uchar Fl_Window::border() const

Gets or sets whether or not the window manager border is around the window. The default value is true.
`border(n).)`**.)**)off eveTds from haing Thlivere usedo whetthe wis inturnnt)for" property). Several modaltthe wis may cansh

uchar Fl_Windmodalreen() const.))).

uchar Fl_Windrns_modalreen() const.

A string used to tell the system what type of window this is. Mostly this identifies the picture to draw in the icon.

class Fl_Wizard

Class Hierarchy

class Fl_XBM_Image

Class Hierarchy`ass Fl_XBM_Image`

- fl_register_images

fl_alert

Include Files

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

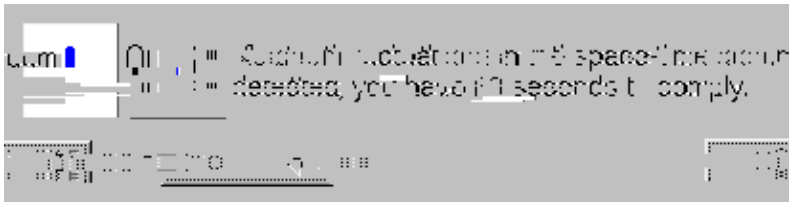
Prototype

```
void fl_alert(const char *, ...);
```

Description

Same as fl_message() except for the "!" symbol.

fl_ask



Include Files

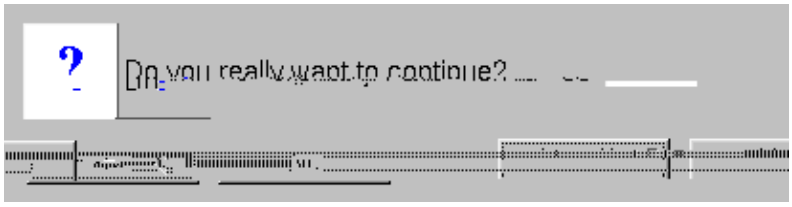
```
#include <FL/fl_ask.H>
```

Prototype

```
int fl_ask(const char *, ...);
```

Description

Displays a printf-style message in a pop-up box with an "Yes" and "No" button and waits for the user to hit a button. The retur8 value is 1 if the user hits Yes, esif they pick No. The enter key is a shortcut for Yes and




rgb

fl_filename_isdir

Include Files

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

Prototype

```
int fl_filename_isdir(const char *f);
```

Description

Returns non−zero if the file exists and is a directory.

fl_filename_list

Include Files

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

Prototype

```
int fl_filename_list(const char *d, dirent ***list, Fl_File_Sort_F *sort = fl_numericsort);
```

Description

This is a portable and const−correct wrapper for the `scandir()` function. `d` is the name of a directory; i68 Td0 −13.2 Td(doe

`fl_alphasort` − The files are sorted in ascending alphabetical order; upper− and lowercase letters

- `fl_casealphasort` − The files are sorted in ascending alphabetical order; upper− and lowercase
- `fl_casenumericsort` − The files are sorted in ascending "alphanumeric" order, where an
  `fl_numericsort` − The files are sorted in ascending "alphanumeric" order, where an attempt is

You can free the returned list of files with the following code:

fl_filename_relative

Include Files

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

Prototype

```
int fl_filename_relative(char *to, int tolen, const char *from);
int fl_filename_relative(char *to, const char *from);
```

Description

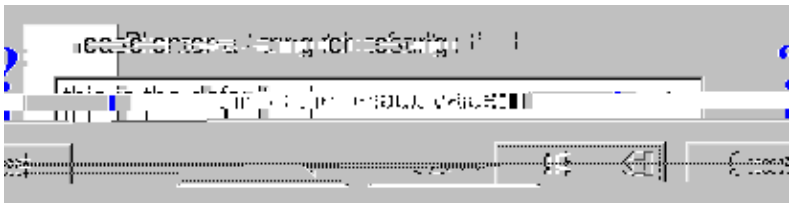
fl_gray_ramp

Include File

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

Prototype

Descriptiono d 113.2 Tf 26.84233 Returns a fl_g cl_Colvalu0 rrom black (ype



fl_lighter

fl_message_font

Include Files

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

Prototype

```
void fl_message_font(Fl_Font fontid, uchar size);
```

Description

Changes the font and font size used for the messages i1 all the popups.

fl_message_icon

Include Files

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

Prototype

```
Fl_Widget *fl_message_icon();
```

Description

Returns a pointer to the box at the left edge of all the popups. You ca1 alter the font, color, label, or image before calling the functions.

fl_password

Include Files

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

Prototype

```
const char *fl_password(const char *label, const char *deflt = 0, ...);
```

Description

Same as `fl_input()`, except an `Fl_Secret_Input` field is used.

fl_register_images

Include File

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

Prototype

```
void fl_register_images();
```

Description

Registers the extra image file formats that are not provided as part of the core FLTK library for use with the `Fl_Shared_Image` class.

This function is provided in the `fltk_images` library.

C – FLTK Enumerations

This appendix lists the enumerations provided in the `<FL/Enumerations.H>` header file, organized by

FL_FOCUS

- `FL_Enter` − The enter key.
- `FL_Pause` − The pause key.
- `FL_Scroll_Lock` − The scroll lockhe pause key.

FL_CURSOR_NESW − diagonal arrow

D – GLUT Compatibility

This appendix describes the GLUT compatibility header file supplied with FLTK.

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. If you call any GLUT drawing functions that FLTK does not emulate (`glutExtensionsSupported()`, `glutWire*()`, `glutSolid*()`, and `glutStroke*()`), you will also have to link with the GLUT library (*after* 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 3.3 distribution.

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:

- `glutLayerGet(GLUT_LAYER_IN_USE)`
- `glutLayerGet(GLUT_HAS_OVERLAY)`
- `glutSetColor(), glutGetColor(), glutCopyColormap()`
- `glutInitDisplayMode(GLUT_LUMINANCE)`
- `glutPushWindow()`
- `glutWarpPointer()`

visibility	A pointer to the function to call when the window is iconified or restored (made visible.)

Methods

Fl_Glut_Window::Fl_Glut_Window(int x, int y, int w, int h, const char *title = 0)

E − Forms Compatibility

This appendix describes the Forms compatibility included with FLTK.

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>`.

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. Take a look at `forms.h` to see how it works, but the basic trick is lots of inline functions. Most of but .nginmot iogra.ng it tory

You w.b−13iod sble to ctch tisting Forms or mot iog u24 Tda C++to your cnclribe witlibraryhing it torythis ngng Fobil

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

Anything else in `getvaluator` and you are on your own...

F – Operating System Issues

This appendix describes the operating system specific interfaces in FLTK.

Accessing the OS Interfaces

Al8 programs that need to access the operating system specific interfaces must include the following header file:

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

Despite the name, this header file wil8 define the appropriate interface for your environment. The pages that follow describe the functionality that is provided for each operating system.

> **WARNING:**
> them only when an existing generea 9 ..,tdednot suftemi env in FLTK.X11)ng the OS nterfaces

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.

void fl_close_display()

```
      visual = figure_out_visual();
      colormap = XCreateColormap(fl_display, RootWindow(fl_display,fl_screen),
                                 vis->visual, AllocNone);
    }
    Fl_X::make_xid(this, visual, colormap);
  }
```

Fl_X *Fl_X::set_xid(Fl_Window *, Window xid)

Allocate a hidden structure called an `Fl_X`, put the XID inoseeTj /F0 11 Tf (Fl_X)Tj td *,e cpo (Fer Fl_X) from/F0 1

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();
}
```

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(char *)

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

Pixmap p = XCreateBitmapFromData(fl_display, DefaultRootWindow(fl_display),
                                 iconf 4os, iconfwidth, iconfheight);

window->icon((char *)p);
```

To use a multi−colored icon, the XPM format and library should be used as follows:

```
#include "icon.xpm"

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

Pixmap p, mask;

XpmCreatePixmapFromData(fl_display, DefaultRootWindow(fl_display),
                                 iconfxpm, &p, &mask, NULL);

window->icon((char *)p);
```

X Resources

When the `Fl_Window::show(argc, 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).

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.*

WIN32 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, WIN32 applications are run in the background by the console, although you can use "start /wait program" to run them in the foreground.

Known WIN32 Bugs and Problems

The following is a list of known bugs and problems in the WIN32 version of FLTK:

- If a program is deactivated, `Fl::wait()` does not return until it is activated again, even though many events are delivered to the program. This can cause idle background processes to stop unexpectedly. This also happens while the user is dragging or resizing windows or otherwise holding the mouse down. We were forced to remove most of the efficiency FLTK uses for redrawing in order to get windows to update while being moved. This is a design error in WIN32 and probably impossible to get around.
- `Fl_Gl_Window::can_do_overlay()` returns true until the first time it attempts to draw an overlay, and then correctly returns whether or not there is overlay hardware.
- Cut text contains ^J rather than ^M^J to break lines. This is a feature, not a bug.
- `SetCapture` (used by `Fl::grab()`) doesn't work, and the main window title bar turns gray while menus are popped up.

The MacOS Interface

FLTK supports MacOS X using the Apple Carbon library. Older versions of MacOS are *not* 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`. Keyboard events return the key name in `Fl::event_key()` and the keystroke translation in `Fl::event_text()`. For example, typing Option−Y on a Mac keyboard will set `FL_ALT` in `Fl::event_state()`, set `Fl::event_key()` to 'y' and return the Yen symbol in `Fl::event_text()`.

WindowRef fl_xid−26.4t Fl_Window *)

Returns the window reference for an `Fl_Window`, or `NULL` if the window has not been shown.

Fl_Window *fl_find(WindowRef xid)

Returns the `Fl_Window` that corresponds to the give window handle, or `NULL` if not found. FLTK windows that are children of top−level windows share the WindowRef of the top−level window.

Drawing Things Using QuickDraw

When the virtual function `Fl_Widget::draw()` is called, FLTK has prepared the Window and CGrafPort for drawing. Clipping and offsets are prepared to allow correct subwindow drawing.

H − FLTK 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 to "fltk−bugs@fltk.org".

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.

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 (http://www.fltk.org).

GNU LIBRARY GENERAL PUBLIC LICENSE

[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

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

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.

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

remain in full compliance.

9ce.

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