**PYGTK**

1. Installing gtk on Ubuntu. On Ubuntu 14.04 LTS version, gtk is available by default. If not, install it by typing with root account

# apt-get install python-gtk2

1. Following are the examples for different widget creations in gtk.

**import gtk**

**window = gtk.Window()**

**window.set\_title("PyGTK Test Window")**

**window.connect("destroy", gtk.main\_quit)**

**window.show\_all()**

**gtk.main()**

**This code shows a centered window.**

import gtk

class PyApp(gtk.Window):

def \_\_init\_\_(self):

super(PyApp, self).\_\_init\_\_()

self.connect("destroy", gtk.main\_quit)

self.set\_size\_request(250, 150)

self.set\_position(gtk.WIN\_POS\_CENTER)

self.show()

PyApp()

gtk.main()

import gtk

We import the gtk module. Here we have objects to create GUI applications.

class PyApp(gtk.Window):

Our application is based on the PyApp class. It inherits from the Window.

def \_\_init\_\_(self):

super(PyApp, self).\_\_init\_\_()

This is the constructor. It builds our application. It also calls its parent constructor through the super() call.

self.connect("destroy", gtk.main\_quit)

We connect the destroy signal to the main\_quit() function. The destroy signal is called when we click on the close button in the titlebar or press Alt+F4. The window is being destroyed, but the application is not. You can see it if you launch the example from the command line. By calling the main\_quit() we quit the application for good.

self.set\_size\_request(250, 150)

We set the size of the window to 250x150px.

self.set\_position(gtk.WIN\_POS\_CENTER)

This line centers the window on the screen.

self.show()

Now we show the window. The window is not visible, until we call the show() method.

PyApp()

gtk.main()

We create the instance of our program and start the main loop.

The code example shows the application icon.

import gtk, sys

class PyApp(gtk.Window):

def \_\_init\_\_(self):

super(PyApp, self).\_\_init\_\_()

self.set\_title("Icon")

self.set\_size\_request(250, 150)

self.set\_position(gtk.WIN\_POS\_CENTER)

try:

self.set\_icon\_from\_file("web.png")

except Exception, e:

print e.message

sys.exit(1)

self.connect("destroy", gtk.main\_quit)

self.show()

PyApp()

gtk.main()

**Buttons**

In the next example, we will further enhance our programming skills with the PyGTK library.

buttons.py

#!/usr/bin/python

# ZetCode PyGTK tutorial

#

# This example shows four buttons

# in various modes

#

# author: jan bodnar

# website: zetcode.com

# last edited: February 2009

import gtk

class PyApp(gtk.Window):

def \_\_init\_\_(self):

super(PyApp, self).\_\_init\_\_()

self.set\_title("Buttons")

self.set\_size\_request(250, 200)

self.set\_position(gtk.WIN\_POS\_CENTER)

btn1 = gtk.Button("Button")

btn1.set\_sensitive(False)

btn2 = gtk.Button("Button")

btn3 = gtk.Button(stock=gtk.STOCK\_CLOSE)

btn4 = gtk.Button("Button")

btn4.set\_size\_request(80, 40)

fixed = gtk.Fixed()

fixed.put(btn1, 20, 30)

fixed.put(btn2, 100, 30)

fixed.put(btn3, 20, 80)

fixed.put(btn4, 100, 80)

self.connect("destroy", gtk.main\_quit)

self.add(fixed)

self.show\_all()

PyApp()

gtk.main()

We show four different buttons on the window. We will see a difference between container widgets and child widgets and will change some properties of child widgets.

btn1 = gtk.Button("Button")

A Button is a child widget. Child widgets are placed inside containers.

btn1.set\_sensitive(False)

We make this button insensitive. This means, we cannot click on it. Nor it can be selected, focused etc. Graphically the widget is grayed out.

btn3 = gtk.Button(stock=gtk.STOCK\_CLOSE)

The third button shows an image inside its area. The PyGTK library has a built-in stock of images that we can use.

btn4.set\_size\_request(80, 40)

Here we change the size of the button.

fixed = gtk.Fixed()

Fixed widget is a non visible container widget. Its purpose is to contain other child widgets.

fixed.put(btn1, 20, 30)

fixed.put(btn2, 100, 30)

...

Here we place button widgets inside fixed container widget.

self.add(fixed)

We set the Fixed container to be the main container for our Window widget.

self.show\_all()

We can either call show\_all() method, or we call show() method on each of the widgets. Including containers.

Figure: Buttons

**Tooltip**

A tooltip is a hint on a widget in the applications. Can be used to provide additional help.

tooltips.py

#!/usr/bin/python

# ZetCode PyGTK tutorial

#

# This code shows a tooltip on

# a window and a button

#

# author: jan bodnar

# website: zetcode.com

# last edited: February 2009

import gtk

class PyApp(gtk.Window):

def \_\_init\_\_(self):

super(PyApp, self).\_\_init\_\_()

self.set\_title("Tooltips")

self.set\_size\_request(250, 200)

self.set\_position(gtk.WIN\_POS\_CENTER)

self.connect("destroy", gtk.main\_quit)

self.fixed = gtk.Fixed()

self.add(self.fixed)

button = gtk.Button("Button")

button.set\_size\_request(80, 35)

self.fixed.put(button, 50, 50)

self.set\_tooltip\_text("Window widget")

button.set\_tooltip\_text("Button widget")

self.show\_all()

PyApp()

gtk.main()

In this example we set a tooltip for a window and for a button.

self.set\_tooltip\_text("Window widget")

button.set\_tooltip\_text("Button widget")

The set\_tooltip\_text() does the job.

Figure: Tooltips

In this chapter, we created first programs in PyGTK programming library.

inews = gtk.MenuItem("Import news feed...")

ibookmarks = gtk.MenuItem("Import bookmarks...")

imail = gtk.MenuItem("Import mail...")

imenu.append(inews)

imenu.append(ibookmarks)

imenu.append(imail)

Submenu has its own menu items.

Figure: Submenu

In this chapter of the PyGTK programming library, we showed, how to work with menus.

**Menus in PyGTK**

In this part of the PyGTK programming tutorial, we will work with menus.

A *menubar* is one of the most common parts of the GUI application. It is a group of commands located in various menus. While in console applications you have to remember all those arcane commands, here we have most of the commands grouped into logical parts. These are accepted standards that further reduce the amount of time spent to learn a new application.

**Simple menu**

In our first example, we will create a menubar with one file menu. The menu will have only one menu item. By selecting the item the application quits.

simplemenu.py

#!/usr/bin/python

# ZetCode PyGTK tutorial

#

# This example shows a simple menu

#

# author: jan bodnar

# website: zetcode.com

# last edited: February 2009

import gtk

class PyApp(gtk.Window):

def \_\_init\_\_(self):

super(PyApp, self).\_\_init\_\_()

self.set\_title("Simple menu")

self.set\_size\_request(250, 200)

self.modify\_bg(gtk.STATE\_NORMAL, gtk.gdk.Color(6400, 6400, 6440))

self.set\_position(gtk.WIN\_POS\_CENTER)

mb = gtk.MenuBar()

filemenu = gtk.Menu()

filem = gtk.MenuItem("File")

filem.set\_submenu(filemenu)

exit = gtk.MenuItem("Exit")

exit.connect("activate", gtk.main\_quit)

filemenu.append(exit)

mb.append(filem)

vbox = gtk.VBox(False, 2)

vbox.pack\_start(mb, False, False, 0)

self.add(vbox)

self.connect("destroy", gtk.main\_quit)

self.show\_all()

PyApp()

gtk.main()

This is a small example with minimal menubar functionality.

mb = gtk.MenuBar()

MenuBar widget is created.

filemenu = gtk.Menu()

filem = gtk.MenuItem("File")

filem.set\_submenu(filemenu)

Toplevel MenuItem is created.

exit = gtk.MenuItem("Exit")

exit.connect("activate", gtk.main\_quit)

filemenu.append(exit)

Exit MenuItem is created and appended to the File MenuItem.

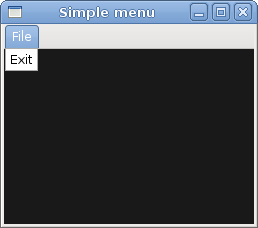
mb.append(filem)

Toplevel MenuItem is appended to the MenuBar widget.

vbox = gtk.VBox(False, 2)

vbox.pack\_start(mb, False, False, 0)

Unlike in other toolkits, we have to take care of the layout management of the menubar ourselves. We put the menubar into a vertical box.

Figure: Simple menu

**Image menu**

In the next example, we will further explore the menus. We will add images and accelerators to our menu items. *Accelerators* are keyboard shortcuts for activating a menu item.

imagemenu.py

#!/usr/bin/python

# ZetCode PyGTK tutorial

#

# This example shows a menu with

# images, accelerators and a separator

#

# author: jan bodnar

# website: zetcode.com

# last edited: February 2009

import gtk

class PyApp(gtk.Window):

def \_\_init\_\_(self):

super(PyApp, self).\_\_init\_\_()

self.set\_title("Image menu")

self.set\_size\_request(250, 200)

self.modify\_bg(gtk.STATE\_NORMAL, gtk.gdk.Color(6400, 6400, 6440))

self.set\_position(gtk.WIN\_POS\_CENTER)

mb = gtk.MenuBar()

filemenu = gtk.Menu()

filem = gtk.MenuItem("\_File")

filem.set\_submenu(filemenu)

agr = gtk.AccelGroup()

self.add\_accel\_group(agr)

newi = gtk.ImageMenuItem(gtk.STOCK\_NEW, agr)

key, mod = gtk.accelerator\_parse("<Control>N")

newi.add\_accelerator("activate", agr, key,

mod, gtk.ACCEL\_VISIBLE)

filemenu.append(newi)

openm = gtk.ImageMenuItem(gtk.STOCK\_OPEN, agr)

key, mod = gtk.accelerator\_parse("<Control>O")

openm.add\_accelerator("activate", agr, key,

mod, gtk.ACCEL\_VISIBLE)

filemenu.append(openm)

sep = gtk.SeparatorMenuItem()

filemenu.append(sep)

exit = gtk.ImageMenuItem(gtk.STOCK\_QUIT, agr)

key, mod = gtk.accelerator\_parse("<Control>Q")

exit.add\_accelerator("activate", agr, key,

mod, gtk.ACCEL\_VISIBLE)

exit.connect("activate", gtk.main\_quit)

filemenu.append(exit)

mb.append(filem)

vbox = gtk.VBox(False, 2)

vbox.pack\_start(mb, False, False, 0)

self.add(vbox)

self.connect("destroy", gtk.main\_quit)

self.show\_all()

PyApp()

gtk.main()

Our example shows a toplevel menu item with three sublevel menu items. Each of the menu items has a image and an accelerator. The accelerator for the quit menu item is active.

agr = gtk.AccelGroup()

self.add\_accel\_group(agr)

To work with accelerators, we create a global AccelGroup object. It will be used later.

newi = gtk.ImageMenuItem(gtk.STOCK\_NEW, agr)

ImageMenuItem is created. The image comes from the stock of images.

key, mod = gtk.accelerator\_parse("<Control>N")

The gtk.accelerator\_parse() function parses the specified accelerator string and returns a 2-tuple containing the keyval and modifier mask corresponding to accelerator.

newi.add\_accelerator("activate", agr, key,

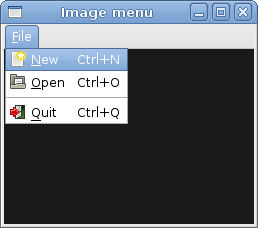
mod, gtk.ACCEL\_VISIBLE)

This creates an Ctrl+Q accelerator for the exit menu item.

sep = gtk.SeparatorMenuItem()

filemenu.append(sep)

These lines create a separator. It is used to group menu items into logical groups.

Figure: Image menu

**CheckMenuItem**

A CheckMenuItem is a menu item with a check box. It can be used to work with boolean properties.

checkmenuitem.py

#!/usr/bin/python

# ZetCode PyGTK tutorial

#

# This example shows how to

# use a CheckMenuItem

#

# author: jan bodnar

# website: zetcode.com

# last edited: February 2009

import gtk

class PyApp(gtk.Window):

def \_\_init\_\_(self):

super(PyApp, self).\_\_init\_\_()

self.set\_title("Check menu item")

self.set\_size\_request(250, 200)

self.modify\_bg(gtk.STATE\_NORMAL, gtk.gdk.Color(6400, 6400, 6440))

self.set\_position(gtk.WIN\_POS\_CENTER)

mb = gtk.MenuBar()

filemenu = gtk.Menu()

filem = gtk.MenuItem("File")

filem.set\_submenu(filemenu)

viewmenu = gtk.Menu()

view = gtk.MenuItem("View")

view.set\_submenu(viewmenu)

stat = gtk.CheckMenuItem("View Statusbar")

stat.set\_active(True)

stat.connect("activate", self.on\_status\_view)

viewmenu.append(stat)

exit = gtk.MenuItem("Exit")

exit.connect("activate", gtk.main\_quit)

filemenu.append(exit)

mb.append(filem)

mb.append(view)

self.statusbar = gtk.Statusbar()

self.statusbar.push(1, "Ready")

vbox = gtk.VBox(False, 2)

vbox.pack\_start(mb, False, False, 0)

vbox.pack\_start(gtk.Label(), True, False, 0)

vbox.pack\_start(self.statusbar, False, False, 0)

self.add(vbox)

self.connect("destroy", gtk.main\_quit)

self.show\_all()

def on\_status\_view(self, widget):

if widget.active:

self.statusbar.show()

else:

self.statusbar.hide()

PyApp()

gtk.main()

In our code example we show a check menu item. If the check box is activated, the statusbar widget is shown. If not, the statusbar is hidden.

stat = gtk.CheckMenuItem("View Statusbar")

CheckMenuItem widget is created.

stat.set\_active(True)

The set\_active() method checks/unchecks the check menu item.

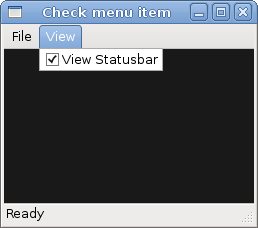
if widget.active:

self.statusbar.show()

else:

self.statusbar.hide()

Depending on the active property of the CheckMenuItem, we show or hide the statusbar widget.

Figure: CheckMenuItem

**Submenu**

Our final example demonstrates how to create a submenu in PyGTK.

submenu.py

#!/usr/bin/python

# ZetCode PyGTK tutorial

#

# This example shows a submenu

#

# author: jan bodnar

# website: zetcode.com

# last edited: February 2009

import gtk

class PyApp(gtk.Window):

def \_\_init\_\_(self):

super(PyApp, self).\_\_init\_\_()

self.set\_title("Submenu")

self.set\_size\_request(250, 200)

self.modify\_bg(gtk.STATE\_NORMAL, gtk.gdk.Color(6400, 6400, 6440))

self.set\_position(gtk.WIN\_POS\_CENTER)

mb = gtk.MenuBar()

filemenu = gtk.Menu()

filem = gtk.MenuItem("File")

filem.set\_submenu(filemenu)

mb.append(filem)

imenu = gtk.Menu()

importm = gtk.MenuItem("Import")

importm.set\_submenu(imenu)

inews = gtk.MenuItem("Import news feed...")

ibookmarks = gtk.MenuItem("Import bookmarks...")

imail = gtk.MenuItem("Import mail...")

imenu.append(inews)

imenu.append(ibookmarks)

imenu.append(imail)

filemenu.append(importm)

exit = gtk.MenuItem("Exit")

exit.connect("activate", gtk.main\_quit)

filemenu.append(exit)

vbox = gtk.VBox(False, 2)

vbox.pack\_start(mb, False, False, 0)

self.add(vbox)

self.connect("destroy", gtk.main\_quit)

self.show\_all()

PyApp()

gtk.main()

Submenu creation.

imenu = gtk.Menu()

A submenu is a Menu.

importm = gtk.MenuItem("Import")

importm.set\_submenu(imenu)

It is a submenu of a menu item, which belogs to toplevel file menu.

inews = gtk.MenuItem("Import news feed...")

ibookmarks = gtk.MenuItem("Import bookmarks...")

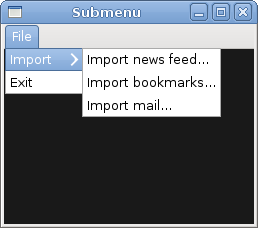
imail = gtk.MenuItem("Import mail...")

imenu.append(inews)

imenu.append(ibookmarks)

imenu.append(imail)

Submenu has its own menu items.

Figure: Submenu

# Toolbars in PyGTK

In this part of the PyGTK programming tutorial, we will work with toolbars.

Menus group commands that we can use in application. Toolbars provide a quick access to the most frequently used commands.

## Simple toolbar

Next we create a simple toolbar.

toolbar.py

#!/usr/bin/python

# ZetCode PyGTK tutorial

#

# This example shows a toolbar

# widget

#

# author: jan bodnar

# website: zetcode.com

# last edited: February 2009

import gtk

class PyApp(gtk.Window):

def \_\_init\_\_(self):

super(PyApp, self).\_\_init\_\_()

self.set\_title("Toolbar")

self.set\_size\_request(250, 200)

self.modify\_bg(gtk.STATE\_NORMAL, gtk.gdk.Color(6400, 6400, 6440))

self.set\_position(gtk.WIN\_POS\_CENTER)

toolbar = gtk.Toolbar()

toolbar.set\_style(gtk.TOOLBAR\_ICONS)

newtb = gtk.ToolButton(gtk.STOCK\_NEW)

opentb = gtk.ToolButton(gtk.STOCK\_OPEN)

savetb = gtk.ToolButton(gtk.STOCK\_SAVE)

sep = gtk.SeparatorToolItem()

quittb = gtk.ToolButton(gtk.STOCK\_QUIT)

toolbar.insert(newtb, 0)

toolbar.insert(opentb, 1)

toolbar.insert(savetb, 2)

toolbar.insert(sep, 3)

toolbar.insert(quittb, 4)

quittb.connect("clicked", gtk.main\_quit)

vbox = gtk.VBox(False, 2)

vbox.pack\_start(toolbar, False, False, 0)

self.add(vbox)

self.connect("destroy", gtk.main\_quit)

self.show\_all()

PyApp()

gtk.main()

The example shows a toolbar and four tool buttons.

toolbar = gtk.Toolbar()

A Toolbar widget is created.

toolbar.set\_style(gtk.TOOLBAR\_ICONS)

On toolbar, we show only icons. No text.

newtb = gtk.ToolButton(gtk.STOCK\_NEW)

A ToolButton with an image from stock is created.

sep = gtk.SeparatorToolItem()

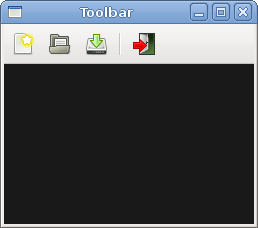
This is a separator. It can be used to group toolbar buttons into logical groups.

toolbar.insert(newtb, 0)

toolbar.insert(opentb, 1)

...

Toolbar buttons are inserted into the toolbar widget.

Figure: Toolbar

## Toolbars

In the second example, we show two toolbars. Many applications have more than one toolbar. We show, how we can do it in PyGTK.

toolbars.py

#!/usr/bin/python

# ZetCode PyGTK tutorial

#

# This example shows two toolbars

# in the application window

#

# author: jan bodnar

# website: zetcode.com

# last edited: February 2009

import gtk

class PyApp(gtk.Window):

def \_\_init\_\_(self):

super(PyApp, self).\_\_init\_\_()

self.set\_title("Toolbars")

self.set\_size\_request(350, 300)

self.modify\_bg(gtk.STATE\_NORMAL, gtk.gdk.Color(6400, 6400, 6440))

self.set\_position(gtk.WIN\_POS\_CENTER)

upper = gtk.Toolbar()

upper.set\_style(gtk.TOOLBAR\_ICONS)

newtb = gtk.ToolButton(gtk.STOCK\_NEW)

opentb = gtk.ToolButton(gtk.STOCK\_OPEN)

savetb = gtk.ToolButton(gtk.STOCK\_SAVE)

upper.insert(newtb, 0)

upper.insert(opentb, 1)

upper.insert(savetb, 2)

lower = gtk.Toolbar()

lower.set\_style(gtk.TOOLBAR\_ICONS)

quittb = gtk.ToolButton(gtk.STOCK\_QUIT)

quittb.connect("clicked", gtk.main\_quit)

lower.insert(quittb, 0)

vbox = gtk.VBox(False, 0)

vbox.pack\_start(upper, False, False, 0)

vbox.pack\_start(lower, False, False, 0)

self.add(vbox)

self.connect("destroy", gtk.main\_quit)

self.show\_all()

PyApp()

gtk.main()

Our applications shows two toolbars.

upper = gtk.Toolbar()

...

lower = gtk.Toolbar()

We create two Toolbar widgets.

upper.insert(newtb, 0)

...

lower.insert(quittb, 0)

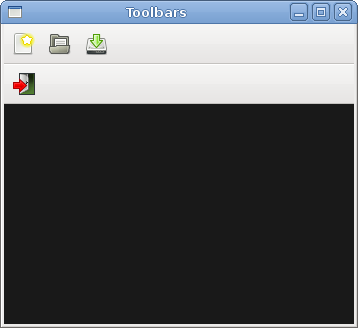
Each of them has its own tool buttons.

vbox = gtk.VBox(False, 0)

vbox.pack\_start(upper, False, False, 0)

vbox.pack\_start(lower, False, False, 0)

Toolbars are packed into the vertical box, one after the other.

Figure: Toolbars

# Widgets in PyGTK

In this part of the PyGTK programming tutorial, we will introduce some PyGTK widgets.

Widgets are basic building blocks of a GUI application. Over the years, several widgets became a standard in all toolkits on all OS platforms. For example a button, a check box or a scroll bar. The PyGTK toolkit's philosophy is to keep the number of widgets at a minimum level. More specialised widgets are created as custom PyGTK widgets.

## Label

The Label widget displays a limited amount of read-only text.

label.py

#!/usr/bin/python

# ZetCode PyGTK tutorial

#

# This example demonstrates the Label widget

#

# author: jan bodnar

# website: zetcode.com

# last edited: February 2009

import gtk

lyrics = """Meet you downstairs in the bar and heard

your rolled up sleeves and your skull t-shirt

You say why did you do it with him today?

and sniff me out like I was Tanqueray

cause you're my fella, my guy

hand me your stella and fly

by the time I'm out the door

you tear men down like Roger Moore

I cheated myself

like I knew I would

I told ya, I was trouble

you know that I'm no good"""

class PyApp(gtk.Window):

def \_\_init\_\_(self):

super(PyApp, self).\_\_init\_\_()

self.set\_position(gtk.WIN\_POS\_CENTER)

self.set\_border\_width(8)

self.connect("destroy", gtk.main\_quit)

self.set\_title("You know I'm no Good")

label = gtk.Label(lyrics)

self.add(label)

self.show\_all()

PyApp()

gtk.main()

The code example shows some lyrics on the window.

lyrics = """Meet you downstairs in the bar and heard

your rolled up sleeves and your skull t-shirt

..."""

This is the text that we display.

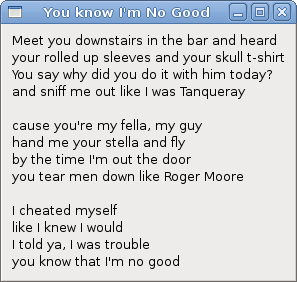
self.set\_border\_width(8)

The Label is surrounded by some empty space.

label = gtk.Label(lyrics)

self.add(label)

The Label widget is created and added to the window.

Figure: Label Widget

## CheckButton

CheckButton is a widget that has two states: on and off. The n state is visualised by a check mark. It is used to denote some boolean property.

checkbutton.py

#!/usr/bin/python

# ZetCode PyGTK tutorial

#

# This example demonstrates the CheckButton widget

#

# author: jan bodnar

# website: zetcode.com

# last edited: February 2009

import gtk

class PyApp(gtk.Window):

def \_\_init\_\_(self):

super(PyApp, self).\_\_init\_\_()

self.set\_title("Check Button")

self.set\_position(gtk.WIN\_POS\_CENTER)

self.set\_default\_size(250, 200)

fixed = gtk.Fixed()

button = gtk.CheckButton("Show title")

button.set\_active(True)

button.unset\_flags(gtk.CAN\_FOCUS)

button.connect("clicked", self.on\_clicked)

fixed.put(button, 50, 50)

self.connect("destroy", gtk.main\_quit)

self.add(fixed)

self.show\_all()

def on\_clicked(self, widget):

if widget.get\_active():

self.set\_title("Check Button")

else:

self.set\_title("")

PyApp()

gtk.main()

We will display a title in the titlebar of the window, depending on the state of the CheckButton.

button = gtk.CheckButton("Show title")

CheckButton widget is created.

button.set\_active(True)

The title is visible by default, so we check the check button by default.

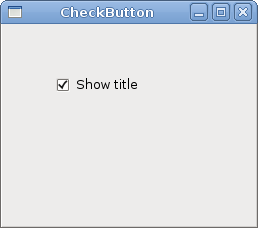
if widget.get\_active():

self.set\_title("Check Button")

else:

self.set\_title("")

If the CheckButton is checked we show the title. Otherwise we put empty text in the titlebar.

Figure: CheckButton

## ComboBox

ComboBox is a widget that allows the user to choose from a list of options.

combobox.py

#!/usr/bin/python

# ZetCode PyGTK tutorial

#

# This example demonstrates the ComboBox widget

#

# author: jan bodnar

# website: zetcode.com

# last edited: February 2009

import gtk

class PyApp(gtk.Window):

def \_\_init\_\_(self):

super(PyApp, self).\_\_init\_\_()

self.set\_title("ComboBox")

self.set\_default\_size(250, 200)

self.set\_position(gtk.WIN\_POS\_CENTER)

cb = gtk.combo\_box\_new\_text()

cb.connect("changed", self.on\_changed)

cb.append\_text('Ubuntu')

cb.append\_text('Mandriva')

cb.append\_text('Redhat')

cb.append\_text('Gentoo')

cb.append\_text('Mint')

fixed = gtk.Fixed()

fixed.put(cb, 50, 30)

self.label = gtk.Label("-")

fixed.put(self.label, 50, 140)

self.add(fixed)

self.connect("destroy", gtk.main\_quit)

self.show\_all()

def on\_changed(self, widget):

self.label.set\_label(widget.get\_active\_text())

PyApp()

gtk.main()

The example shows a combo box and a label. The combo box has a list of six options. These are the names of Linux distros. The label widget shows the selected option from the combo box.

cb = gtk.combo\_box\_new\_text()

The gtk.combo\_box\_new\_text() function is a convenience function that constructs a new text combo box. It is a ComboBox just displaying strings.

cb.append\_text('Ubuntu')

cb.append\_text('Mandriva')

cb.append\_text('Redhat')

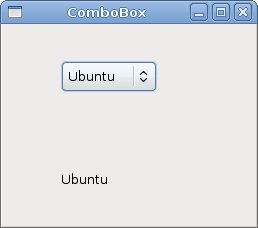
cb.append\_text('Gentoo')

cb.append\_text('Mint')

The ComboBox is filled with textual data.

self.label.set\_label(widget.get\_active\_text())

Inside the on\_changed() method, we get the selected text out of the combo box and set it to the label.

Figure: ComboBox

## Image

The next example introduces the Image widget. This widget displays pictures.

image.py

#!/usr/bin/python

# ZetCode PyGTK tutorial

#

# This example demonstrates the Image widget

#

# author: jan bodnar

# website: zetcode.com

# last edited: February 2009

import gtk

class PyApp(gtk.Window):

def \_\_init\_\_(self):

super(PyApp, self).\_\_init\_\_()

self.set\_title("Red Rock")

self.set\_position(gtk.WIN\_POS\_CENTER)

self.set\_border\_width(2)

image = gtk.Image()

image.set\_from\_file("redrock.png")

self.connect("destroy", gtk.main\_quit)

self.add(image)

self.show\_all()

PyApp()

gtk.main()

We show the Red Rock castle in the window.

image = gtk.Image()

Image widget is created.

image.set\_from\_file("redrock.png")

We set a PNG image to the Image widget. The picture is loaded from the file on the disk.

Figure: Image

In this chapter, we showed the first pack of basic widgets of the PyGTK programming library.

**Entry**

The Entry is a single line text entry field. This widget is used to enter textual data.

entry.py

#!/usr/bin/python

# ZetCode PyGTK tutorial

#

# This example demonstrates the Entry widget

#

# author: jan bodnar

# website: zetcode.com

# last edited: February 2009

import gtk

class PyApp(gtk.Window):

def \_\_init\_\_(self):

super(PyApp, self).\_\_init\_\_()

self.set\_title("Entry")

self.set\_size\_request(250, 200)

self.set\_position(gtk.WIN\_POS\_CENTER)

fixed = gtk.Fixed()

self.label = gtk.Label("...")

fixed.put(self.label, 60, 40)

entry = gtk.Entry()

entry.add\_events(gtk.gdk.KEY\_RELEASE\_MASK)

fixed.put(entry, 60, 100)

entry.connect("key-release-event", self.on\_key\_release)

self.connect("destroy", gtk.main\_quit)

self.add(fixed)

self.show\_all()

def on\_key\_release(self, widget, event):

self.label.set\_text(widget.get\_text())

PyApp()

gtk.main()

This example shows an entry widget and a label. The text that we key in the entry is displayed immediately in the label control.

entry = gtk.Entry()

Entry widget is created.

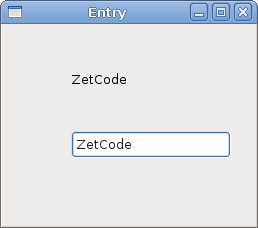
entry.connect("key-release-event", self.on\_key\_release)

If the text in the Entry widget is changed, we call the on\_key\_release() method.

def on\_key\_release(self, widget, event):

self.label.set\_text(widget.get\_text())

We get the text from the Entry widget and set it to the label.

Figure: Entry Widget

**ToggleButton**

ToggleButton is a button that has two states. Pressed and not pressed. You toggle between these two states by clicking on it. There are situations where this functionality fits well.

togglebuttons.py

#!/usr/bin/python

# ZetCode PyGTK tutorial

#

# This example demonstrates the ToggleButton widget

#

# author: jan bodnar

# website: zetcode.com

# last edited: February 2009

import gtk

class PyApp(gtk.Window):

def \_\_init\_\_(self):

super(PyApp, self).\_\_init\_\_()

self.color = [0, 0, 0]

self.set\_title("ToggleButtons")

self.resize(350, 240)

self.set\_position(gtk.WIN\_POS\_CENTER)

self.connect("destroy", gtk.main\_quit)

red = gtk.ToggleButton("Red")

red.set\_size\_request(80, 35)

red.connect("clicked", self.onred)

green = gtk.ToggleButton("Green")

green.set\_size\_request(80, 35)

green.connect("clicked", self.ongreen)

blue = gtk.ToggleButton("Blue")

blue.set\_size\_request(80, 35)

blue.connect("clicked", self.onblue)

self.darea = gtk.DrawingArea()

self.darea.set\_size\_request(150, 150)

self.darea.modify\_bg(gtk.STATE\_NORMAL, gtk.gdk.color\_parse("black"))

fixed = gtk.Fixed()

fixed.put(red, 30, 30)

fixed.put(green, 30, 80)

fixed.put(blue, 30, 130)

fixed.put(self.darea, 150, 30)

self.add(fixed)

self.show\_all()

def onred(self, widget):

if widget.get\_active():

self.color[0] = 65535

else: self.color[0] = 0

self.darea.modify\_bg(gtk.STATE\_NORMAL, gtk.gdk.Color(self.color[0],

self.color[1], self.color[2]))

def ongreen(self, widget):

if (widget.get\_active()):

self.color[1] = 65535

else: self.color[1] = 0

self.darea.modify\_bg(gtk.STATE\_NORMAL, gtk.gdk.Color(self.color[0],

self.color[1], self.color[2]))

def onblue(self, widget):

if (widget.get\_active()):

self.color[2] = 65535

else: self.color[2] = 0

self.darea.modify\_bg(gtk.STATE\_NORMAL, gtk.gdk.Color(self.color[0],

self.color[1], self.color[2]))

PyApp()

gtk.main()

In our example, we show three toggle buttons and a DrawingArea. We set the background colour of the area to black. The togglebuttons will toggle the red, green and blue parts of the colour value. The background colour will depend on which togglebuttons we have pressed.

self.color = [0, 0, 0]

This is the colour value that is going to be updated with the toggle buttons.

red = gtk.ToggleButton("Red")

red.set\_size\_request(80, 35)

red.connect("clicked", self.onred)

The ToggleButton widget is created. We set its size to 80x35 pixels. Each of the toggle buttons has its own handler method.

self.darea = gtk.DrawingArea()

self.darea.set\_size\_request(150, 150)

self.darea.modify\_bg(gtk.STATE\_NORMAL, gtk.gdk.color\_parse("black"))

The DrawingArea widget is the widget that displays the colour, mixed by the toggle buttons. At start, it shows black colour.

if widget.get\_active():

self.color[0] = 65535

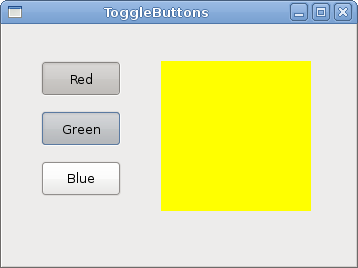
else: self.color[0] = 0

If the toggle button is pressed, we change the R, G or B part of the colour accordingly.

self.darea.modify\_bg(gtk.STATE\_NORMAL, gtk.gdk.Color(self.color[0],

self.color[1], self.color[2]))

We update the colour of the DrawingArea widget.

Figure: ToggleButton widget

**Calendar**

Our final widget is the Calendar widget. It is used to work with dates.

calendar.py

#!/usr/bin/python

# ZetCode PyGTK tutorial

#

# This example demonstrates the Calendar widget

#

# author: jan bodnar

# website: zetcode.com

# last edited: February 2009

import gtk

class PyApp(gtk.Window):

def \_\_init\_\_(self):

super(PyApp, self).\_\_init\_\_()

self.set\_title("Calendar")

self.set\_size\_request(300, 270)

self.set\_position(gtk.WIN\_POS\_CENTER)

self.set\_border\_width(2)

self.label = gtk.Label("...")

calendar = gtk.Calendar()

calendar.connect("day\_selected", self.on\_day\_selected)

fix = gtk.Fixed()

fix.put(calendar, 20, 20)

fix.put(self.label, 40, 230)

self.add(fix)

self.connect("destroy", gtk.main\_quit)

self.show\_all()

def on\_day\_selected(self, widget):

(year, month, day) = widget.get\_date()

self.label.set\_label(str(month) + "/" + str(day) + "/" + str(year))

PyApp()

gtk.main()

We have the Calendar widget and a Label. The selected day from the calendar is shown in the label.

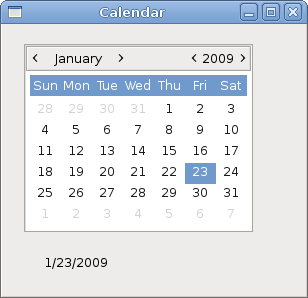
calendar = gtk.Calendar()

Calendar widget is created.

(year, month, day) = widget.get\_date()

self.label.set\_label(str(month) + "/" + str(day) + "/" + str(year))

In the on\_day\_selected() method we retrieve the currently selected date, and update the label.

Figure: Calendar

In this chapter of the PyGTK tutorial, we finished talking about the PyGTK widgets.