

PyFTLK Manual

Michael Sheldon

October 20, 2005

Copyright

This document is licensed under the Creative Commons [Attribution-ShareALike 2.5 License](#).

Examples

All examples in this manual can be downloaded from:
<http://linux.mikeasoft.com/pyfltk/examples.tar.gz>.

Preface

This manual is written in the hope of expanding the usage of PyFLTK (and by extension FLTK), the Fast Light Toolkit bindings for the Python programming language. It is dedicated to all the hard working PyFLTK and FLTK developers who have put together such an amazing toolkit.

As projects work hard to provide low end computer systems for schools, developing countries and computer training centres the development of lightweight software to support these systems becomes increasingly important. PyFLTK makes this task much simpler, reducing the learning curve to creating such software whilst not sacrificing too much in the way of efficiency.

I'm probably woefully unqualified to be writing such a document, anyone more knowledgeable than myself is welcome to submit changes to webmaster@mikeasoft.com.

This manual is designed to be modular, so it should be possible to skip between sections depending on your interest; there's no need to read the whole manual from front to back unless you really want to. I shall be assuming some knowledge of Python and the object oriented programming model.

Contents

1 Quick Tutorials	5
1.1 Introduction	5
1.2 Windows	5
1.3 Adding Widgets	6
1.4 Callbacks	7
1.5 Labels	8
1.6 Text Input/Output	9
1.7 Styles & Colours	10
2 Comprehensive Tutorials	12
2.1 Introduction	12
2.2 Building A Calculator	12
2.3 OpenGL	12
3 Fluid & FLConvert	13
A API Reference	14
A.1 Classes	15
A.1.1 FL_Adjuster	15
A.1.2 FL_Bitmap	15
A.1.3 FL_BMP_Image	15
A.1.4 FL_Box	15
A.1.5 FL_Browser	15
A.1.6 FL_Browser_	15
A.1.7 FL_Button	15
A.1.8 FL_Chart	15
A.1.9 FL_Check_Browser	15
A.1.10 FL_Check_Button	15
A.1.11 FL_Choice	15
A.1.12 FL_Clock	15
A.1.13 FL_Color_Chooser	15
A.1.14 FL_Counter	15
A.1.15 FL_Dial	15
A.1.16 FL_Double_Window	15
A.1.17 FL_End	15
A.1.18 FL_File_Browser	15
A.1.19 FL_File_Chooser	15
A.1.20 FL_File_Icon	15
A.1.21 FL_File_Input	15
A.1.22 FL_Free	15
A.1.23 FL_GIF_Images	15
A.1.24 FL_GL_Window	15

A.1.25	FL_Group	15
A.1.26	FL_Help_Dialog	15
A.1.27	FL_Help_View	15
A.1.28	FL_Hold_Browser	15
A.1.29	FL_Image	15
A.1.30	FL_Input	15
A.1.31	FL_Input_	15
A.1.32	FL_Int_Input	15
A.1.33	FL_JPEG_Image	15
A.1.34	FL_Light_Button	15
A.1.35	FL_Menu_	15
A.1.36	FL_Menu_Bar	15
A.1.37	FL_Menu_Button	15
A.1.38	FL_Menu_Item	15
A.1.39	FL_Menu_Window	15
A.1.40	FL_Multi_Browser	15
A.1.41	FL_Multiline_Input	15
A.1.42	FL_Multiline_Output	15
A.1.43	FL_Output	15
A.1.44	FL_Overlay_Window	15
A.1.45	FL_Pack	15
A.1.46	FL_Pixmap	15
A.1.47	FL_PNG_Image	15
A.1.48	FL_PNM_Image	15
A.1.49	FL_Positioner	15
A.1.50	FL_Preferences	15
A.1.51	FL_Progress	15
A.1.52	FL_Repeat_Button	15
A.1.53	FL_RGB_Image	15
A.1.54	FL_Return_Button	15
A.1.55	FL_Roller	15
A.1.56	FL_Round_Button	15
A.1.57	FL_Scroll	15
A.1.58	FL_Scrollbar	15
A.1.59	FL_Secret_Input	15
A.1.60	FL_Select_Browser	15
A.1.61	FL_Single_Window	15
A.1.62	FL_Slider	15
A.1.63	FL_Tabs	15
A.1.64	FL_Text_Buffer	15
A.1.65	FL_Text_Display	15
A.1.66	FL_Text_Editor	15
A.1.67	FL_Tile	15
A.1.68	FL_Tiled_Image	15
A.1.69	FL_Timer	15
A.1.70	FL_Tooltip	15
A.1.71	FL_Valuator	15
A.1.72	FL_Value_Input	15
A.1.73	FL_Value_Output	15
A.1.74	FL_Value_Slider	15
A.1.75	FL_Widget	15
A.1.76	FL_Window	15
A.1.77	FL_XBM_Image	15
A.1.78	FL_XPM_Image	15

A.2	Functions	15
A.2.1	fl_alert	15
A.2.2	fl_ask	15
A.2.3	fl_beep	15
A.2.4	fl_choice	15
A.2.5	fl_color_average	15
A.2.6	fl_color_chooser	15
A.2.7	fl_color_cube	15
A.2.8	fl_contrast	15
A.2.9	fl_cursor	15
A.2.10	fl_darker	15
A.2.11	fl_dir_chooser	15
A.2.12	fl_file_chooser	15
A.2.13	fl_file_chooser_callback	15
A.2.14	fl_filename_absolute	15
A.2.15	fl_filename_expand	15
A.2.16	fl_filename_ext	15
A.2.17	fl_filename_isdir	15
A.2.18	fl_filename_list	15
A.2.19	fl_filename_match	15
A.2.20	fl_filename_name	15
A.2.21	fl_filename_relative	15
A.2.22	fl_filename_setext	15
A.2.23	fl_gray_ramp	15
A.2.24	fl_input	15
A.2.25	fl_lighter	15
A.2.26	fl_message	15
A.2.27	fl_message_font	15
A.2.28	fl_message_icon	15
A.2.29	fl_password	15
A.2.30	fl_register_images	15
A.2.31	fl_rgb_color	15
A.2.32	fl_show_colormap	15
B	Styles	16
B.1	Boxes	16
B.2	Frames	16

Chapter 1

Quick Tutorials

1.1 Introduction

In this sections we'll create a group of small programs to demonstrate the main features and design ideas behind PyFLTK. If you are brand new to PyFLTK you might find it advisable to read through these tutorials in order, since later tutorials will assume knowledge of these basics.

1.2 Windows

When creating any graphical program windows are pretty important, they give us a nice blank canvas to starting sticking widgets to. Creating a window in FLTK is pretty simple, take a look at the below example, each line is commented to show what's happening.

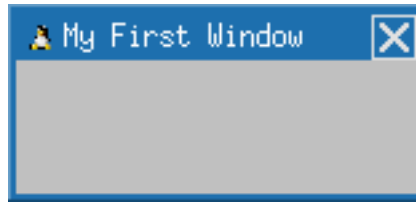
```
#!/usr/bin/python
#Give us access to FLTK
from fltk import *

#Create a class for our GUI
class MyApp:

    #Run these commands when the class is initialized
    def __init__(self):
        #Create a window 150 pixels wide and 50 pixels high
        self.window = Fl_Window(150, 50)
        #Set the window title
        self.window.label("My First Window")
        #Draw the window
        self.window.show()

#Create an instance of our GUI class
app = MyApp()
#Start the main FLTK loop
Fl.run()
```

Output:



The only command that needs further explanation here is *Fl.run()*. This command tells FLTK to go in to a loop, processing all graphical events. You need to make sure all your setup is done before this command.

1.3 Adding Widgets

A widget is any graphical element such as a window, a button, a text box, etc. Certain widgets (all children of *Fl_Group*) can have other widgets added to them, windows are just such a type of widget. There are two ways to add a widget to a window (or any other group), either during the creation stage of that window or later with the *add()* method.

```
#!/usr/bin/python
from fltk import *

class MyApp:

    def __init__(self):
        #Create our window
        self.window = Fl_Window(150, 60)
        self.window.label("Adding Widgets")

        #Create a button
        #This button is added to the window automatically
        self.button1 = Fl_Button(15, 0, 120, 20)
        self.button1.label("Creation Stage")

        #End the creation phase of the window
        self.window.end()
        self.window.show()

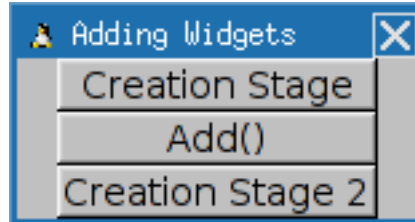
        #Create another button
        self.button2 = Fl_Button(15, 20, 120, 20)
        self.button2.label("Add()")
        #This one needs to be added manually
        self.window.add(self.button2)

        #Manually begin creation phase
        self.window.begin()
        #Add a third button
        self.button3 = Fl_Button(15, 40, 120, 20)
        self.button3.label("Creation Stage 2")
        #End creation phase
        self.window.end()

app = MyApp()
```

```
Fl.run()
```

Output:



As you can see any widgets created before the *self.window.end()* method are automatically added to *self.window*, whereas those after it have to be added manually. For the third button we manually start the creation phase again with *self.window.begin()*. You can add as many widgets as you like during the creation phase.

1.4 Callbacks

A pretty GUI isn't a lot of use if we have no way of receiving input from the user, this is where callbacks come in. You can register a particular method to be executed when some change occurs in a widget (e.g. a button being clicked), this is called a callback. This is done with the *callback()* method of the widget you want to monitor.

```
#!/usr/bin/python
from fltk import *

class MyApp:

    def __init__(self):
        #Create our window
        self.window = Fl_Window(150, 40)
        self.window.label("Callbacks")

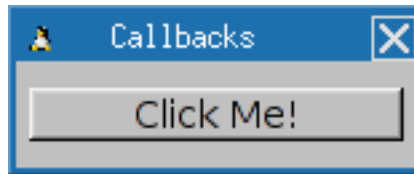
        #Create a button
        self.button1 = Fl_Button(5, 10, 140, 20)
        self.button1.label("Click Me!")
        #Create a callback to our 'clicked' function
        self.button1.callback(self.clicked)

        self.window.end()
        self.window.show()

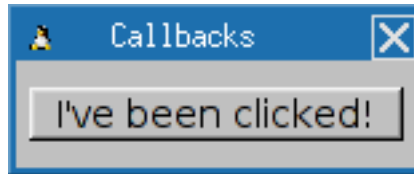
    #This function is called when button1 is clicked
    def clicked(self, widget):
        widget.label("I've been clicked!")

app = MyApp()
Fl.run()
```


Output: Before clicking



Output: After clicking



The callback method itself is fairly self explanatory, we just specify the method we want to be called, but the type of method we need to create needs a little more detail. Our method has to accept an argument for the widget that is calling it, this allows us to write a method that can handle many similar widgets. If we preferred however, we could rewrite this method to use `self.button1.label('I've been clicked')` instead of `widget.label()`, they both point to the same widget in the end.

1.5 Labels

FLTK doesn't have a concept of labels as separate entities like most toolkits, instead each FLTK widget has the opportunity to have a label associated with it through the `label()` method. With certain widgets these labels take on special properties, such as with windows where they become the window title. The following example shows the various methods used to control labels.

```
#!/usr/bin/python
from fltk import *

class MyApp:

    def __init__(self):
        #Create our window
        self.window = Fl_Window(240, 30)
        #This label is used for the window title
        self.window.label("Labels")

        #Create an input field
        self.input = Fl_Input(130, 5, 105, 20)
        #Set the label for the input field
        self.input.label("Some Input:")
        #Make the label shadowed
        self.input.labeltype(FL_SHADOW_LABEL)
        #Set the label size
        self.input.labelsize(18)
        #Set the label colour
```

```

self.input.labelcolor(fl_rgb_color(0, 0, 75))
#Set the label font
self.input.labelfont(FL_COURIER_ITALIC)
#Align the label to the left
self.input.align(FL_ALIGN_LEFT)

self.window.end()
self.window.show()

```

```

app = MyApp()
Fl.run()

```

Output:



The `fl_rgb_color()` function is explained at the end of the Styles & Colours section (1.7).

1.6 Text Input/Output

FLTK uses a system of buffers and displays for handling text. An *Fl_Text_Buffer* holds the actual text and formatting information while an *Fl_Text_Display* or an *Fl_Text_Editor* displays it. The following example shows both an *Fl_Text_Display* and an *Fl_Text_Editor* using the same *Fl_Text_Buffer* to illustrate their independence of one another.

```

#!/usr/bin/python
from fltk import *

class MyApp:

    def __init__(self):
        #Create our window
        self.window = Fl_Window(200, 185)
        self.window.label("Text Input/Output")

        #Create a text buffer, this stores the actual text
        self.textbuffer = Fl_Text_Buffer()
        #Add some text to it
        self.textbuffer.text("All the cool kids use FLTK.")

        #Create a text display, this shows the text but doesn't allow editing
        self.textdisplay = Fl_Text_Display(5, 20, 190, 70)
        #Set textdisplay to use our buffer
        self.textdisplay.buffer(self.textbuffer)
        self.textdisplay.label("Text Display")

        #Create a text editor, this shows the text and allows editing

```

```

self.textedit = Fl_Text_Editor(5, 110, 190, 70)
#Set textedit to use the same buffer as the display
self.textedit.buffer(self.textbuffer)
self.textedit.label("Text Editor")

self.window.end()
self.window.show()

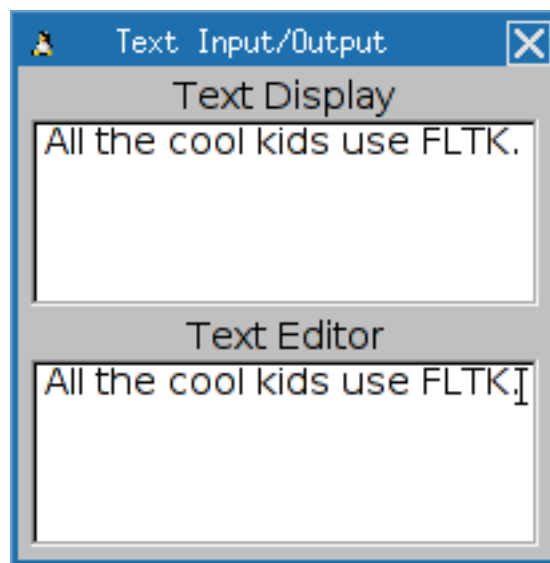
```

```

app = MyApp()
Fl.run()

```

Output:



If you try editing the text in the editor section you'll see it also changes in the display section, this is because the text is being changed in the buffer we created and they both share this.

1.7 Styles & Colours

For most purposes the default look is perfectly good and it's probably a good idea to stick with this in preparation for a shift to FLTK2 which supports user defined themes, but sometimes it's necessary to make something look distinctly different for one reason or another. FLTK supports a range of different styles that can be applied to most widgets, as well as allowing for customisation of colours. In my opinion the prettiest of these is the plastic style, as shown in the following example.

```

#!/usr/bin/python
from fltk import *

class MyApp:

    def __init__(self):
        #Create our window

```

```

self.window = Fl_Window(150, 55)
self.window.label("Styles")

#Create a normal looking button
self.button1 = Fl_Button(5, 5, 140, 20)
self.button1.label("Normal Style")

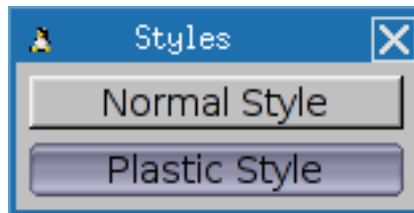
#Create a plastic looking button
self.button2 = Fl_Button(5, 30, 140, 20)
self.button2.label("Plastic Style")
#Set the box style
self.button2.box(FL_PLASTIC_UP_BOX)
#Set the main colour
self.button2.color(fl_rgb_color(0, 0, 100))
#Set the colour when being clicked
self.button2.selection_color(fl_rgb_color(50, 50, 150))

self.window.end()
self.window.show()

app = MyApp()
Fl.run()

```

Output:



As shown above we set the style with the *box()* method of a widget. For changing colours the *fl_rgb_color()* function is very useful, it converts RGB (Red, Green, Blue) values to FLTK colours, be warned that at the time of writing this function has a bug making values greater than 127 in the red channel cause errors.

As well as a range of box styles being available there are a group of frame styles (changed with the *frame()* method) which can be mixed with the box styles, see [B.1](#) for a full list of styles.

Chapter 2

Comprehensive Tutorials

2.1 Introduction

In this chapter we'll build a few complete applications and show off some of the more advanced features of PyFLTK.

2.2 Building A Calculator

2.3 OpenGL

Chapter 3

Fluid & FlConvert

Appendix A

API Reference

A.1 Classes

A.1.1 Fl_Adjuster

A.1.2 Fl_Bitmap

A.1.3 Fl_BMP_Image

A.1.4 Fl_Box

A.1.5 Fl_Browser

A.1.6 Fl_Browser_

A.1.7 Fl_Button

A.1.8 Fl_Chart

A.1.9 Fl_Check_Browser

A.1.10 Fl_Check_Button

A.1.11 Fl_Choice

A.1.12 Fl_Clock

A.1.13 Fl_Color_Chooser

A.1.14 Fl_Counter

A.1.15 Fl_Dial

A.1.16 Fl_Double_Window

A.1.17 Fl_End

A.1.18 Fl_File_Browser

A.1.19 Fl_File_Chooser

A.1.20 Fl_File_Icon

A.1.21 Fl_File_Input

A.1.22 Fl_Free

A.1.23 Fl_GIF_Images

A.1.24 Fl_GL_Window

A.1.25 Fl_Group

A.1.26 Fl_Help_Dialog

Appendix B

Styles

B.1 Boxes

B.2 Frames