PyGTK 教程

英文版来自: http://www.zetcode.com/tutorials/pygtktutorial/

翻译：iceleaf <iceleaf916@gmail.com>

这是 PyGTK 教程，本教程适合于初学者和有经验的程序员。

00 PyGTK 简介

在这个部分，我们将谈谈 PyGTK GUI（图形用户界面）库和一般的 Python 语言编程。

About

本教程是 PyGTK 编程教程。其中的例子在 Linux 上被创建并测试通过。PyGTK 编程教程适合于新手以及高级用户。

PyGTK

PyGTK 是一套 GTK+ GUI 库的 Python 封装。它为创建桌面程序提供了一套综合的图形元素和其它实用的编程工具。它是 GNOME 项目的一部分。PyGTK 是基于 LGPL 许可之下的免费软件。其原始作者是 **James Henstridge**。PyGTK 非常容易使用，对于速成原型法，它是相当理想的。普遍地认为，PyGTK 是最流行的 GTK+库封装中的一种。

PyGTK 包含以下几个模块：

GObject 是基类，它为 PyGTK 所以类提供通用的属性和函数。**ATK** 是一个提供辅助功能的工具包。该工具包提供了帮助残障人士使用计算机的各种工具。**GTK** 是用户界面模块。**Pango** 是一个用于处理文本和国际化的库。**Cairo** 是一个用于创建 2D 矢量模型的库。**Glade** 是用来从 XML 描述中构建 GUI 界面。

Python

Python 是一个动态的面向对象的编程语言。它是一种通用编程语言。它能被用于许多种类的软件开发。Python 语言的设计目的是强调程序员的生产率和代码的可读性。它最初是由 **Guido van Rossum** 开发的，并且于 1991 年第一次被发布。创造 Python 语言的灵感来源于 ABC, Haskell, Java, Lisp, Icon 和 Perl 这些编程语言。Python 是一种高级的、通用的、跨平台的解释型语言。Python 是一种极为简洁的语言。它的一种最明显的特征之一是，它不使用逗号和括号，而是使用缩进来代替。Python 当前有两个主要的分支——Python 2.x 和 Python 3.x。Python 3.x 与之前的 Python 发行版相比，停止了向后兼容。它被创建用来纠正语言设计上的缺陷，使该语言更加简洁。Python 2.x 的最新版本是 2.7.1，Python 3.x 的是 3.1.3。本教程是为当前 Python 2.x 的版本所写。现在 Python 是由来自世界各地的一大群志愿者维护。

GTK+

GTK+是一个用于创建图形用户界面的库。该库是用 C 语言创建。GTK+库也被称为 GIMP 工具包。最初，该库被创建是为了开发 GIMP 图像处理程序。自此，GTK+成为了 Linux 和 BSD Unix 下最流行的工具包之一。现在，在开源世界中大多数的 GUI 软件是用 QT 或者 GTK+创建。GTK+是一个面向对象的应用程序接口。面向对象系统是基于 Glib 对象系统而创建，Glib 库是 GTK+库的基础。GObject 也能够使程序员创建各种各样其它编程语言的绑定。GTK+语言的绑定包括 C++, Python, Perl, Java, C#以及其它程序设计语言。

Gnome 和 XFce 桌面环境已经以 GTK+库为基础被创建。SWT 和 wxWidgets 是著名的编程框架，它们也是用 GTK+创建的。使用 GTK+的杰出的软件程序包括 Firefox 或者 Inkscape 等。

01 PyGTK 的第一步

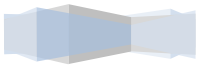
在本教程的这部分里，我们将进行我们编程的第一步。我们将创建示例程序。

Simple example

第一个代码示例是一个非常简单的

Code: center.py

```
#!/usr/bin/python
# ZetCode PyGTK tutorial
#
# This is a trivial PyGTK example
#
# author: jan bodnar
# website: zetcode.com
# last edited: February 2009
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()
```

这段代码展示了一个位于屏幕中心的窗口。

```
    import gtk
```

我们导入 gtk 模块。在这里，我们用对象来创建 GUI 应用程序。

```
    class PyApp(gtk.Window):
```

我们的程序基于 **PyApp** 类，它继承自 **Window**。

```
    def __init__(self):
        super(PyApp, self).__init__()
```

这是构造函数，它初始化我们的程序。它也通过 **super()**函数回调它的父构造函数。

```
    self.connect("destroy", gtk.main_quit)
```

我们连接 **destroy** 信号到 **main_quit()**函数。当我们点击窗口标题栏的关闭按钮或者按下 ALt+F4，**destroy** 信号将会被调用。窗口将会被销毁，但是程序没有被停止。如果你从命令行启动这个例子，你会看到这种情况。我们通过调用 **main_quit()**函数退出程序，这是很好的做法。

```
    self.set_size_request(250, 150)
```

我们设置窗口的尺寸为 250×150px.

```
    self.set_position(gtk.WIN_POS_CENTER)
```

这一行使窗口位居屏幕的中心。

```
    self.show()
```

现在我们显示这个窗口。这个窗口直到我们调用 show()方法，才会是可见的。

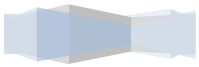
```
    PyApp()
    gtk.main()
```

我们创建了我们的程序的实例，并且开始了主循环。

Icon

在下一个示例中，我们将显示程序的图标。大多数的窗口管理器会在窗口标题栏左上角和任务栏上显示图标。

Code: icon.py



```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# This example shows an icon
# in the titlebar of the window
#
# author: jan bodnar
# website: zetcode.com
# last edited: February 2009


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()
```

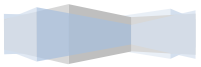
以上代码示例展示了程序图标。

```
    self.set_title("Icon")
```

我们为这个窗口设置一个标题。

```
    self.set_icon_from_file("web.png")
```

set_icon_from_file()方法是为窗口设置一个图标。图片被从磁盘当前工作目录被加载。



图片：icon

Buttons

在下个例子中，我们将进一步提高我们的 PyGTK 库编程技巧。

Code :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()
```

我们在窗口上展示了 4 个不同的按钮。我们将看见容器部件（container widgets）和子部件（child widgets）之间的不同，并且将会更改子部件的一些属性（properties ）。

```
    btn1 = gtk.Button("Button")
```

一个 **Button** 就是一个子部件。子部件被放置在容器内。

```
    btn1.set_sensitive(False)
```

我们使这个按钮不敏感（insensitive）。这意味着，我们不能点击它了，它也不能被选择、聚焦等。这个部件图形化地变灰。

```
    btn3 = gtk.Button(stock=gtk.STOCK_CLOSE)
```

第三个按钮在它的区域里显示了一个图片。PyGTK 库中有一个内置的图片库，我们可以使用它。（此处可以参考 The gtk Class Reference ）

```
    btn4.set_size_request(80, 40)
```

这里我更改了按钮的尺寸。

```
    fixed = gtk.Fixed()
```

Fixed 部件是一个不可见的容器部件（container widget）。它的用途是用来包含其它子部件。

```
    fixed.put(btn1, 20, 30)
    fixed.put(btn2, 100, 30)
    ...
```

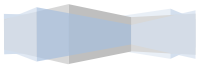
这里我们将按钮部件放置到 fixed 容器部件。

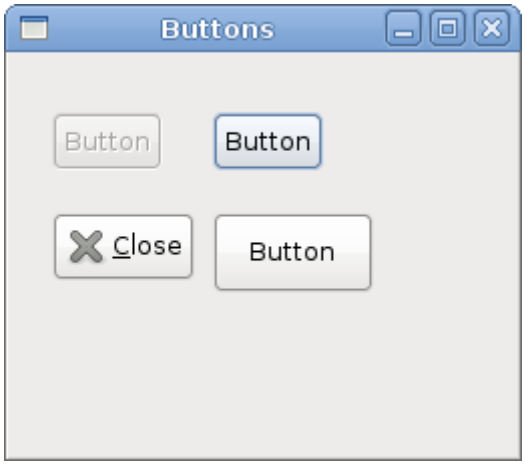
```
    self.add(fixed)
```

我们设置 **Fixed** 容器成为我们的 **Window** 部件的主容器。

```
    self.show_all()
```

要么我们调用 show_all()方法，要么就对每个部件，包括容器，调用 show()方法。





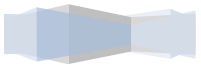
图片：Buttons

Tooltip

一个提示文本（Tooltip）就是在应用程序中对一个部件用途的建议。它能够被用来提供额外的帮助。

Code: 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)
```

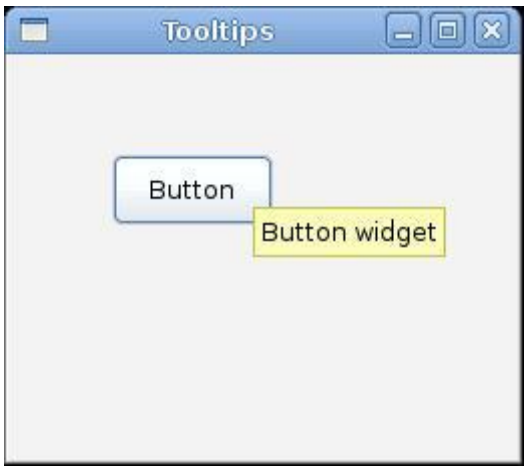



```
        self.set_tooltip_text("Window widget")
        button.set_tooltip_text("Button widget")

        self.show_all()

PyApp()
gtk.main()
```

在这个例子中，我们对一个窗口和一个按钮各设置了一个提示文本（tooltip）。

```
    self.set_tooltip_text("Window widget")
    button.set_tooltip_text("Button widget")
```

用 set_tooltip_text()方法做这项工作。



图片：Tooltips

在这章中，我们用 PyGTK 编程库创建了第一个程序。

02 PyGTK 中的布局管理

本教程在这章中，我们将怎样在窗口或者对话框（dialogs）中布置我们的部件。

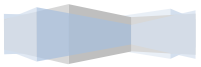
当我们设计程序的 GUI 界面时，我们决定哪些部件我们将会使用，和我们将怎样在程序中组织那些部件。为了组织我们的部件，我们使用专门的不可见部件，其被称为**布局容器**（**layout containers**）。在这章中，我们将提到 **Alignment, Fixed, VBox** 和 **Table** 这四种布局容器(**layout containers**)。

Fixed

Fixed 容器将放置位置固定和尺寸固定的子部件。这个容器不进行自动的布局管理。在大多数的程序中，我们不用这种容器。但是在一些专门的领域，我们会用它。例如游戏，一些工作在图表中的专门程序，那些能被移动可变化尺寸的组件（就想在电子表格程序中的一个 chart 表一样），小型的学习示例等。

Code: fixed.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# This example demonstrates a Fixed
# container widget
#
# author: jan bodnar
# website: zetcode.com
# last edited: February 2009

import gtk
import sys


class PyApp(gtk.Window):

    def __init__(self):
        super(PyApp, self).__init__()

        self.set_title("Fixed")
        self.set_size_request(300, 280)
        self.modify_bg(gtk.STATE_NORMAL, gtk.gdk.Color(6400, 6400, 6440))
        self.set_position(gtk.WIN_POS_CENTER)

        try:
            self.bardejov = gtk.gdk.pixbuf_new_from_file("bardejov.jpg")
            self.rotunda = gtk.gdk.pixbuf_new_from_file("rotunda.jpg")
            self.mincol = gtk.gdk.pixbuf_new_from_file("mincol.jpg")
        except Exception, e:
            print e.message
            sys.exit(1)


        image1 = gtk.Image()
        image2 = gtk.Image()
        image3 = gtk.Image()

        image1.set_from_pixbuf(self.bardejov)
        image2.set_from_pixbuf(self.rotunda)
        image3.set_from_pixbuf(self.mincol)

        fix = gtk.Fixed()

        fix.put(image1, 20, 20)
        fix.put(image2, 40, 160)
        fix.put(image3, 170, 50)

        self.add(fix)

        self.connect("destroy", gtk.main_quit)
        self.show_all()

PyApp()
gtk.main()
```

在我们的例子中，我们在窗口上显示了 3 个小图片。在我们放置这些图片的地方，我们明确地指定其 x，y 坐标。

```
self.modify_bg(gtk.STATE_NORMAL, gtk.gdk.Color(6400, 6400, 6440))
```

为了更好的视觉体验，我们更改了窗口的背景颜色为暗灰色。

```
self.bardejov = gtk.gdk.pixbuf_new_from_file("bardejov.jpg")
```

我们从磁盘中一个文件里载入图片。

```
image1 = gtk.Image()
image2 = gtk.Image()
image3 = gtk.Image()

image1.set_from_pixbuf(self.bardejov)
image2.set_from_pixbuf(self.rotunda)
image3.set_from_pixbuf(self.mincol)
```

Image 是一个部件，其作用是用来显示图片的。它在构造函数占用了一个 Pixbuf（图片缓存）对象。

```
fix = gtk.Fixed()
```

我们创建了 Fixed 容器。

```
fix.put(image1, 20, 20)
```

我们将第一张图片放置在坐标为 x=20,y=20 的位置上。

```
self.add(fix)
```

最后，我们将 Fixed 容器添加到窗口中。

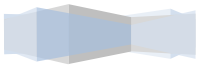
Figure: Fixed

Alignment

Alignment 容器控制它的子部件的对齐和尺寸。

Code: alignment.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# This example shows how to use
# the Alignment 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("Alignment")
        self.set_size_request(260, 150)
        self.set_position(gtk.WIN_POS_CENTER)

        vbox = gtk.VBox(False, 5)
        hbox = gtk.HBox(True, 3)

        valign = gtk.Alignment(0, 1, 0, 0)
        vbox.pack_start(valign)

        ok = gtk.Button("OK")
        ok.set_size_request(70, 30)
        close = gtk.Button("Close")

        hbox.add(ok)
        hbox.add(close)

        halign = gtk.Alignment(1, 0, 0, 0)
        halign.add(hbox)

        vbox.pack_start(halign, False, False, 3)

        self.add(vbox)

        self.connect("destroy", gtk.main_quit)
        self.show_all()

PyApp()
gtk.main()
```

在上述代码示例中，我们放置了两个按钮在窗口的右下角。为了完成这项任务，我们使用了一个水平箱子容器（horizontal box）和一个垂直箱子容器（vertical box）以及两个 alignment 容器。

```
valign = gtk.Alignment(0, 1, 0, 0)
```

这一步将会放置小部件到底部。

```
vbox.pack_start(valign)
```

在这里，我们 Alignment 容器放置到垂直箱子容器中。

```
hbox = gtk.HBox(True, 3)
...
ok = gtk.Button("OK")
ok.set_size_request(70, 30)
close = gtk.Button("Close")

hbox.add(ok)
hbox.add(close)
```

我们创建了一个水平箱子容器，并放置了两个按钮到其中。

```
halign = gtk.Alignment(1, 0, 0, 0)
halign.add(hbox)

vbox.pack_start(halign, False, False, 3)
```

这将创建一个 alignment 容器，这个容器将放置其子部件到右边。我们添加水平箱子容器到 alignment 容器中，并将 alignment 容器放置进垂直箱子容器中。我们必须记住的是：alignment 容器中只能有一个子部件。这就是为什么我们必须用 box 容器。

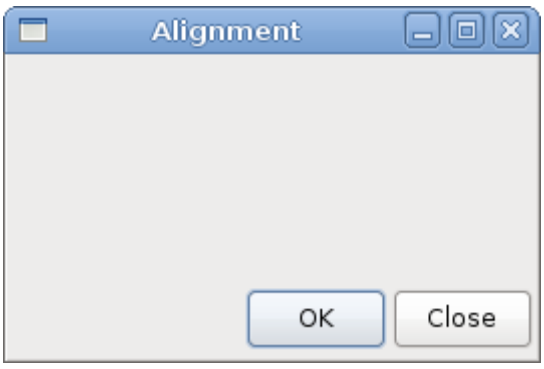


Figure: alignment

【译者理解：alignment 容器的位置按顺序分别是 gtk.alignment(右，下，上，左)】

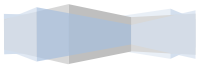
Table

Table 容器部件是将其部件安排在行和列中。

Code: calculator.py

```
#!/usr/bin/python
```

```
# ZetCode PyGTK tutorial
#
# This example shows how to use
# the Table container 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("Calculator")
        self.set_size_request(250, 230)
        self.set_position(gtk.WIN_POS_CENTER)

        vbox = gtk.VBox(False, 2)

        mb = gtk.MenuBar()
        filemenu = gtk.Menu()
        filem = gtk.MenuItem("File")
        filem.set_submenu(filemenu)
        mb.append(filem)

        vbox.pack_start(mb, False, False, 0)

        table = gtk.Table(5, 4, True)

        table.attach(gtk.Button("Cls"), 0, 1, 0, 1)
        table.attach(gtk.Button("Bck"), 1, 2, 0, 1)
        table.attach(gtk.Label(), 2, 3, 0, 1)
        table.attach(gtk.Button("Close"), 3, 4, 0, 1)

        table.attach(gtk.Button("7"), 0, 1, 1, 2)
        table.attach(gtk.Button("8"), 1, 2, 1, 2)
        table.attach(gtk.Button("9"), 2, 3, 1, 2)
        table.attach(gtk.Button("/"), 3, 4, 1, 2)

        table.attach(gtk.Button("4"), 0, 1, 2, 3)
        table.attach(gtk.Button("5"), 1, 2, 2, 3)
        table.attach(gtk.Button("6"), 2, 3, 2, 3)
        table.attach(gtk.Button("*"), 3, 4, 2, 3)

        table.attach(gtk.Button("1"), 0, 1, 3, 4)
        table.attach(gtk.Button("2"), 1, 2, 3, 4)
        table.attach(gtk.Button("3"), 2, 3, 3, 4)
        table.attach(gtk.Button("-"), 3, 4, 3, 4)

        table.attach(gtk.Button("0"), 0, 1, 4, 5)
        table.attach(gtk.Button("."), 1, 2, 4, 5)
        table.attach(gtk.Button("="), 2, 3, 4, 5)
```

```
        table.attach(gtk.Button("+"), 3, 4, 4, 5)

        vbox.pack_start(gtk.Entry(), False, False, 0)
        vbox.pack_end(table, True, True, 0)

        self.add(vbox)

        self.connect("destroy", gtk.main_quit)
        self.show_all()


PyApp()
gtk.main()
```

我们用 Table 容器部件创建了一个计算器的框架。

```
    table = gtk.Table(5, 4, True)
```

我们创建了一个 5 行 4 列的 table 容器部件。第三个参数是同质参数，如果被设置为 ture，table 中所有的部件将是相同的尺寸。而所有部件的尺寸与 table 容器中最大的部件的尺寸相同。

```
    table.attach(gtk.Button("Cls"), 0, 1, 0, 1)
```

我们附加了一个按钮到 table 容器中，其位置在表格的左上单元（cell）。前面两个参数代表这个单元的左侧和右侧，后两个参数代表这个单元的上部和下部。【译者：此处表述比较难以理解，待研究。。。】

```
    vbox.pack_end(table, True, True, 0)
```

我们将 table 容器部件放置到垂直箱子容器中。

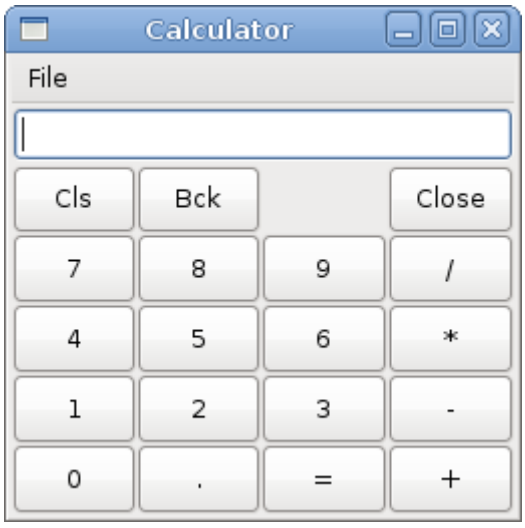


Figure: Calculator skeleton

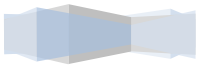
Windows

下面我们将创建一个更加高级的例子。我们展示了一个窗口，其可以在 JDeveloper 集

成开发环境（IDE）中被找到。

Code: windows.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# This is a more complicated layout
# example
#
# author: jan bodnar
# website: zetcode.com
# last edited: February 2009

import gtk
import sys


class PyApp(gtk.Window):

    def __init__(self):
        super(PyApp, self).__init__()

        self.set_title("Windows")
        self.set_size_request(300, 250)
        self.set_border_width(8)
        self.set_position(gtk.WIN_POS_CENTER)

        table = gtk.Table(8, 4, False)
        table.set_col_spacings(3)

        title = gtk.Label("Windows")

        halign = gtk.Alignment(0, 0, 0, 0)
        halign.add(title)

        table.attach(halign, 0, 1, 0, 1, gtk.FILL,
            gtk.FILL, 0, 0);

        wins = gtk.TextView()
        wins.set_editable(False)
        wins.modify_fg(gtk.STATE_NORMAL, gtk.gdk.Color(5140, 5140, 5140))
        wins.set_cursor_visible(False)
        table.attach(wins, 0, 2, 1, 3, gtk.FILL | gtk.EXPAND,
            gtk.FILL | gtk.EXPAND, 1, 1)

        activate = gtk.Button("Activate")
        activate.set_size_request(50, 30)
        table.attach(activate, 3, 4, 1, 2, gtk.FILL,
            gtk.SHRINK, 1, 1)

        valign = gtk.Alignment(0, 0, 0, 0)
        close = gtk.Button("Close")
        close.set_size_request(70, 30)
        valign.add(close)
        table.set_row_spacing(1, 3)
```

```
        table.attach(valign, 3, 4, 2, 3, gtk.FILL,
            gtk.FILL | gtk.EXPAND, 1, 1)

        halign2 = gtk.Alignment(0, 1, 0, 0)
        help = gtk.Button("Help")
        help.set_size_request(70, 30)
        halign2.add(help)
        table.set_row_spacing(3, 6)
        table.attach(halign2, 0, 1, 4, 5, gtk.FILL,
            gtk.FILL, 0, 0)

        ok = gtk.Button("OK")
        ok.set_size_request(70, 30)
        table.attach(ok, 3, 4, 4, 5, gtk.FILL,
            gtk.FILL, 0, 0);

        self.add(table)

        self.connect("destroy", gtk.main_quit)
        self.show_all()


PyApp()
gtk.main()
```

上述代码示例中展示了，我们在 PyGTK 中怎样创建相似的窗口。

```
    table = gtk.Table(8, 4, False)
    table.set_col_spacings(3)
```

这个例子是基于 table 容器的。在每一列中间将有 3 个像素（px）的间隔。

```
    title = gtk.Label("Windows")

    halign = gtk.Alignment(0, 0, 0, 0)
    halign.add(title)

    table.attach(halign, 0, 1, 0, 1, gtk.FILL,
        gtk.FILL, 0, 0);
```

上述代码创建了一个文本标签（label），其被排在左边。这个文本标签（label）被放置在 Table 容器的第一行。

```
    wins = gtk.TextView()
    wins.set_editable(False)
    wins.modify_fg(gtk.STATE_NORMAL, gtk.gdk.Color(5140, 5140, 5140))
    wins.set_cursor_visible(False)
    table.attach(wins, 0, 2, 1, 3, gtk.FILL | gtk.EXPAND,
        gtk.FILL | gtk.EXPAND, 1, 1)
```

上述代码中的文本视图（TextView）部件跨越了两行两列。我们使其不可编辑，并隐藏了光标。


```
    valign = gtk.Alignment(0, 0, 0, 0)
    close = gtk.Button("Close")
    close.set_size_request(70, 30)
    valign.add(close)
```

```
    table.set_row_spacing(1, 3)
    table.attach(valign, 3, 4, 2, 3, gtk.FILL,
        gtk.FILL | gtk.EXPAND, 1, 1)
```

我们将 close 按钮放进紧接着文本视图部件的第四列中（我们从 0 数起）。我们将按钮添加进 alignment 部件，这样我们就能将它排在顶部。

Figure: Windows

PyGTK 编程教程的这章是关于布局管理（layout management）。

03 PyGTK 中的菜单

在本教程的这个部分，我们将在菜单下工作

Menubar（菜单栏）是 GUI 程序最常见部分中的一个。它是位于各种菜单中的一组命令。当在控制台程序中，你必须记住所以那些晦涩难懂的命令，而在菜单里我们将大多数命令分组进入合乎逻辑的部分。因此，有一个公认的标准，那就是菜单能进一步减少大量的时间去学习一个新的程序。

Simple menu

在我们的第一个例子中，我们将创建一个只有一个文件菜单（file menu）的的菜单栏（menubar）。而这个菜单也仅仅只有一个菜单项目，通过选择这个菜单项，程序将会退出。

Code: 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()
```

这是一个包含最小菜单栏功能的的小示例。

```
    mb = gtk.MenuBar()
```

MenuBar 部件被创建。

```
    filemenu = gtk.Menu()
    filem = gtk.MenuItem("File")
    filem.set_submenu(filemenu)
```

顶层的 MenuItem 部件被创建。

```
    exit = gtk.MenuItem("Exit")
    exit.connect("activate", gtk.main_quit)
    filemenu.append(exit)
```


退出 MenuItem 项被创建，并且被附加在文件 MenuItem 之下。

```
    mb.append(filem)
```

顶层的 MenuItem 被追加到 MenuBar 部件之上

```
    vbox = gtk.VBox(False, 2)
    vbox.pack_start(mb, False, False, 0)
```

不想其它的工具包，我们得小心地布局我们的菜单栏。我们将菜单栏放进垂直箱子容器。

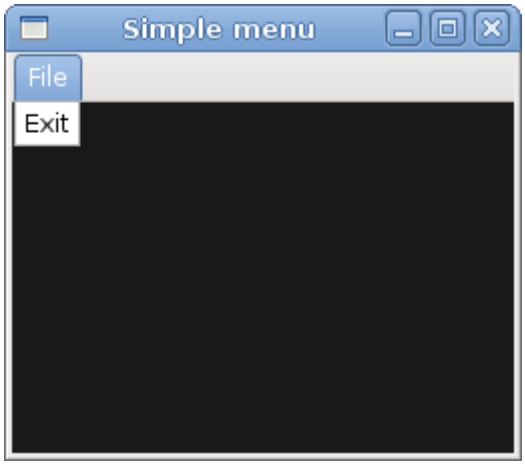


Figure: Simple Menu

Image Menu

在下一个例子中，我们将更深入的探究菜单。我们将增加图片和快捷键到我们的菜单项。Accelerators 就是为了激活菜单项的键盘快捷键。

Code: 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()
```

我们的示例展示了一个顶层菜单项，其包含了 3 个次级菜单项。每一个菜单都包含一个图片和一个快捷键。退出菜单项的快捷键是激活的。

```
    agr = gtk.AccelGroup()
    self.add_accel_group(agr)
```


为了使用快捷键，我们创建了一个全局的 AccelGroup 对象，它将在之后被使用。

```
    newi = gtk.ImageMenuItem(gtk.STOCK_NEW, agr)
```

ImageMenuItem 被创建。图片来自于内置的图片库。

```
key, mod = gtk.accelerator_parse("<Control>N")
```

函数 gtk.accelerator_parse()，是从语法上分析指定的快捷键字符，并且返回一个含 2 个元素的元组，两个元素分别为：keyval（字符键对应的键值，如：N=110，O=111 等）和与快捷键相关的修饰控制键（如：Control, Alt 等）的对象。【此处（ ）中为译者测试结果】

```
newi.add_accelerator("activate", agr, key,
    mod, gtk.ACCEL_VISIBLE)
```

以上代码，为退出菜单项创建了 Ctrl+Q 快捷键。

```
sep = gtk.SeparatorMenuItem()
filemenu.append(sep)
```

这些代码创建了一个分隔符，其用于将菜单项分成逻辑组。

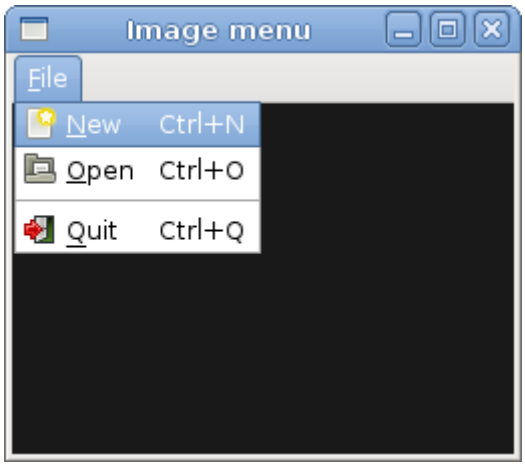


Figure: Image Menu

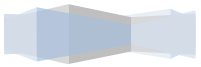
CheckMenuItem

CheckMenuItem 就是一个包含复选框的菜单项。它被用来工作于布尔属性（boolean properties）。

Code: 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()
```

在我们的代码示例中，我们展示了一个复选框菜单项。如果复选框被激活，则状态栏部件将被显示，否则被隐藏。

```
stat = gtk.CheckMenuItem("View Statusbar")
```

CheckMenuItem 部件被创建。

```
stat.set_active(True)
```

set_active()方法是用于激活或者不激活复选框菜单项。

```
if widget.active:
    self.statusbar.show()
else:
    self.statusbar.hide()
```

依据 CheckMenuItem 的激活属性，我们可以显示或者隐藏状态栏部件。

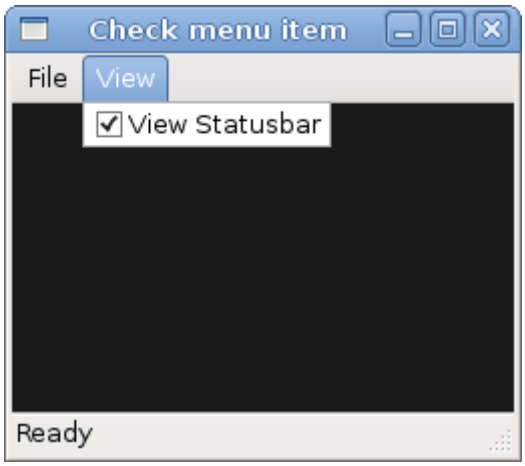


Figure: CheckMenuItem

SubMenu

在最后的例子中，我们证明了在 PyGTK 中怎样创建了一个次级菜单。

Code: 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()
```

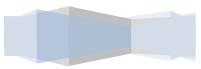
创建次级菜单。

```
    imenu = gtk.Menu()
```

一个次级菜单就是一个 Menu

```
    importm = gtk.MenuItem("Import")
    importm.set_submenu(imenu)
```

这是一个菜单项的次级菜单，其属于顶级文件菜单。



```
    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)
```

次级菜单有其自己的菜单项。

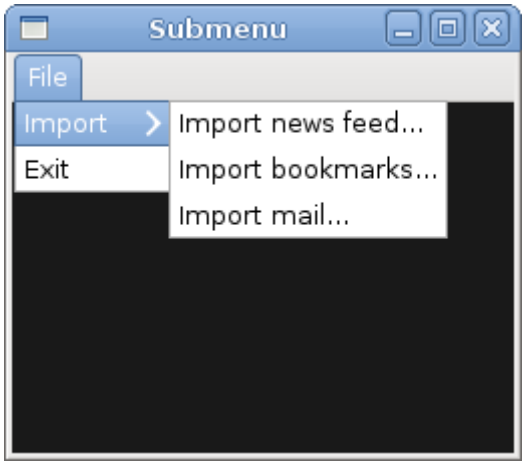


Figure: Submenu

在 PyGTK 编程教程的这章中，我们展示了怎样用菜单工作。

04 PyGTK 中的工具栏

在 PyGTK 教程的这部分，我们将和工具栏一起工作。

在应用程序中，我们可以使用菜单组合命令。而工具栏提供了一个更快捷的方式，来访问使用频率很高的惯用的命令。

Simple toolbar

下面我们创建了一个简单的工具栏

Code: 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()
```

这个例子展示了一个工具栏，其中包括四个工具按钮。

```
    toolbar = gtk.Toolbar()
```

一个 **Toolbar** 部件被创建。

```
    toolbar.set_style(gtk.TOOLBAR_ICONS)
```

在工具栏上，我们让它仅仅展示图标，而没有文本。

```
    newtb = gtk.ToolButton(gtk.STOCK_NEW)
```

一个工具按钮被创建，它包含一个来自 stock（gtk 内部储备）的图片。

```
    sep = gtk.SeparatorToolItem()
```


这是一个分隔符，它能将工具栏上的按钮分成逻辑组。

```
    toolbar.insert(newtb, 0)
    toolbar.insert(opentb, 1)
```

```
...
```

工具栏按钮被插入到工具栏部件中了。

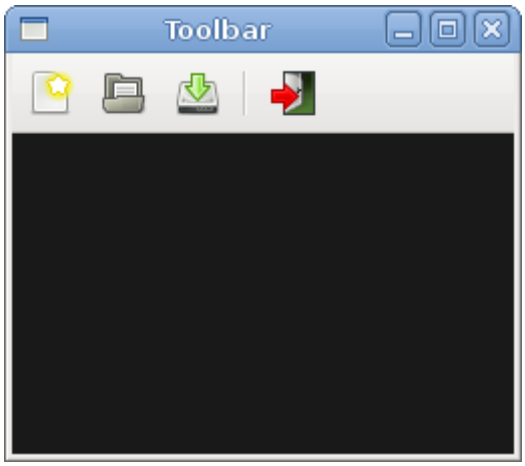


Figure: Toolbar

Toolbars

在第二个列子中，我们将展示两个工具栏。许多应用程序有超过一个的工具栏。我们将展示在 PyGTK 我们怎样做到这样。

Code：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()
```

我们的程序展示了两个工具栏。

```
    upper = gtk.Toolbar()
    ...
    lower = gtk.Toolbar()
```

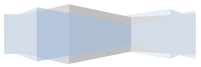
我们创建了两个工具栏部件。

```
    upper.insert(newtb, 0)
    ...
    lower.insert(quittb, 0)
```

它们中的每一个都有自己的工具按钮。

```
    vbox = gtk.VBox(False, 0)
    vbox.pack_start(upper, False, False, 0)
    vbox.pack_start(lower, False, False, 0)
```

工具栏被放置进垂直的箱子里，一个接着另一个。



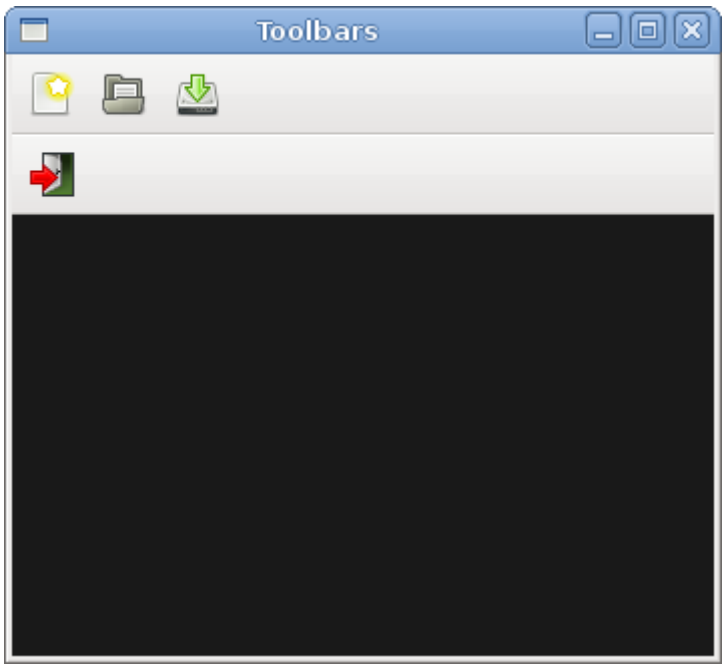


Figure: Toolbars

Undo redo

下面的列子将论证，我们怎样使在工具栏上的按钮变成不活动的状态。在 GUI 编程中，这是个常规的实践。例如保存按钮。如果我们将文档的所有变化保存到了硬盘，那么保存按钮在大多数的编辑器中会被设置为不活动状态。这种方式是应用程序提示用户，所有的更改都已经被保存了。

Code: ubdoredo.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# This example shows how to
# activate/deactivate a ToolButton
#
# 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)

        self.count = 2

        toolbar = gtk.Toolbar()
        toolbar.set_style(gtk.TOOLBAR_ICONS)

        self.undo = gtk.ToolButton(gtk.STOCK_UNDO)
        self.redo = gtk.ToolButton(gtk.STOCK_REDO)
        sep = gtk.SeparatorToolItem()
        quit = gtk.ToolButton(gtk.STOCK_QUIT)

        toolbar.insert(self.undo, 0)
        toolbar.insert(self.redo, 1)
        toolbar.insert(sep, 2)
        toolbar.insert(quit, 3)

        self.undo.connect("clicked", self.on_undo)
        self.redo.connect("clicked", self.on_redo)
        quit.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()

    def on_undo(self, widget):
        self.count = self.count - 1

        if self.count <= 0:
            self.undo.set_sensitive(False)
            self.redo.set_sensitive(True)


    def on_redo(self, widget):
        self.count = self.count + 1

        if self.count >= 5:
            self.redo.set_sensitive(False)
            self.undo.set_sensitive(True)

PyApp()
gtk.main()
```

我们从 PyGTK 的 stock 图片资源中创建了撤销和重做按钮。对每个按钮进行几次点击之后，它们变成不活动状态。按钮呈现为灰色。

```
    self.count = 2
```


self.count 变量决定，哪个按钮是活动的或者不活动的。

```
    self.undo = gtk.ToolButton(gtk.STOCK_UNDO)
    self.redo = gtk.ToolButton(gtk.STOCK_REDO)
```

我们有两个工具按钮，撤销和重做工具按钮。图片来自 stock 资源。

```
self.undo.connect("clicked", self.on_undo)
self.redo.connect("clicked", self.on_redo)
```

我们为两个工具按钮的 **clicked**（点击）信号都连接了一个方法。

```
if self.count <= 0:
    self.undo.set_sensitive(False)
    self.redo.set_sensitive(True)
```

为了激活一个部件，我们使用了 set_sensitive()方法。

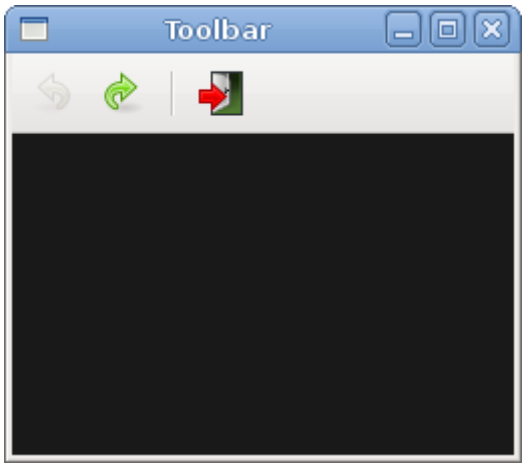


Figure: Undo redo

在 PyGTK 编程库教程的这章中，我们提到了工具栏。

05 PyGTK 中的信号和事件

在 PyGTK 教程的这章中，我们将讨论信号和事件。

所有的 GUI 应用程序都是事件驱动，PyGTK 应用程序也不列外。程序调用 **gtk.main()** 函数开始一个主循环。该循环不断地检查新产生的事件。如果没有事件，程序将等待，什么都不做。

Events（事件）是从 X 服务到程序之间的消息。当我们点击一个按钮部件，这个点击（**clicked**）信号将会被发出（be **emitted**）。有些信号是所有的部件都继承，例如 **destroy**（销毁部件）；有些信号是部件所特有的，例如 toggled 信号是 toggle 按钮所特有的。

程序员用信号处理函数（signal **handlers**）来对各种信号进行交互（react）。这些处理函数在回收前被 GTK 程序员调用。【译者注：此处翻译有些别扭，英文比较难以理解】

```
handler_id = button.connect("clicked", self.on_clicked)
```

这里我们使用 GObject 类（ GtkButton 是一个 GObject 类的子类 ）的 **connect()**方法，来连接一个回调函数 **on_clicked()**到被称为 **clicked** 的信号上。

connect()方法返回一个 handler id，它被用来唯一地表式这个回调方法。这个 id 可以被下列方法使用：

```
def disconnect(handler_id)
```

```
def handler_disconnect(handler_id)
def handler_is_connected(handler_id)
def handler_block(handler_id)
def handler_unblock(handler_id)
```

这些方法能够从一个 GObject 部件断开其处理函数，或者阻塞、放行它。

Signals vs events

关于这两者之间的不同，我们往往有许多困惑。

信号(signals)和事件(events)是两个不同的事情。一个事件是窗口系统事件中的一个一对一的映射。摁下键盘键，更改窗口大小或者摁下按钮都是典型的窗口系统事件。窗口系统事件被报告给应用程序的主循环。GDK 解释这些窗口系统事件，并且通过信号传递过去。

一个信号是一个回调机制。如果一个对象想要被通知关于另外一个对象的活动或状态变化，它就注册一个回调信号（callback）。当这个对象发出了一个信号，它就查找回调信号列表，这些列表中的信号已经被注册过了，然后为特定的信号调用回调函数。它能够有选择地发送一些事先定义好的数据。

信号（signals）是一种通用目的通知框架。它们不仅被用作 UI 变化的通知，还被用作应用程序状态变化的通知。信号机制（signals）是通用的，也是强大的，它们的使用非常广泛。任何 GObject 对象都能发射或者接受一个对象。某种类型的对象可能有一个或者多个信号，其中每一个都可能有一个参数列表和返回值。然后处理函数（handlers）能被连接到这种类型的对象的实例。当一个信号在一个实例上被发射，每个已经连接的处理函数将会被调用。

信号机制和事件机制之间唯一的关系是，信号被用来发射来自 X 服务的事件的通知。

信号机制是 gtk.GObject 的一种特征，也是它的一个子类。事件是一个 Gdk/Xlib 的概念。

Simple example

下面的例子将展示，我们在两个基本的信号之间进行交互。

Code: quitbutton.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# The example shows how to work with
# destroy and clicked signals
#
# 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("Quit Button")
        self.set_size_request(250, 200)
        self.set_position(gtk.WIN_POS_CENTER)
        self.connect("destroy", self.on_destroy)

        fixed = gtk.Fixed()

        quit = gtk.Button("Quit")
        quit.connect("clicked", self.on_clicked)
        quit.set_size_request(80, 35)

        fixed.put(quit, 50, 50)

        self.add(fixed)
        self.show_all()

    def on_destroy(self, widget):
        gtk.main_quit()

    def on_clicked(self, widget):
        gtk.main_quit()


PyApp()
gtk.main()
```

当我们关闭窗口的时候，销毁（destroy）信号被触发。而默认情况下，当我们点击标题栏上的关闭按钮，程序不会退出。

```
    self.connect("destroy", self.on_destroy)
```

connect()方法将 **on_destroy()**方法连接到 **destroy** 信号上。

```
    quit.connect("clicked", self.on_clicked)
```

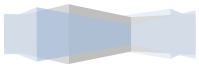
按下退出按钮，clicked 信号被激发。当我们点击退出按钮，我们就调用了 on_clicked()方法。

```
    def on_destroy(self, widget):
        gtk.main_quit()
```

在 on_destroy()方法中 ,我们对 destroy 信号作出反应。我们调用了 gtk.main_quit()方法，它使我们的应用程序终止。

```
    def on_clicked(self, widget):
        gtk.main_quit()
```

这里是 on_clicked()方法。它获得了两个参数，其中 widget 参数就是这个激发该信号的对象，在我们的例子中，它就是 quit 按钮。不同的对象发送不同的信号。发送到方法函数的信号和参数能够在 PyGTK 库的参考手册上找到。参考手册网址是：http://pygtk.org/docs/pygtk/index.html



Creating a custom signal

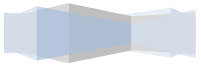
在下面的代码示例中，我们创建并发送了一个定制的信号(custom signal)。

Code： customsignal.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# This example shows how to create
# and send a custom singal
#
# author: jan bodnar
# website: zetcode.com
# last edited: February 2009

import gobject


class Sender(gobject.GObject):
    def __init__(self):
        self.__gobject_init__()


gobject.type_register(Sender)
gobject.signal_new("z_signal", Sender, gobject.SIGNAL_RUN_FIRST,
                   gobject.TYPE_NONE, ())


class Receiver(gobject.GObject):
    def __init__(self, sender):
        self.__gobject_init__()

        sender.connect('z_signal', self.report_signal)

    def report_signal(self, sender):
        print "Receiver reacts to z_signal"


def user_callback(object):
    print "user callback reacts to z_signal"

if __name__ == '__main__':

    sender = Sender()
    receiver = Receiver(sender)

    sender.connect("z_signal", user_callback)
    sender.emit("z_signal")
```

我们创建了两个 GObject 对象，sender 和 receiver 对象。Sender 发射了一个信号，这个信号被 receiver 接收了。我们也将一个回调函数连接到了这个信号。

```
class Sender(gobject.GObject):
    def __init__(self):
```

```
        self.__gobject_init__()
```

这是发送者对象，他通过默认的构造函数被创建。

```
gobject.type_register(Sender)
gobject.signal_new("z_signal", Sender, gobject.SIGNAL_RUN_FIRST,
                   gobject.TYPE_NONE, ())
```

我们注册了一个新的对象和一个新的信号。Signal_new()函数为 Sender 对象注册了一个被称为 z_signal 的信号。这个 SIGNAL_RUN_FIRST 参数意思是，接收信号对象的默认处理函数作为第一次被调用。后面两个参数是返回值类型和参数类型。在我们的例子中，我们不返回任何值，并且不发送参数。

```
sender.connect('z_signal', self.report_signal)
```

接收者 receiver 对象监听 z_signal 信号。

```
sender = Sender()
receiver = Receiver(sender)
```

Sender 和 Receiver 对象被实例化。receiver 将 sender 作为参数，因此 receiver 能够监听 sender 的信号。

```
sender.connect("z_signal", user_callback)
```

这里我们将这个信号连接到用户的回调函数。

```
sender.emit("z_signal")
```

z_signal 信号正在被发射。

```
class Sender(gobject.GObject):

    __gsignals__ = {
        'z_signal': (gobject.SIGNAL_RUN_LAST, gobject.TYPE_NONE, ()),
    }

    def __init__(self):
        self.__gobject_init__()

gobject.type_register(Sender)
```

我们也可以用**__gsignals__**类的属性来注册一个新的信号。

Predefined signal handlers

在 PyGTK 中的某些对象可能有预定义的信号处理函数(predefined signal handlers)。这些函数的名称以 do_*开始，例如：do_expose()，do_show()或者 do_clicked()等。

Code: move.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# This example overrides predefined
# do_configure_event() signal handler
#
# author: jan bodnar
# website: zetcode.com
# last edited: February 2009


import gtk
import gobject

class PyApp(gtk.Window):
    __gsignals__ = {
        "configure-event" : "override"
        }

    def __init__(self):
        super(PyApp, self).__init__()

        self.set_size_request(200, 150)
        self.set_position(gtk.WIN_POS_CENTER)

        self.connect("destroy", gtk.main_quit)

        self.show_all()

    def do_configure_event(self, event):

        title = "%s, %s" % (event.x, event.y)
        self.set_title(title)
        gtk.Window.do_configure_event(self, event)


PyApp()
gtk.main()
```

当我们移动窗口和更改窗口大小的时候后，X 服务将会发送配置事件(configure events)，然后它们就被传送到 **configure-event** 信号。

在我们的示例代码中，我们在标题栏上展示了窗口的左上角的 x, y 坐标。我们可以很简单地将一个信号处理函数连接到 **configure-event** 信号，但是我们采用了一个不同的策略。在实现逻辑上的需要的地方，我们重写了默认的类处理函数。

```
__gsignals__ = {
    "configure-event" : "override"
    }
```

这里说明，我们将要重写默认的 **on_configure_event()**方法。


```
def do_configure_event(self, event):

    title = "%s, %s" % (event.x, event.y)
```

```
    self.set_title(title)
    gtk.Window.do_configure_event(self, event)
```

此处我们重写了预定义的 **do_configure_event()**方法。我们设置窗口的坐标显示到窗口的标题上。当然也要注意最后一行，它明确地调用了父类的 **do_configure_event()**方法。这是因为它还做了重要的工作。请尝试注释掉这一行，看看发生了什么。改变窗口大小将不能正确工作【译者注：在 Win 下尝试似乎都能正确工作】。如果我们重写了默认的处理函数，我有可能会或者不会调用父类的方法，在这个例子中，我们必须要调用。

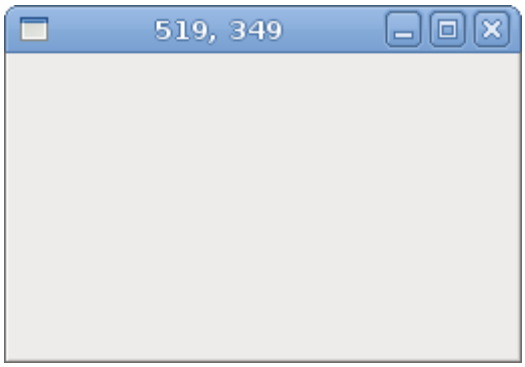


Figure: Configure signal

Signals of a button

下面的例子将展示各种各样的按钮信号。

Code: buttonsignals.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# This program shows various signals
# of a button widget
# It emits a button-release-event which
# triggers a released singal
#
# 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("Signals")
```

```
        self.set_size_request(250, 200)
        self.set_position(gtk.WIN_POS_CENTER)
        self.connect("destroy", gtk.main_quit)

        fixed = gtk.Fixed()

        self.quit = gtk.Button("Quit")

        self.quit.connect("pressed", self.on_pressed)
        self.quit.connect("released", self.on_released)
        self.quit.connect("clicked", self.on_clicked)

        self.quit.set_size_request(80, 35)

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

        self.add(fixed)
        self.show_all()
        self.emit_signal()

    def emit_signal(self):

        event = gtk.gdk.Event(gtk.gdk.BUTTON_RELEASE)
        event.button = 1
        event.window = self.quit.window
        event.send_event = True

        self.quit.emit("button-release-event", event)


    def on_clicked(self, widget):
        print "clicked"

    def on_released(self, widget):
        print "released"

    def on_pressed(self, widget):
        print "pressed"


PyApp()
gtk.main()
```

一个按钮能够发出不止一种类型的信号。我们这里用它们的三种信号进行工作，分别是 **clicked**, **pressed** 和 **released** 信号。我们也展示了一个事件信号激发另外一个信号。

```
self.quit.connect("pressed", self.on_pressed)
self.quit.connect("released", self.on_released)
self.quit.connect("clicked", self.on_clicked)
```

我们对所有三个信号注册了回调函数。

```
self.emit_signal()
```


程序开始运行的时候，我们激发了一个特别的信号。

```
def emit_signal(self):
```

```
    event = gtk.gdk.Event(gtk.gdk.BUTTON_RELEASE)
    event.button = 1
    event.window = self.quit.window
    event.send_event = True

    self.quit.emit("button-release-event", event)
```

我们激发 **button-release-event** 信号，它将一个 Event 对象作为参数。当这个程序启动之后，我们应该在控制台窗口上看到"released"文本。当我们点击一下按钮，所有三个信号都被激发了。

Blocking an event handler

我们能够阻塞一个信号的处理函数，下面的例子将展示这个。

Code: block.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# This example shows how to block/unblock
# a signal handler
#
# 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("Blocking a callback")
        self.set_size_request(250, 180)
        self.set_position(gtk.WIN_POS_CENTER)

        fixed = gtk.Fixed()
        button = gtk.Button("Click")
        button.set_size_request(80, 35)
        self.id = button.connect("clicked", self.on_clicked)
        fixed.put(button, 30, 50)

        check = gtk.CheckButton("Connect")
        check.set_active(True)
        check.connect("clicked", self.toggle_blocking, button)
        fixed.put(check, 130, 50)


        self.connect("destroy", gtk.main_quit)
```

```
        self.add(fixed)
        self.show_all()

    def on_clicked(self, widget):
        print "clicked"

    def toggle_blocking(self, checkbox, button):
        if checkbox.get_active():
           button.handler_unblock(self.id)
        else:
           button.handler_block(self.id)

PyApp()
gtk.main()
```

在这个代码示例中，我们有一个按钮和一个复选框。当我们点击按钮并且复选框是选中状态时，我们在控制台上显示了"clicked"的文本。这个复选框阻塞或者放行来自按钮 clicked 信号和它的处理函数方法之间的连接。

```
self.id = button.connect("clicked", self.on_clicked)
```

connect()方法返回这个处理函数的 id， 这个 id 用来阻塞或者放行这个处理。

```
def toggle_blocking(self, checkbox, button):
    if checkbox.get_active():
       button.handler_unblock(self.id)
    else:
       button.handler_block(self.id)
```

这几行代码是阻塞和放行相应的方法的回调。

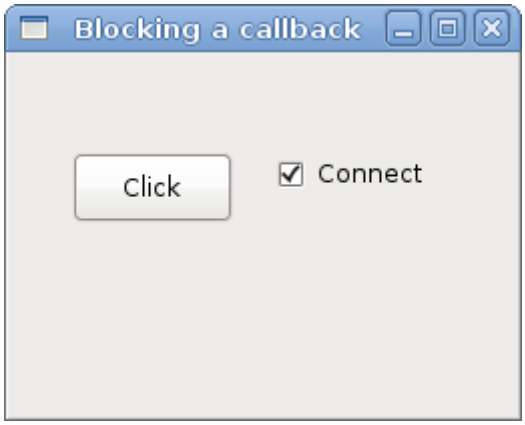


Figure: blocking a callback

06 PyGTK 中的部件

在 PyGTK 教程的这个部分，我们将介绍一些 PyGTK 的部件。

部件是 GUI 应用程序基本的构建单元。经过这些年的发展，有些部件已经成为所有的操作系统平台上的所有工具包的一个标准。例如：按钮，复选框或者滚动条。PyGTK 工具包的设计哲学是在一个最低的水平保持部件的数量。而比较特殊的部件都将作为一个自定义部件被创建。

Label

标签（Label）部件就是显示数量有限的只读的文本。

Code: 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()
```

上面的代码示例在窗口上展示了一些歌词。

```
lyrics = """Meet you downstairs in the bar and heard
your rolled up sleeves and your skull t-shirt
..."""
```

这是我们要显示的文本。

```
self.set_border_width(8)
```

这个标签被一些空白空间包围着。

```
label = gtk.Label(lyrics)
self.add(label)
```

这个标签被创建，并且被添加到窗口中。

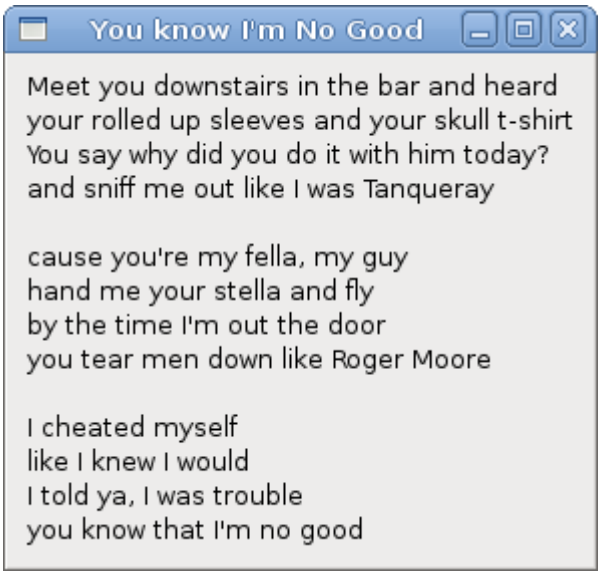


Figure: Label widget

CheckButton

复选按钮（也叫复选框）是一种有两个状态的部件，开和关。打开状态是被一个复选标记显示出来。它一般用来指示一些布尔属性。

Code: 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()
```

我们将会依据复选按钮的状态在窗口的标题栏上显示一个标题。

```
button = gtk.CheckButton("Show title")
```

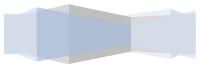
复选按钮部件被创建。

```
button.set_active(True)
```

标题在默认状态下是可见的，因此默认选中复选按钮。

```
if widget.get_active():
    self.set_title("Check Button")
else:
    self.set_title("")
```

如果复选按钮是选中的，我们就显示标题，否则我们将标题栏上的标题设置为空的。



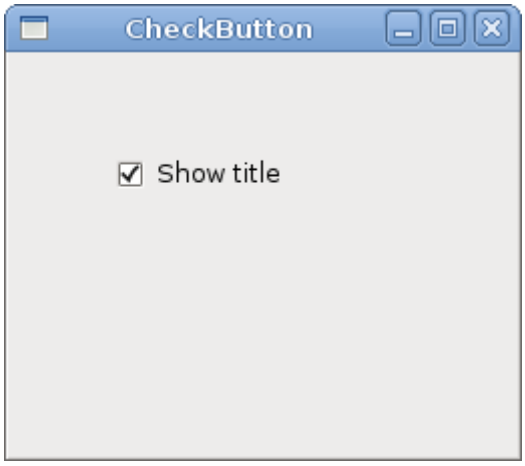


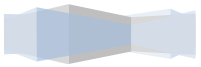
Figure: Check button

ComboBox

组合框是一种允许用户从一个选项列表中做选择的部件。

Code: 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()
```

这个例子展示了一个组合框和一个标签。组合框有一个六个选项的列表。这些都是 Linux 发行版的名字。标签展示从组合框中被选中的选项。

```
cb = gtk.combo_box_new_text()
```

这个 **gtk.combo_box_new_text()**函数是一个便利函数，它构造了一个新的文本组合框。它是一个仅仅显示字符串的 **ComboBox**。

```
cb.append_text('Ubuntu')
cb.append_text('Mandriva')
cb.append_text('Redhat')
cb.append_text('Gentoo')
cb.append_text('Mint')
```

这个复选框被填充了文本的数据。

```
self.label.set_label(widget.get_active_text())
```

在 on_changed()方法中，我们从组合框得到选中的文本，并且将它设置到标签中。

Figure: ComboBox

Image

下面的例子是介绍图像部件。这个部件是用来显示图片的。

Code: 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()
```

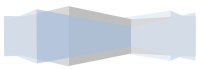
我们在窗口中展示了 Red Rock 城堡。

```
image = gtk.Image()
```

Image 部件被创建。

```
image.set_from_file("redrock.png")
```

我们为 Image 部件设置了一个 png 格式的图片。这个图片是从磁盘上的文件中加载的。



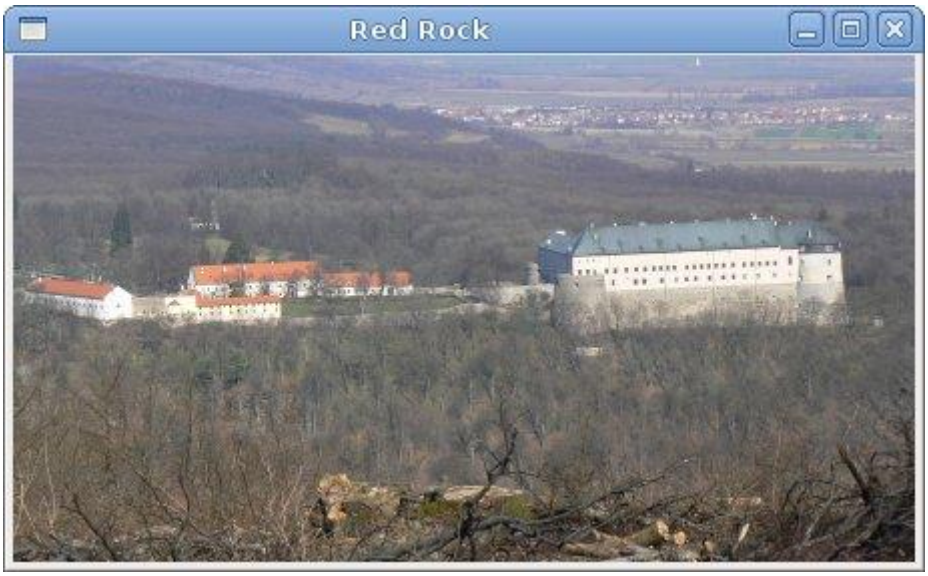


Figure: Image

在这个章节中，我们展示了 PyGTK 编程库的基本部件的第一包。

07 PyGTK 中的部件 II

在 PyGTK 教程的这个部分，我们将继续介绍 PyGTK 的部件。

Entry

Entry 就是一个单行文本输入框。这个部件主要用来输入文本数据。

Code: 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()
```

这个例子展示了一个 entry 和一个 label。我们键入到 entry 中文本会立刻在 label 控制区显示出来。

```
entry = gtk.Entry()
```

Entry 部件被创建。

```
entry.connect("key-release-event", self.on_key_release)
```

如果 entry 部件中的文本被改变，我们就调用 on_key_release()方法。

```
def on_key_release(self, widget, event):
    self.label.set_text(widget.get_text())
```

我们从 entry 部件得到文本，并且将它设置给 label 部件。

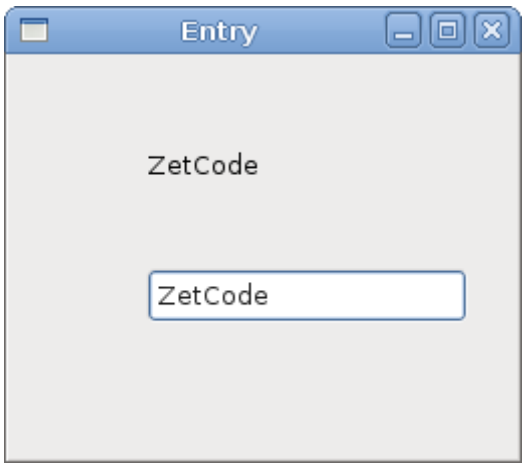


Figure: Entry Widget

HScale

HScale 就是一个水平滑动器。它让用户以图形的方式，在有限的间距下，通过移动滑动块来选择一个值。我们的例子展示了一个声音的控制器。

Code: hscale.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# This example demonstrates the HScale widget
#
# author: jan bodnar
# website: zetcode.com
# last edited: February 2009


import gtk
import sys


class PyApp(gtk.Window):

    def __init__(self):
        super(PyApp, self).__init__()

        self.set_title("Scale")
        self.set_size_request(260, 150)
        self.set_position(gtk.WIN_POS_CENTER)

        scale = gtk.HScale()
        scale.set_range(0, 100)
        scale.set_increments(1, 10)
        scale.set_digits(0)
        scale.set_size_request(160, 35)
        scale.connect("value-changed", self.on_changed)

        self.load_pixbufs()

        self.image = gtk.Image()
        self.image.set_from_pixbuf(self.mutp)

        fix = gtk.Fixed()
        fix.put(scale, 20, 40)
        fix.put(self.image, 219, 50)

        self.add(fix)

        self.connect("destroy", lambda w: gtk.main_quit())
        self.show_all()

    def load_pixbufs(self):
```

```
        try:
            self.mutp = gtk.gdk.pixbuf_new_from_file("mute.png")
            self.minp = gtk.gdk.pixbuf_new_from_file("min.png")
            self.medp = gtk.gdk.pixbuf_new_from_file("med.png")
            self.maxp = gtk.gdk.pixbuf_new_from_file("max.png")

        except Exception, e:
            print "Error reading Pixbufs"
            print e.message
            sys.exit(1)


    def on_changed(self, widget):
        val = widget.get_value()

        if val == 0:
            self.image.set_from_pixbuf(self.mutp)
        elif val > 0 and val <= 30:
            self.image.set_from_pixbuf(self.minp)
        elif val > 30 and val < 80:
            self.image.set_from_pixbuf(self.medp)
        else:
            self.image.set_from_pixbuf(self.maxp)


PyApp()
gtk.main()
```

在上面的例子中，我们有一个 **HScale** 和一个 **Image** 部件。通过拖动这个标尺，我们可以改变 Image 部件中的图像。

```
scale = gtk.HScale()
```

HScale 部件被创建。

```
scale.set_range(0, 100)
```

我们设置了这个标尺的低值和高值范围。

```
scale.set_increments(1, 10)
```

set_increments()方法为这个范围设置步长和页尺寸。

```
scale.set_digits(0)
```

我们想让这个标尺的值为整数，所以我们将小数部分的位数设置为 0。

```
if val == 0:
    self.image.set_from_pixbuf(self.mutp)
elif val > 0 and val <= 30:
    self.image.set_from_pixbuf(self.minp)
elif val > 30 and val < 80:
    self.image.set_from_pixbuf(self.medp)
else:
    self.image.set_from_pixbuf(self.maxp)
```

依据获得的值，我们改变 Image 部件中的图像。

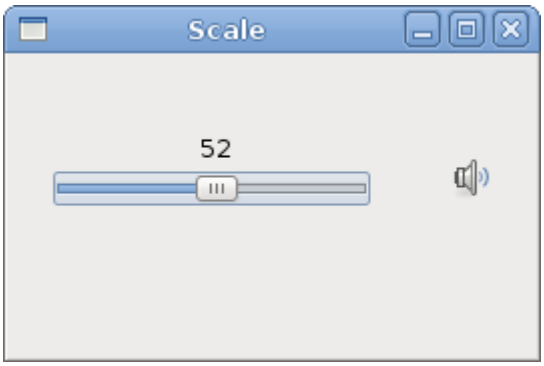


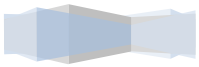
Figure: HScale Widget

ToggleButton

ToggleButton 是一种只有两个状态的按钮，按下了和未按下。你通过点击它来在这两个状态之间转换。在某些情况下，这个功能是相当适用的。

Code: 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()
```

在我们的例子中，我们显示了三个 **ToggleButton** 和一个 **DrawingArea**。我们设置这个区域的背景颜色为黑色。这些 togglebutton 将会分别切换这个颜色值的红绿蓝三个部分。背景颜色将会依据我们按下三个 togglebutton 的状态来变化。


```
self.color = [0, 0, 0]
```

这就是将要被三个 togglebutton 更新的颜色值。

```
red = gtk.ToggleButton("Red")
red.set_size_request(80, 35)
red.connect("clicked", self.onred)
```

ToggleButton 部件被创建，我们设置它的大小为 80*35px，每个切换按钮都有它自己的处理函数方法。

```
self.darea = gtk.DrawingArea()
self.darea.set_size_request(150, 150)
self.darea.modify_bg(gtk.STATE_NORMAL, gtk.gdk.color_parse("black"))
```

DrawingArea 部件就是那种能显示颜色的部件，其颜色是有三个切换按钮的状态给出的值混合而来。最开始，它显示为黑色。

```
if widget.get_active():
    self.color[0] = 65535
else: self.color[0] = 0
```

如果切换按钮被按下，我们对颜色的 R，G 和 B 的值做出相应的改变。

```
self.darea.modify_bg(gtk.STATE_NORMAL, gtk.gdk.Color(self.color[0],
          self.color[1], self.color[2]))
```

我们更新 **DrawingArea** 部件颜色。

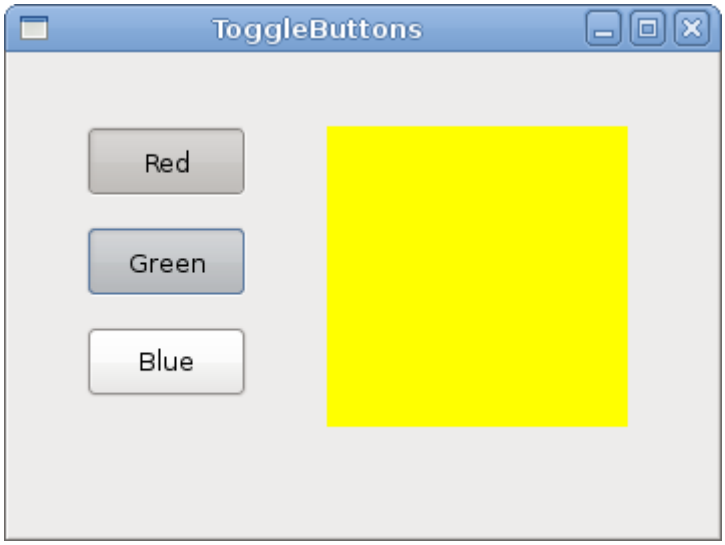


Figure: ToggleButtons Widget

Calendar

我们最后的部件就是日历部件。它被用来做与日期相关的工作。

Code: 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()
```

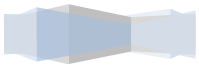
我们有一个**Calendar**部件和一个**Label**部件，从 Calendar 中选择的日期将被显示在 Label 中。

```
calendar = gtk.Calendar()
```

Calendar 部件被创建。

```
(year, month, day) = widget.get_date()
self.label.set_label(str(month) + "/" + str(day) + "/" + str(year))
```

在 **on_day_selected()**方法中，我们获取了当前选择的日期，并且更新到 Label 中。



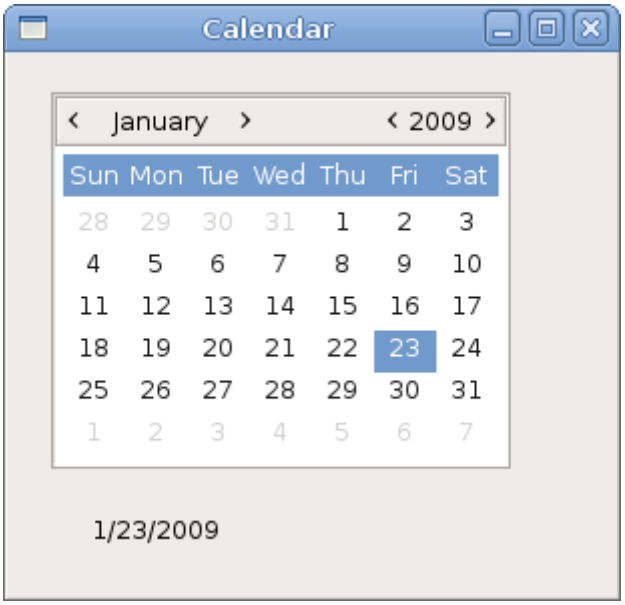


Figure: Calendar

在教程的这章中，我们完成了对 PyGTK 部件的讨论。

08 PyGTK 中的高级部件

在教程的这个部分，我们将介绍一些更高级的部件。

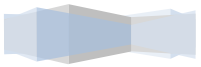
IconView

IconView 是一个在网格中显示一系列的图标的部件。

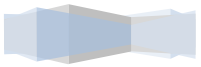
Code: iconview.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# This example demonstrates the IconView widget.
# It shows the contents of the currently selected
# directory on the disk.
#
# author: jan bodnar
# website: zetcode.com
# last edited: February 2009

import gtk
import os

COL_PATH = 0
```

```
COL_PIXBUF = 1
COL_IS_DIRECTORY = 2


class PyApp(gtk.Window):
    def __init__(self):
        super(PyApp, self).__init__()

        self.set_size_request(650, 400)
        self.set_position(gtk.WIN_POS_CENTER)

        self.connect("destroy", gtk.main_quit)
        self.set_title("IconView")

        self.current_directory = '/'

        vbox = gtk.VBox(False, 0);

        toolbar = gtk.Toolbar()
        vbox.pack_start(toolbar, False, False, 0)

        self.upButton = gtk.ToolButton(gtk.STOCK_GO_UP);
        self.upButton.set_is_important(True)
        self.upButton.set_sensitive(False)
        toolbar.insert(self.upButton, -1)

        homeButton = gtk.ToolButton(gtk.STOCK_HOME)
        homeButton.set_is_important(True)
        toolbar.insert(homeButton, -1)

        self.fileIcon = self.get_icon(gtk.STOCK_FILE)
        self.dirIcon = self.get_icon(gtk.STOCK_OPEN)

        sw = gtk.ScrolledWindow()
        sw.set_shadow_type(gtk.SHADOW_ETCHED_IN)
        sw.set_policy(gtk.POLICY_AUTOMATIC, gtk.POLICY_AUTOMATIC)
        vbox.pack_start(sw, True, True, 0)

        self.store = self.create_store()
        self.fill_store()

        iconView = gtk.IconView(self.store)
        iconView.set_selection_mode(gtk.SELECTION_MULTIPLE)

        self.upButton.connect("clicked", self.on_up_clicked)
        homeButton.connect("clicked", self.on_home_clicked)

        iconView.set_text_column(COL_PATH)
        iconView.set_pixbuf_column(COL_PIXBUF)

        iconView.connect("item-activated", self.on_item_activated)
        sw.add(iconView)
        iconView.grab_focus()

        self.add(vbox)
        self.show_all()

    def get_icon(self, name):
```

```
        theme = gtk.icon_theme_get_default()
        return theme.load_icon(name, 48, 0)


    def create_store(self):
        store = gtk.ListStore(str, gtk.gdk.Pixbuf, bool)
        store.set_sort_column_id(COL_PATH, gtk.SORT_ASCENDING)
        return store


    def fill_store(self):
        self.store.clear()

        if self.current_directory == None:
            return

        for fl in os.listdir(self.current_directory):

            if not fl[0] == '.':
                if os.path.isdir(os.path.join(self.current_directory,
fl)):
                    self.store.append([fl, self.dirIcon, True])
                else:
                    self.store.append([fl, self.fileIcon, False])



    def on_home_clicked(self, widget):
        self.current_directory =
os.path.realpath(os.path.expanduser('~'))
        self.fill_store()
        self.upButton.set_sensitive(True)


    def on_item_activated(self, widget, item):

        model = widget.get_model()
        path = model[item][COL_PATH]
        isDir = model[item][COL_IS_DIRECTORY]

        if not isDir:
            return

        self.current_directory = self.current_directory + os.path.sep +
path
        self.fill_store()
        self.upButton.set_sensitive(True)


    def on_up_clicked(self, widget):
        self.current_directory = os.path.dirname(self.current_directory)
        self.fill_store()
        sensitive = True
        if self.current_directory == "/": sensitive = False
        self.upButton.set_sensitive(sensitive)


PyApp()
```

```
gtk.main()
```

这个例子显示了当前选择的目录的图标。它有一个工具栏和两个按钮，up 按钮和 Home 按钮。我们使用它们来操作文件系统。

```
self.current_directory = '/'
```

current_directory 就是被 **IconView** 部件显示的目录。

```
def create_store(self):
    store = gtk.ListStore(str, gtk.gdk.Pixbuf, bool)
    store.set_sort_column_id(COL_PATH, gtk.SORT_ASCENDING)
    return store
```

create_store()方法创建了一个 **ListStore**，它是 IconView 部件中使用的数据模型。它有三个参数，目录名，图标的图像缓存和一个布尔变量，其指示的是我们有一个目录还是一个文件。

```
 f not fl[0] == '.':
    if os.path.isdir(os.path.join(self.current_directory, fl)):
        self.store.append([fl, self.dirIcon, True])
    else:
        self.store.append([fl, self.fileIcon, False])
```

在 **fill_store()**方法中，我们用数据充填了这个 ListStore。这里，我们从当前路径中找出了所有的目录，我们排除了以“.”开头的不可见目录。

```
def on_home_clicked(self, widget):
    self.current_directory = os.path.realpath(os.path.expanduser('~'))
    self.fill_store()
    self.upButton.set_sensitive(True)
```

如果我们点击 Home 按钮，那么 home 目录将会成为当前目录。我们重新充填了这个 ListStore，并且使 up 按钮处于激活状态。

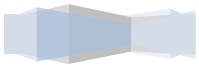
在 **on_item_activated()**方法中，我们和一个事件进行了交互，这个事件就是当我们点击 IconView 部件中的一个图标的时候产生的。

```
model = widget.get_model()
path = model[item][COL_PATH]
isDir = model[item][COL_IS_DIRECTORY]

if not isDir:
    return
```

我们得到了被激活项目的路径，并且我们来判断，它是否是一个目录或者一个文件，如果它是一个文件，我们就直接返回。

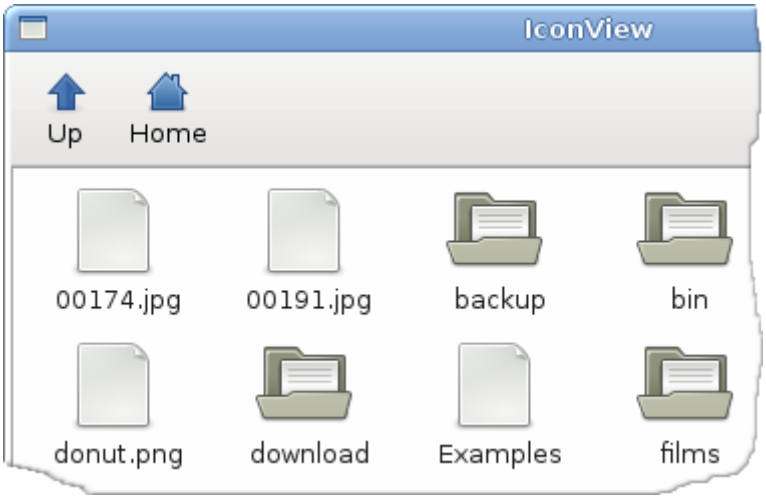
```
self.current_directory = self.current_directory + os.path.sep + path
self.fill_store()
self.upButton.set_sensitive(True)
```


如果他是一个目录，我们就将当前目录替换根目录，并且重新充填 ListStore 和激活 up 按钮。



```
 def on_up_clicked(self, widget):
     self.current_directory = os.path.dirname(self.current_directory)
     self.fill_store()
     sensitive = True
     if self.current_directory == "/": sensitive = False
     self.upButton.set_sensitive(sensitive)
```

如果我们点击 up 按钮，我们就用当前目录的上级目录替换它自己，重新充填 ListStore，



并且如果我们还在文件系统的根目录之下，up 按钮会被激活。

Figure: IconView

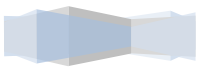
ListView

在下面的例子中，我们使用一个 **TreeView** 部件来展示一个列表视图(list view)，并且 ListStore 又被用来存储数据。

Code: listview.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# This example shows a TreeView widget
# in a list view mode
#
# author: jan bodnar
# website: zetcode.com
# last edited: February 2009


import gtk

actresses = [('jessica alba', 'pomona', '1981'), ('sigourney weaver', 'new
york', '1949'),
    ('angelina jolie', 'los angeles', '1975'), ('natalie portman',
'jerusalem', '1981'),
```

```
    ('rachel weiss', 'london', '1971'), ('scarlett johansson', 'new york',
'1984' )]


class PyApp(gtk.Window):
    def __init__(self):
        super(PyApp, self).__init__()

        self.set_size_request(350, 250)
        self.set_position(gtk.WIN_POS_CENTER)

        self.connect("destroy", gtk.main_quit)
        self.set_title("ListView")

        vbox = gtk.VBox(False, 8)

        sw = gtk.ScrolledWindow()
        sw.set_shadow_type(gtk.SHADOW_ETCHED_IN)
        sw.set_policy(gtk.POLICY_AUTOMATIC, gtk.POLICY_AUTOMATIC)

        vbox.pack_start(sw, True, True, 0)

        store = self.create_model()

        treeView = gtk.TreeView(store)
        treeView.connect("row-activated", self.on_activated)
        treeView.set_rules_hint(True)
        sw.add(treeView)

        self.create_columns(treeView)
        self.statusbar = gtk.Statusbar()

        vbox.pack_start(self.statusbar, False, False, 0)

        self.add(vbox)
        self.show_all()


    def create_model(self):
        store = gtk.ListStore(str, str, str)

        for act in actresses:
            store.append([act[0], act[1], act[2]])

        return store


    def create_columns(self, treeView):

        rendererText = gtk.CellRendererText()
        column = gtk.TreeViewColumn("Name", rendererText, text=0)
        column.set_sort_column_id(0)
        treeView.append_column(column)

        rendererText = gtk.CellRendererText()
        column = gtk.TreeViewColumn("Place", rendererText, text=1)
        column.set_sort_column_id(1)
        treeView.append_column(column)
```

```
        rendererText = gtk.CellRendererText()
        column = gtk.TreeViewColumn("Year", rendererText, text=2)
        column.set_sort_column_id(2)
        treeView.append_column(column)


    def on_activated(self, widget, row, col):

        model = widget.get_model()
        text = model[row][0] + ", " + model[row][1] + ", " + model[row][2]
        self.statusbar.push(0, text)


PyApp()
gtk.main()
```

在我们的例子中，我们在 TreeView 部件中显示了有六个女演员的列表。它们中的每一行显示了名字、出生地和出生年份。

```
def create_model(self):
    store = gtk.ListStore(str, str, str)

    for act in actresses:
        store.append([act[0], act[1], act[2]])

    return store
```

在 **create_model()**方法中，我们创建了 ListStore。这个 ListStore 有三个参数，女演员的名字，出生地和出生年份。这是我们 **TreeView** 部件的数据模型。

```
treeView = gtk.TreeView(store)
treeView.connect("row-activated", self.on_activated)
treeView.set_rules_hint(True)
```

这里我们创建了一个 TreeView 部件，将 ListStore 作为参数。**set_rules_hint()**方法改变 TreeView 部件中没两行的背景颜色。

```
rendererText = gtk.CellRendererText()

column = gtk.TreeViewColumn("Name", rendererText, text=0)
column.set_sort_column_id(0)
treeView.append_column(column)
```

在 **create_columns()**方法中，我们对我们的 TreeView 部件增加了三栏。上面的代码创建了一栏来显示女演员的名字。**CellRendererText** 从三个模型中的第一栏中获取它的文本。（默认文本为 0）

```
def on_activated(self, widget, row, col):

    model = widget.get_model()
    text = model[row][0] + ", " + model[row][1] + ", " + model[row][2]
    self.statusbar.push(0, text)
```

如果我们双击某个项目，我们将在状态栏中显示整行的内容。

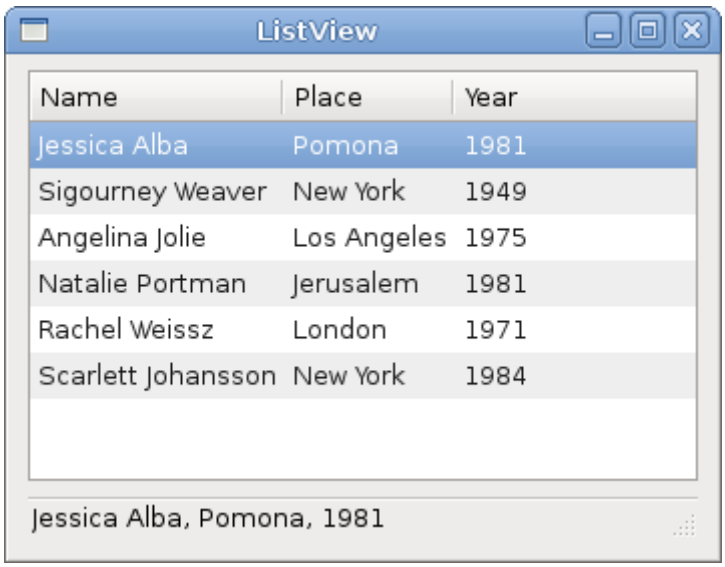


Figure: ListView

Tree

在这章最后的例子中，我们使用 TreeView 部件来展示一个分层结构的数据树。

Code: tree.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# This example shows a TreeView widget
# in a tree view mode
#
# 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_size_request(400, 300)
        self.set_position(gtk.WIN_POS_CENTER)

        self.connect("destroy", gtk.main_quit)
        self.set_title("Tree")

        tree = gtk.TreeView()
```

```
        languages = gtk.TreeViewColumn()
        languages.set_title("Programming languages")

        cell = gtk.CellRendererText()
        languages.pack_start(cell, True)
        languages.add_attribute(cell, "text", 0)

        treestore = gtk.TreeStore(str)

        it = treestore.append(None, ["Scripting languages"])
        treestore.append(it, ["Python"])
        treestore.append(it, ["PHP"])
        treestore.append(it, ["Perl"])
        treestore.append(it, ["Ruby"])

        it = treestore.append(None, ["Compiling languages"])
        treestore.append(it, ["C#"])
        treestore.append(it, ["C++"])
        treestore.append(it, ["C"])
        treestore.append(it, ["Java"])

        tree.append_column(languages)
        tree.set_model(treestore)

        self.add(tree)
        self.show_all()


PyApp()
gtk.main()
```

这次我们使用 **TreeView** 部件来展示分层结构的数据。

```
tree = gtk.TreeView()
```

TreeView 部件被创建。

```
languages = gtk.TreeViewColumn()
languages.set_title("Programming languages")
```

它含有一个名为“Programming languages”的一栏。

```
cell = gtk.CellRendererText()
languages.pack_start(cell, True)
languages.add_attribute(cell, "text", 0)
```

我们在 TreeView 部件中展示了文本数据。

```
treestore = gtk.TreeStore(str)
```

为了存储数据，我们使用 ListStore 对象。

```
it = treestore.append(None, ["Scripting languages"])
treestore.append(it, ["Python"])
treestore.append(it, ["PHP"])
```

我们追加数据到这个树中。**TreeIter** 对象被用来在行中存取数据。

```
tree.append_column(languages)
```

一栏被追加到树中。

```
tree.set_model(treestore)
```

最后我们为这个树部件设置一个数据模型。

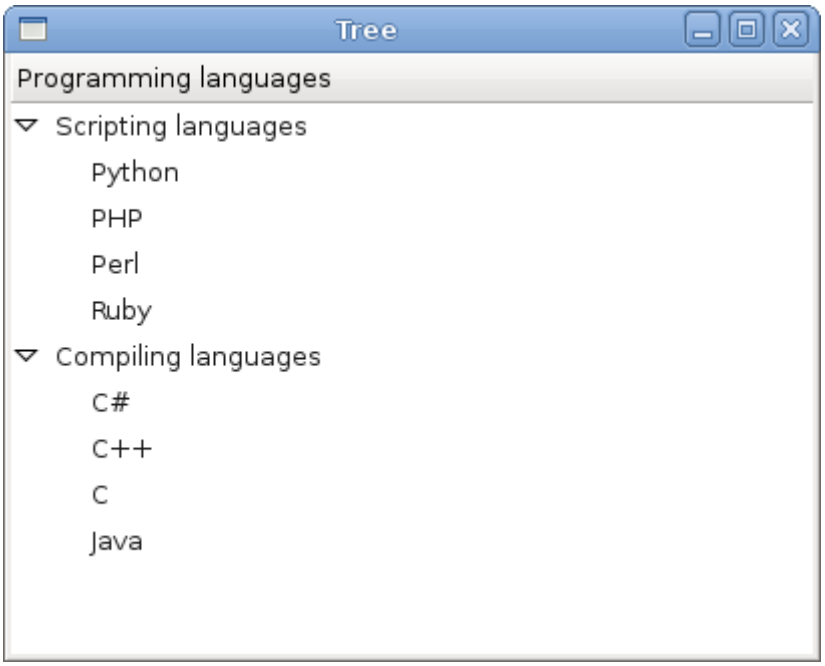


Figure: Tree

在教程的这章中，我们讨论了 PyGTK 中的高级部件。

09 PyGTK 中的对话框

在教程的这个部分，我们将介绍对话框。

对话框窗口或者对话框是大多数现代 GUI 程序一个不可或缺的部分。对框框被定义为一个或更多人之间的对话。在计算机程序中，对话框就是用来与程序对话的窗口。并且反之亦然，对话框是用来输入数据，更改数据，改变应用程序的设置等。对话框是用户和计算机程序之间重要的交流途径。

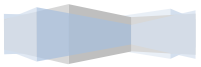
Message dialogs

消息对话框是一种很便利的对话框，它为程序的用户提供各种消息。消息一般是由文本的和图像的数据组成。

Code: message.py

```
#!/usr/bin/python
```

```
# ZetCode PyGTK tutorial
#
# This example shows message
# dialogs
#
# 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_size_request(250, 100)
        self.set_position(gtk.WIN_POS_CENTER)
        self.connect("destroy", gtk.main_quit)
        self.set_title("Message dialogs")


        table = gtk.Table(2, 2, True);

        info = gtk.Button("Information")
        warn = gtk.Button("Warning")
        ques = gtk.Button("Question")
        erro = gtk.Button("Error")


        info.connect("clicked", self.on_info)
        warn.connect("clicked", self.on_warn)
        ques.connect("clicked", self.on_ques)
        erro.connect("clicked", self.on_erro)

        table.attach(info, 0, 1, 0, 1)
        table.attach(warn, 1, 2, 0, 1)
        table.attach(ques, 0, 1, 1, 2)
        table.attach(erro, 1, 2, 1, 2)


        self.add(table)
        self.show_all()

    def on_info(self, widget):
        md = gtk.MessageDialog(self,
            gtk.DIALOG_DESTROY_WITH_PARENT, gtk.MESSAGE_INFO,
            gtk.BUTTONS_CLOSE, "Download completed")
        md.run()
        md.destroy()
```

```
    def on_erro(self, widget):
        md = gtk.MessageDialog(self,
            gtk.DIALOG_DESTROY_WITH_PARENT, gtk.MESSAGE_ERROR,
            gtk.BUTTONS_CLOSE, "Error loading file")
```

```
        md.run()
        md.destroy()


    def on_ques(self, widget):
        md = gtk.MessageDialog(self,
            gtk.DIALOG_DESTROY_WITH_PARENT, gtk.MESSAGE_QUESTION,
            gtk.BUTTONS_CLOSE, "Are you sure to quit?")
        md.run()
        md.destroy()


    def on_warn(self, widget):
        md = gtk.MessageDialog(self,
            gtk.DIALOG_DESTROY_WITH_PARENT, gtk.MESSAGE_WARNING,
            gtk.BUTTONS_CLOSE, "Unallowed operation")
        md.run()
        md.destroy()


PyApp()
gtk.main()
```

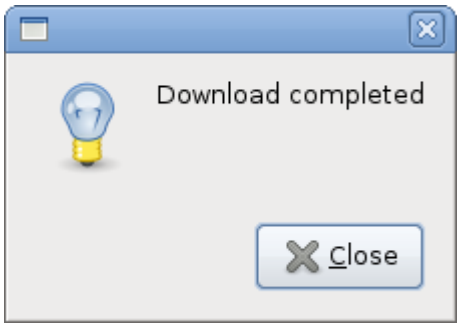
在我们的例子中，我们展示了四种消息对话框，信息、警告、问题和错误消息对话框。

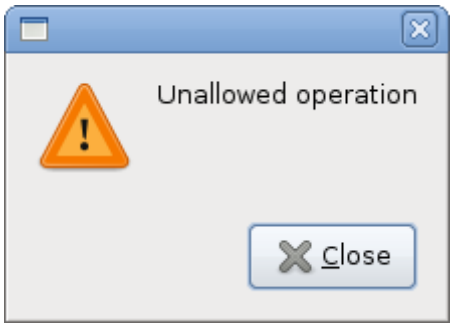
```
info = gtk.Button("Information")
warn = gtk.Button("Warning")
ques = gtk.Button("Question")
erro = gtk.Button("Error")
```

我们有四个按钮，每个按钮都会显示一种不同类型的消息对话框。

```
md = gtk.MessageDialog(self,
    gtk.DIALOG_DESTROY_WITH_PARENT, gtk.MESSAGE_INFO,
        gtk.BUTTONS_CLOSE, "Download completed")
md.run()
md.destroy()
```

如果我们点击 info 按钮，信息消息对话框将会显示。**MESSAGE_INFO** 会被指定为这种对话框的类型，显示在对话框中的按钮将会被指定为 **BUTTON_CLOSE** 类型。最后一个参数是将要显示的消息内容。而对话框将用 **run()**方法来显示，程序员还必须调用 **destroy()**或者 **hide()**方法。





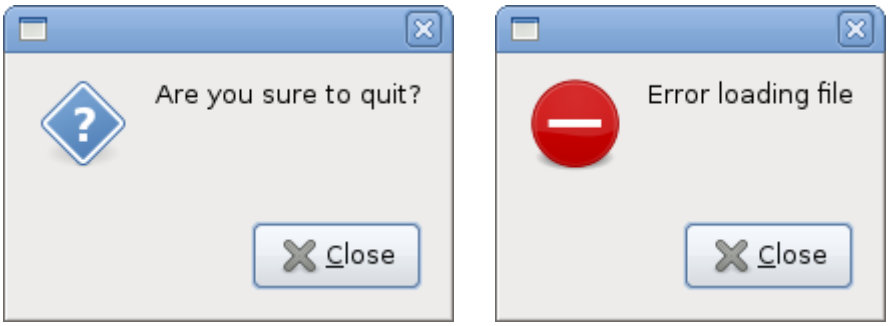


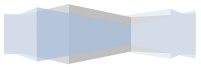
Figure: Meaage

AboutDialog

关于对话框是显示关于程序信息的对话框。在关于对话框中能够显示 logo，程序的名称，版本，版权说明，网站或者许可证信息。它还有可能给出作者、文档编写者、翻译者和艺术设计者相关的赞许信息。

Code: aboutdialog.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# This example demonstrates the
# AboutDialog dialog
#
# 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_size_request(300, 150)
        self.set_position(gtk.WIN_POS_CENTER)
        self.connect("destroy", gtk.main_quit)
        self.set_title("About battery")


        button = gtk.Button("About")
        button.set_size_request(80, 30)
        button.connect("clicked", self.on_clicked)

        fix = gtk.Fixed()
        fix.put(button, 20, 20)

        self.add(fix)
```

```
        self.show_all()

    def on_clicked(self, widget):
        about = gtk.AboutDialog()
        about.set_program_name("Battery")
        about.set_version("0.1")
        about.set_copyright("(c) Jan Bodnar")
        about.set_comments("Battery is a simple tool for battery checking")
        about.set_website("http://www.zetcode.com")
        about.set_logo(gtk.gdk.pixbuf_new_from_file("battery.png"))
        about.run()
        about.destroy()

PyApp()
gtk.main()
```

这个代码示例使用了一个包含其一些特征的关于对话框。

```
about = gtk.AboutDialog()
```

我们创建了一个关于对话框。

```
about = gtk.AboutDialog()
about.set_program_name("Battery")
about.set_version("0.1")
about.set_copyright("(c) Jan Bodnar")
```

我们指定了名称、版本号和版权信息。

```
about.set_logo(gtk.gdk.pixbuf_new_from_file("battery.png"))
```

这一行创建了一个 logo。

Figure: AboutDialog

FontSelectionDialog

字体选择对话框就是一种为了选择字体的对话框。在程序中，它有很典型的应用，如做一些

文本编辑，或者文本格式化。

Code: fontdialog.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# This example works with the
# FontSelection dialog
#
# author: jan bodnar
# website: zetcode.com
# last edited: February 2009


import gtk
import pango


class PyApp(gtk.Window):
    def __init__(self):
        gtk.Window.__init__(self)
        self.set_size_request(300, 150)
        self.set_position(gtk.WIN_POS_CENTER)
        self.connect("destroy", gtk.main_quit)
        self.set_title("Font Selection Dialog")


        self.label = gtk.Label("The only victory over love is flight.")
        button = gtk.Button("Select font")
        button.connect("clicked", self.on_clicked)

        fix = gtk.Fixed()
        fix.put(button, 100, 30)
        fix.put(self.label, 30, 90)
        self.add(fix)

        self.show_all()

    def on_clicked(self, widget):
        fdia = gtk.FontSelectionDialog("Select font name")
        response = fdia.run()

        if response == gtk.RESPONSE_OK:
            font_desc = pango.FontDescription(fdia.get_font_name())
            if font_desc:
                self.label.modify_font(font_desc)

        fdia.destroy()


PyApp()
gtk.main()
```


在这个代码示例中，我们有一个按钮和一个标签。我们通过点击按钮来显示一个**FontSelectionDialog**。

```
fdia = gtk.FontSelectionDialog("Select font name")
```

我们创建了一个 FontSelectionDialog。

```
if response == gtk.RESPONSE_OK:
    font_desc = pango.FontDescription(fdia.get_font_name())
    if font_desc:
        self.label.modify_font(font_desc)
```

如果我们点击 OK 按钮 ,标签部件的内容的字体将变成我们在字体选择对话框中所选择的字体。

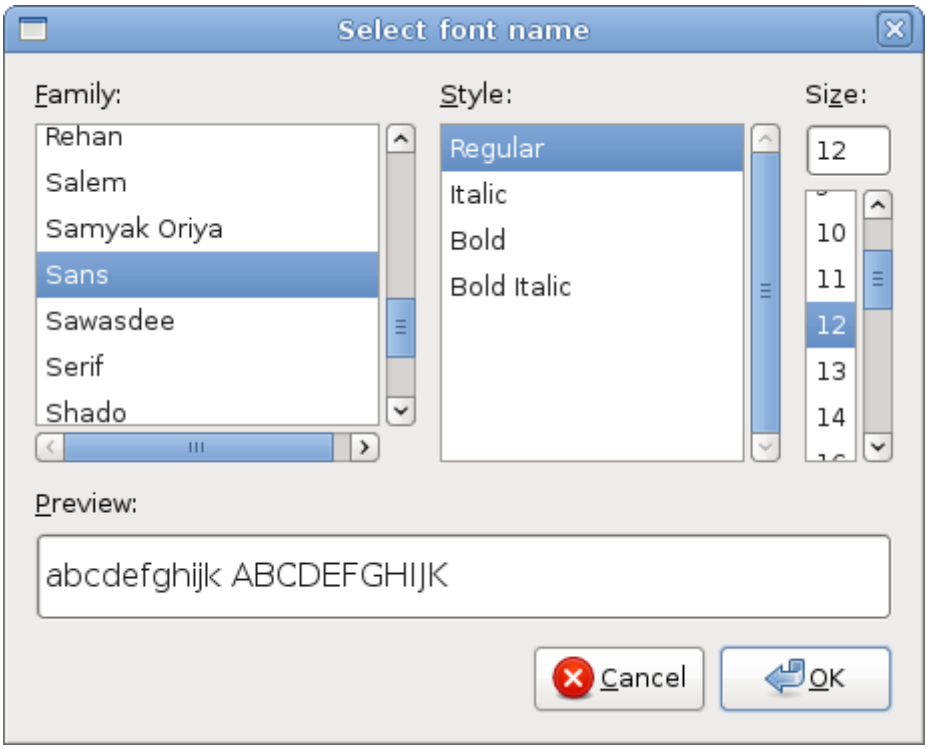


Figure: FontSelectionDialog

ColorSelectionDialog

颜色选择对话框是一种用来选择颜色的对话框。

Code: colordialog.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# This example works with the
# ColorSelection dialog
#
# 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_size_request(300, 150)
        self.set_position(gtk.WIN_POS_CENTER)
        self.connect("destroy", gtk.main_quit)
        self.set_title("Color Selection Dialog")


        self.label = gtk.Label("The only victory over love is flight.")
        button = gtk.Button("Select color")
        button.connect("clicked", self.on_clicked)

        fix = gtk.Fixed()
        fix.put(button, 100, 30)
        fix.put(self.label, 30, 90)
        self.add(fix)

        self.show_all()

    def on_clicked(self, widget):
        cdia = gtk.ColorSelectionDialog("Select color")
        response = cdia.run()

        if response == gtk.RESPONSE_OK:
            colorsel = cdia.colorsel
            color = colorsel.get_current_color()
            self.label.modify_fg(gtk.STATE_NORMAL, color)

        cdia.destroy()


PyApp()
gtk.main()
```

这个列子相对于前一个非常简单，这次我们改变标签的颜色。

```
cdia = gtk.ColorSelectionDialog("Select color")
```

我们创建了一个 **ColorSelectionDialog**。

```
if response == gtk.RESPONSE_OK:
    colorsel = cdia.colorsel
    color = colorsel.get_current_color()
    self.label.modify_fg(gtk.STATE_NORMAL, color)
```

如果用花点击 OK，我们就得到了所选择的颜色，并且来改变标签的颜色。

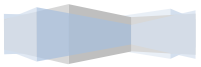


Figure: ColorSelectionDialog

在教程的这章中，我们用 PyGTK 内置的对话框来工作。

10 Pango

在教程的这个部分，我们将探索 Pango 库。

Pango 是一个免费的开源的计算处理库，它用来在较高的层次处理国际化的文本。使用不同类型的字体后端，并且提供跨平台的支持。（来自 Wikipedia）

Pango 提供了高级的字体和文本处理函数，它被用来为 GDK 和 GTK 服务。

Simple Example

在我们的第一个例子中，我们将展示怎样去改变我们标签部件的字体。

Code: quotes.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# This example shows how to modify
# the font of a label
#
# author: jan bodnar
# website: zetcode.com
# last edited: February 2009
```

```
import gtk
import pango

quotes = """Excess of joy is harder to bear than any amount of sorrow.
The more one judges, the less one loves.
There is no such thing as a great talent without great will power.
"""


class PyApp(gtk.Window):
    def __init__(self):
        super(PyApp, self).__init__()

        self.connect("destroy", gtk.main_quit)
        self.set_title("Quotes")

        label = gtk.Label(quotes)
        gtk.gdk.beep()

        fontdesc = pango.FontDescription("Purisa 10")
        label.modify_font(fontdesc)

        fix = gtk.Fixed()

        fix.put(label, 5, 5)

        self.add(fix)
        self.set_position(gtk.WIN_POS_CENTER)
        self.show_all()

PyApp()
gtk.main()
```

在我们上面的代码示例中，我们有一个包含三个引用的标签部件。我们改变它的字体为 Purisa 10。

```
quotes = """Excess of joy is harder to bear than any amount of sorrow.
The more one judges, the less one loves.
There is no such thing as a great talent without great will power.
"""
```

这是将会展示在标签中的文本。

```
fontdesc = pango.FontDescription("Purisa 10")
```

FontDescription 方法被用来指定字体的特征。

```
label.modify_font(fontdesc)
```

我们改变标签部件的字体为 Purisa 10。

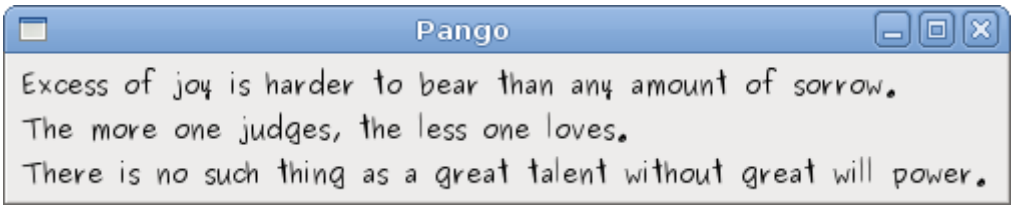


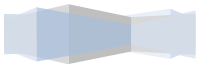
Figure: Quotations

System Fonts

下面的代码例子将在一个 TreeView 部件中展示所有可用的字体。

Code: systemfonts.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# This example lists all available
# fonts on a system in a TreeView widget
#
# author: jan bodnar
# website: zetcode.com
# last edited: February 2009

import gtk
import pango


class PyApp(gtk.Window):
    def __init__(self):
        super(PyApp, self).__init__()

        self.set_size_request(350, 250)
        self.set_border_width(8)
        self.connect("destroy", gtk.main_quit)
        self.set_title("System fonts")

        sw = gtk.ScrolledWindow()
        sw.set_shadow_type(gtk.SHADOW_ETCHED_IN)
        sw.set_policy(gtk.POLICY_AUTOMATIC, gtk.POLICY_AUTOMATIC)

        context = self.create_pango_context()
        self.fam = context.list_families()

        store = self.create_model()

        treeView = gtk.TreeView(store)
        treeView.set_rules_hint(True)
        sw.add(treeView)

        self.create_column(treeView)

        self.add(sw)

        self.set_position(gtk.WIN_POS_CENTER)
        self.show_all()


    def create_column(self, treeView):
        rendererText = gtk.CellRendererText()
        column = gtk.TreeViewColumn("FontName", rendererText, text=0)
        column.set_sort_column_id(0)
```

```
        treeView.append_column(column)

    def create_model(self):
        store = gtk.ListStore(str)

        for ff in self.fam:
            store.append([ff.get_name()])

        return store


PyApp()
gtk.main()
```

在代码示例中展示了系统中所有可用的字体。

```
context = self.create_pango_context()
```

这行代码创建了一个 Pango 的 Context 对象。它包含了对文本进行修饰过程的全局信息。

```
self.fam = context.list_families()
```

从这个 Context 对象，我们获取了所有可用的字体家族。

```
for ff in self.fam:
    store.append([ff.get_name()])
```

在 TreeView 部件模型创建的过程中，我们从字体家族的数组中获取所有的字体名称，并且将它们放到 ListStore 中。

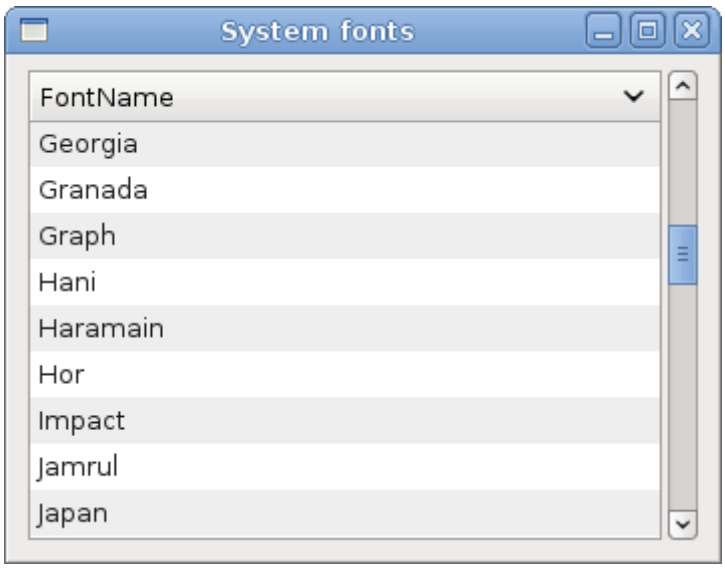


Figure: SystemFonts

Unicode

Pango 被用作处理国际化的文本。

Code: unicode.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

# ZetCode PyGTK tutorial
#
# This example displays text
# in azbuka
#
# author: jan bodnar
# website: zetcode.com
# last edited: February 2009

import gtk
import pango


obj = unicode(u'''Ф ё д о р  М и х а й л о в и ч  Д о с т о е в с ки й
р о д и л с я 30 о кт яб р я (11 н о яб р я)
1821 г о д а  в  М о с кв е . Б ы л  в т о р ы м  и з  7 д е т е й .
О т е ц , М и х а и л  А н д р е е в и ч ,
р а б о т а л  в г о с п и т а л е  д л я  б е д н ы х . М а т ь , М а
р и я  Ф ё д о р о в н а
(в  д е в и ч е с т в е  Н е ч а е в а ), п р о и с х о д и л а  и з
ку п е ч е с ко г о  р о д а .''')


class PyApp(gtk.Window):
    def __init__(self):
        super(PyApp, self).__init__()

        self.connect("destroy", gtk.main_quit)
        self.set_title("Unicode")

        label = gtk.Label(obj.encode('utf-8'))

        fontdesc = pango.FontDescription("Purisa 10")
        label.modify_font(fontdesc)

        fix = gtk.Fixed()

        fix.put(label, 5, 5)

        self.add(fix)
        self.set_position(gtk.WIN_POS_CENTER)
        self.show_all()

PyApp()
gtk.main()
```

我们用 azbuka 语言展示了一些文本。【译者注 :azbuka 不知道什么意思 ,有可能拼写错误 , 此处应该代表一种语言文字的名称】


```
# -*- coding: utf-8 -*-
```

为了在源代码中能够直接用国际化的文本进行工作 ,我们必须提供这个神奇的注释。注意了 ,

它必须被放置在第一或者第二行。

```
obj = unicode(u'''Ф ё д о р  М и х а й л о в и ч  Д о с т о е в с ки й
р о д и л с я 30 о кт  яб р я (11 н о яб р я)
1821 г о д а  в  М о с кв е . Б ы л  в т о р ым  и з  7 д е т е й .
О т е ц , М и х а и л  А н д р е е в и ч ,
р а б о т а л  в г о с п и т а л е  д л я  б е д н ых . М а т ь , М а
р и я  Ф ё д о р о в н а
(в  д е в и ч е с т в е  Н е ч а е в а ), п р о и с х о д и л а  и з
ку п е ч е с ко г о  р о д а .''')
```

这是用 azbuka 语言拼写的文本。

```
Label label = new Label(text);
```

我们将编码过的文本放进标签中。

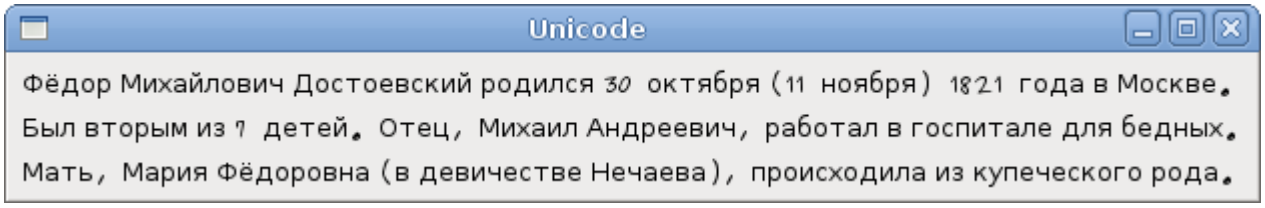


Figure: Unicode

Attributes

Pango 属性就是一种应用于部分文本的属性特征。

Code: attributes.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# In this program we work with
# pango attributes
#
# author: jan bodnar
# website: zetcode.com
# last edited: February 2009

import gtk
import pango

text = "Valour fate kinship darkness"


class PyApp(gtk.Window):
    def __init__(self):
        super(PyApp, self).__init__()

        self.connect("destroy", gtk.main_quit)
        self.set_title("Attributes")
```

```
        label = gtk.Label(text)

        attr = pango.AttrList()

        fg_color = pango.AttrForeground(65535, 0, 0, 0, 6)
        underline = pango.AttrUnderline(pango.UNDERLINE_DOUBLE, 7, 11)
        bg_color = pango.AttrBackground(40000, 40000, 40000, 12, 19)
        strike = pango.AttrStrikethrough(True, 20, 29)
        size = pango.AttrSize(30000, 0, -1)

        attr.insert(fg_color)
        attr.insert(underline)
        attr.insert(bg_color)
        attr.insert(size)
        attr.insert(strike)

        label.set_attributes(attr)

        fix = gtk.Fixed()

        fix.put(label, 5, 5)

        self.add(fix)
        self.set_position(gtk.WIN_POS_CENTER)
        self.show_all()

PyApp()
gtk.main()
```

在上面的代码示例中，我们展示了应用于文本的四种不同的属性特征。

```
attr = pango.AttrList()
```

Pango 的属性列表就是一个保存属性的对象。

```
fg_color = pango.AttrForeground(65535, 0, 0, 0, 6)
```

这里我们创建了一种属性，它用来将文本的颜色修饰为红色。前面的三个参数是一种颜色的 R、G、B 值。后面两个参数是我们要应用这个属性的文本的开头和结尾索引值。

```
label.set_attributes(attr)
```

我们为标签设置这个属性列表。

Figure: Pango attributes

在教程的这章中，我们用 Pango 库来工作。

11 Pango II

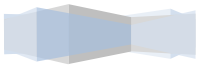
在教程的这个部分，我们将继续探索 Pango 库。

Animated text

在下面的例子中，我们将在一个窗口中显示动态的文本。

Code: animation.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# This example shows animated text
#
# author: jan bodnar
# website: zetcode.com
# last edited: February 2009

import gtk
import glib
import pango
import math


class PyApp(gtk.Window):
    def __init__(self):
        super(PyApp, self).__init__()

        self.connect("destroy", gtk.main_quit)
        glib.timeout_add(160, self.on_timer)

        self.count = 1

        self.set_border_width(10)
        self.set_title("ZetCode")

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

        fontdesc = pango.FontDescription("Serif Bold 30")
        self.label.modify_font(fontdesc)

        vbox = gtk.VBox(False, 0)
        vbox.add(self.label)

        self.add(vbox)
        self.set_size_request(300, 250)
        self.set_position(gtk.WIN_POS_CENTER)
        self.show_all()

    def on_timer(self):
        attr = pango.AttrList()
        self.count = self.count + 1
```

```
        for i in range(7):
            r = pango.AttrRise(int(math.sin(self.count+i)*20)*pango.SCALE,
i, i+1)
            attr.insert(r)

        self.label.set_attributes(attr)
        return True


PyApp()
gtk.main()
```

在上面的代码示例中，我们在标签部件中有一行文本。通过不断的更改它的 Pango 属性，这个文本就变成了动态的了。

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

fontdesc = pango.FontDescription("Serif Bold 30")
self.label.modify_font(fontdesc)
```

我们创建了一个标签，并且改变它的字体。为了更好的视觉效果，我们选择了较大的文本。

```
vbox = gtk.VBox(False, 0)
vbox.add(self.label)
```

我们将标签放置到垂直箱子中，这将使标签位于窗口的中间。

这个动画在 **on_timer()**方法中被执行。

```
for i in range(7):
    r = pango.AttrRise(int(math.sin(self.count+i)*20)*pango.SCALE, i, i+1)
    attr.insert(r)
```

在我们的文本中 ,我们有七个字母。我们为每个字母周期性地更改 Pango 的 AttrRise 属性。升降是基于三角正弦函数。文本的运动是沿着由正弦函数所绘成的笛卡尔曲线的。

当然也要注意到 pango.SCALE 常量。Pango 有它自己的内部单元。它们不同于那些通常被 widgets 用来绘制图形或文本的单元。我们必须用这个常数来乘以我们的数字。

Figure: Animated Text

Using markup language

我们可以通过使用内置的标记语言来改变文本的属性。

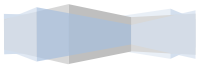
Code: markup.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# This example uses markup language
# to change attributes of the text
#
# author: jan bodnar
# website: zetcode.com
# last edited: February 2009

import gtk
import pango

quote = "<span foreground='blue' size='19000'>The only victory over love is
flight</span>"


class PyApp(gtk.Window):
    def __init__(self):
        super(PyApp, self).__init__()

        self.set_title("Markup")
        self.set_border_width(5)
        self.connect("destroy", gtk.main_quit)

        label = gtk.Label()
        label.set_markup(quote)

        vbox = gtk.VBox(False, 0)
        vbox.add(label)

        self.add(vbox)
        self.set_position(gtk.WIN_POS_CENTER)
        self.show_all()

PyApp()
gtk.main()
```

在代码示例中，我们有一个标签，我们通过标记语言来改变它的文本属性。

```
quote = "<span foreground='blue' size='19000'>The only victory over love is
flight</span>"
```

这是包含有标记语言的文本。



```
label = gtk.Label()
label.set_markup(quote)
```

我们创建了一个标签部件，并且为它设置了一个含有标记的文本。

Figure: Using Markup

Pango layout

Pango 的布局就是一个对象，它用来呈现一个段落包含属性的文本。

Code: layout.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# This example shows pango Layout
# in action
#
# author: jan bodnar
# website: zetcode.com
# last edited: February 2009


import gtk
import pango


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 Area(gtk.DrawingArea):
    def __init__(self):
        super(Area, self).__init__()
        self.modify_bg(gtk.STATE_NORMAL,   gtk.gdk.Color(16400,   16400,
16440))
```

```
        self.connect("expose_event", self.expose)

    def expose(self, widget, event):

        gc = self.get_style().fg_gc[gtk.STATE_NORMAL]
        font_desc = pango.FontDescription('Sans 10')

        layout = self.create_pango_layout(lyrics)
        width, height = self.get_size_request()

        attr = pango.AttrList()

        fg_color = pango.AttrForeground(60535, 60535, 60535, 0, -1)
        attr.insert(fg_color)

        layout.set_width(pango.SCALE * self.allocation.width)
        layout.set_spacing(pango.SCALE * 3)
        layout.set_alignment(pango.ALIGN_CENTER)
        layout.set_font_description(font_desc)
        layout.set_attributes(attr)

        self.window.draw_layout(gc, 0, 5, layout)



class PyApp(gtk.Window):
    def __init__(self):
        super(PyApp, self).__init__()

        self.connect("destroy", gtk.main_quit)
        self.set_title("You know I'm no Good")

        self.add(Area())
        self.set_size_request(300, 300)
        self.set_position(gtk.WIN_POS_CENTER)
        self.show_all()


PyApp()
gtk.main()
```

在之前的例子中，我们在现存的部件中改变文本。现在，我们将用 **pango layout** 在 **DrawingArea** 部件上绘制文本。我们将用到 **GDK** 的绘制工具来绘制。

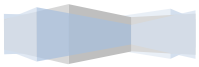
```
gc = self.get_style().fg_gc[gtk.STATE_NORMAL]
```

我们得到了 DrawingArea 部件的图形环境。

```
layout = self.create_pango_layout(lyrics)
```

这里创建了 pango layout 对象。

```
layout.set_width(pango.SCALE * self.allocation.width)
layout.set_spacing(pango.SCALE * 3)
layout.set_alignment(pango.ALIGN_CENTER)
layout.set_font_description(font_desc)
layout.set_attributes(attr)
```

我们更改布局的宽度、间隔、对齐、字体并且设置文本的属性。

```
self.window.draw_layout(gc, 0, 5, layout)
```

Layout 被绘进窗口中。

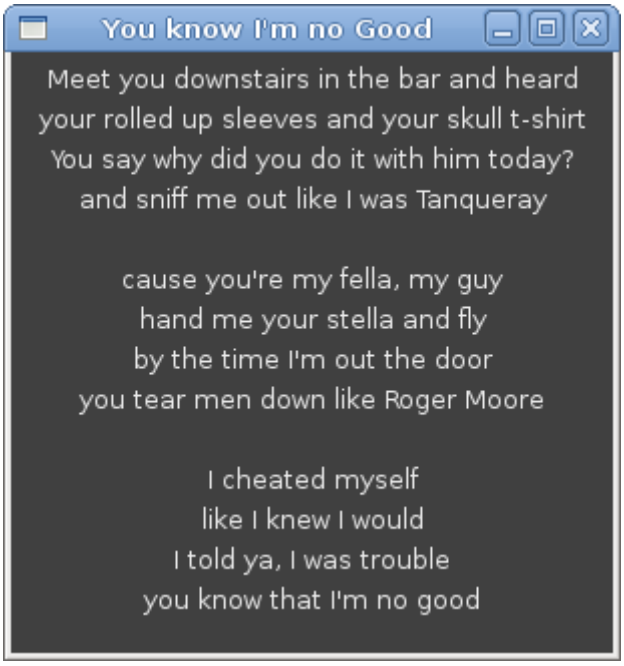


Figure: Layout

在教程的这章中，我们用 Pango 库做了更深入的工作。

12 用 Cairo 库绘图

在教程的这个部分，我们将用 Cairo 做一些绘图的事情。

Cairo 是一种用来创建 2D 矢量图形的库。我们可以用它来绘制我们的部件，图标或者各种效果或动画。

Simple drawing

Stroke 操作是绘制模型的边界，而 fill 操作是填充模型的内部。下面我们将展示这两个操作。

Code: simpledrawing.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# This code example draws a circle
# using the cairo library
```

```
#
# author: jan bodnar
# website: zetcode.com
# last edited: February 2009


import gtk
import math

class PyApp(gtk.Window):

    def __init__(self):
        super(PyApp, self).__init__()

        self.set_title("Simple drawing")
        self.resize(230, 150)
        self.set_position(gtk.WIN_POS_CENTER)

        self.connect("destroy", gtk.main_quit)

        darea = gtk.DrawingArea()
        darea.connect("expose-event", self.expose)
        self.add(darea)

        self.show_all()

    def expose(self, widget, event):

        cr = widget.window.cairo_create()

        cr.set_line_width(9)
        cr.set_source_rgb(0.7, 0.2, 0.0)

        w = self.allocation.width
        h = self.allocation.height

        cr.translate(w/2, h/2)
        cr.arc(0, 0, 50, 0, 2*math.pi)
        cr.stroke_preserve()

        cr.set_source_rgb(0.3, 0.4, 0.6)
        cr.fill()


PyApp()
gtk.main()
```

在我们的例子中，我们将画一个圆，并且它将有立体的颜色。

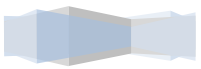
```
darea = gtk.DrawingArea()
```

我们将要在 DrawingArea 部件上做绘图操作。

```
darea.connect("expose-event", self.expose)
```

我们在一个方法中做所有的绘图工作，这个方法是 expose-event 信号的处理函数。

```
cr = widget.window.cairo_create()
```

我们从绘图区域的 gdk.Window 创建了 cairo context 对象。这个 context 是一种用来在所有可绘图对象上绘画的对象。

```
cr.set_line_width(9)
```

我们设置线的宽度为 9 个像素。

```
cr.set_source_rgb(0.7, 0.2, 0.0)
```

我们将颜色设置为暗红色。

```
w = self.allocation.width
h = self.allocation.height

cr.translate(w/2, h/2)
```

我们得到绘图区域的宽度和高度。我们移动原点到窗口的中心。

```
cr.arc(0, 0, 50, 0, 2*math.pi)
cr.stroke_preserve()
```

stroke_preserve()方法是根据当前的线宽度(line width)、线连接(line join)、线终结(line cap)和混合设定(dash settings)来绘出(stroke)当前的路径。【译者注：此处翻译可能有误，请自行理解英文意义】。不同于 **stroke()**方法，它在 cairo context 内保持线的路径。

```
cr.set_source_rgb(0.3, 0.4, 0.6)
cr.fill()
```

这里用一些蓝色来填充圆的内部。

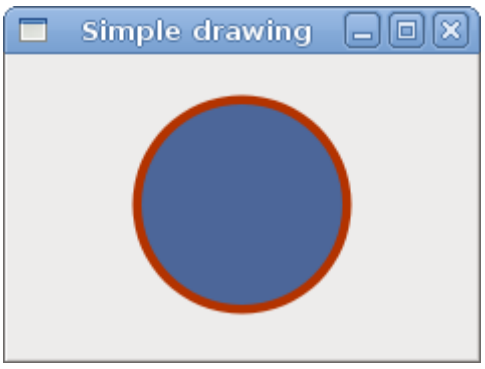


Figure: Simple Drawing

Basic shapes

下面的例子将绘制一些基本图形（Basic shapes）到窗口上。

Code: basicshapes.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
```

```
#
# This code example draws basic shapes
# with the cairo library
#
# author: jan bodnar
# website: zetcode.com
# last edited: February 2009

import gtk
import math

class PyApp(gtk.Window):

    def __init__(self):
        super(PyApp, self).__init__()

        self.set_title("Basic shapes")
        self.set_size_request(390, 240)
        self.set_position(gtk.WIN_POS_CENTER)

        self.connect("destroy", gtk.main_quit)

        darea = gtk.DrawingArea()
        darea.connect("expose-event", self.expose)
        self.add(darea)

        self.show_all()

    def expose(self, widget, event):

        cr = widget.window.cairo_create()
        cr.set_source_rgb(0.6, 0.6, 0.6)

        cr.rectangle(20, 20, 120, 80)
        cr.rectangle(180, 20, 80, 80)
        cr.fill()

        cr.arc(330, 60, 40, 0, 2*math.pi)
        cr.fill()

        cr.arc(90, 160, 40, math.pi/4, math.pi)
        cr.fill()

        cr.translate(220, 180)
        cr.scale(1, 0.7)
        cr.arc(0, 0, 50, 0, 2*math.pi)
        cr.fill()


PyApp()
gtk.main()
```

在这个例子中，我们创建了一个长方形、一个正方形、一个圆形、一个弧形和一个椭圆形。


```
cr.rectangle(20, 20, 120, 80)
cr.rectangle(180, 20, 80, 80)
cr.fill()
```

这几行绘制了一个长方形和一个正方形。

```
cr.arc(330, 60, 40, 0, 2*math.pi)
cr.fill()
```

在这里，**arc()**方法绘制出了一个整圆。

```
cr.scale(1, 0.7)
cr.arc(0, 0, 50, 0, 2*math.pi)
cr.fill()
```

如果你想画一个椭圆的，你首先得做一些缩放。这里 **scale()**方法将 Y 轴的距离缩小了。

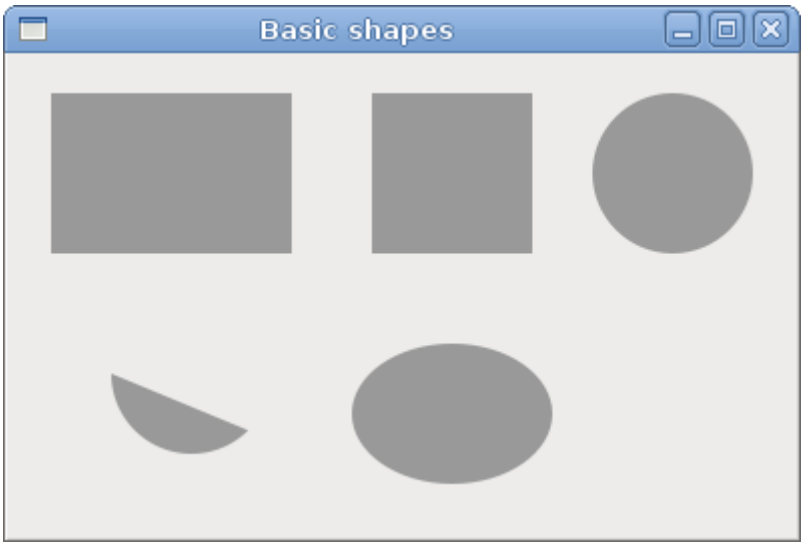


Figure: Basic Shapes

Colors

Color 就是一个描述红、绿、蓝(RGB)的强度值组合的对象。Cairo 有效的 RGB 值是在 0 到 1 之间变化。

Code: colors.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# This program shows how to work
# with colors in cairo
#
# 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("Colors")
        self.resize(360, 100)
        self.set_position(gtk.WIN_POS_CENTER)

        self.connect("destroy", gtk.main_quit)

        darea = gtk.DrawingArea()
        darea.connect("expose-event", self.expose)
        self.add(darea)

        self.show_all()

    def expose(self, widget, event):

        cr = widget.window.cairo_create()

        cr.set_source_rgb(0.2, 0.23, 0.9)
        cr.rectangle(10, 15, 90, 60)
        cr.fill()

        cr.set_source_rgb(0.9, 0.1, 0.1)
        cr.rectangle(130, 15, 90, 60)
        cr.fill()

        cr.set_source_rgb(0.4, 0.9, 0.4)
        cr.rectangle(250, 15, 90, 60)
        cr.fill()

PyApp()
gtk.main()
```

我们用三个不同的颜色绘制了三个长方形。

```
cr.set_source_rgb(0.2, 0.23, 0.9)
```

set_source_rgb()方法是为 cairo context 对象设置一个颜色。方法的三个参数是颜色的强度值。

```
cr.rectangle(10, 15, 90, 60)
cr.fill()
```

我们创建了一个长方形的形状，并且用先前指定的颜色来填充它。

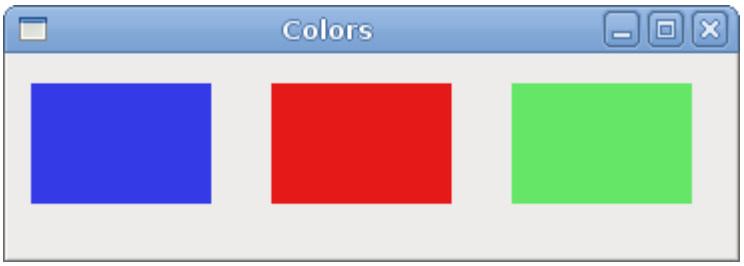


Figure: Colors

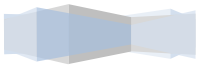
Transparent rectangles

透明度是能够通过肉眼看到的一种特性。最简单的理解透明度的方法是去想象一下一块玻璃或者水。在技术上，光线能够穿透玻璃，这样我们就能够看到玻璃后面的物体了。

在我们的计算机图形中，我们可以使用 alpha 混合来完成透明的效果。而 alpha 混合是一种将图像和其背景结合起来并且创建出局部透明的外观的处理过程。混合过程就是用的 alpha 通道。（引用 wikipedia.com，answers.com）

Code: transparentrectangles.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# This program shows transparent
# rectangles using cairo
#
# 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("Transparent rectangles")
        self.resize(590, 90)
        self.set_position(gtk.WIN_POS_CENTER)

        self.connect("destroy", gtk.main_quit)

        darea = gtk.DrawingArea()
        darea.connect("expose-event", self.expose)
        self.add(darea)

        self.show_all()

    def expose(self, widget, event):

        cr = widget.window.cairo_create()

        for i in range(1, 11):
            cr.set_source_rgba(0, 0, 1, i*0.1)
            cr.rectangle(50*i, 20, 40, 40)
            cr.fill()
```

```
PyApp()
gtk.main()
```

在例子中，我们用不同透明度水平值来画 10 个矩形。

```
cr.set_source_rgba(0, 0, 1, i*0.1)
```

set_source_rgba()方法的最后一个参数就是 alpha 透明度。

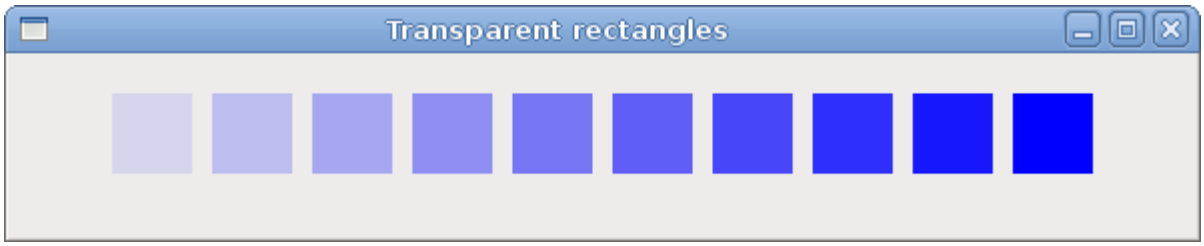


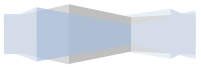
Figure: Transparent Rectangles

Soulmate

下面的例子，我们将在窗口上画一些文本。

Code: soulmate.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# This program draws text
# using cairo
#
# author: jan bodnar
# website: zetcode.com
# last edited: February 2009


import gtk
import cairo

class PyApp(gtk.Window):

    def __init__(self):
        super(PyApp, self).__init__()

        self.set_title("Soulmate")
        self.set_size_request(370, 240)
        self.set_position(gtk.WIN_POS_CENTER)

        self.connect("destroy", gtk.main_quit)

        darea = gtk.DrawingArea()
        darea.connect("expose-event", self.expose)
        self.add(darea)
```

```
        self.show_all()

    def expose(self, widget, event):

        cr = widget.window.cairo_create()

        cr.set_source_rgb(0.1, 0.1, 0.1)

        cr.select_font_face("Purisa", cairo.FONT_SLANT_NORMAL,
            cairo.FONT_WEIGHT_NORMAL)
        cr.set_font_size(13)

        cr.move_to(20, 30)
        cr.show_text("Most relationships seem so transitory")
        cr.move_to(20, 60)
        cr.show_text("They're all good but not the permanent one")
        cr.move_to(20, 120)
        cr.show_text("Who doesn't long for someone to hold")
        cr.move_to(20, 150)
        cr.show_text("Who knows how to love without being told")
        cr.move_to(20, 180)
        cr.show_text("Somebody tell me why I'm on my own")
        cr.move_to(20, 210)
        cr.show_text("If there's a soulmate for everyone")


PyApp()
gtk.main()
```

我们从 Natasha Bedingfields Soulmate 这首歌显示了部分歌词。

```
cr.select_font_face("Purisa", cairo.FONT_SLANT_NORMAL,
    cairo.FONT_WEIGHT_NORMAL)
```

这里我们指定了我们要使用的字体。

```
cr.set_font_size(13)
```

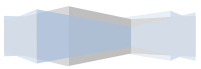
这里我们指定了字体的大小。

```
cr.move_to(20, 30)
```

这里我们将点移动到我们要绘制文本的地方。

```
cr.show_text("Most relationships seem so transitory")
```

用 **show_text()**方法在窗口上绘制文本。



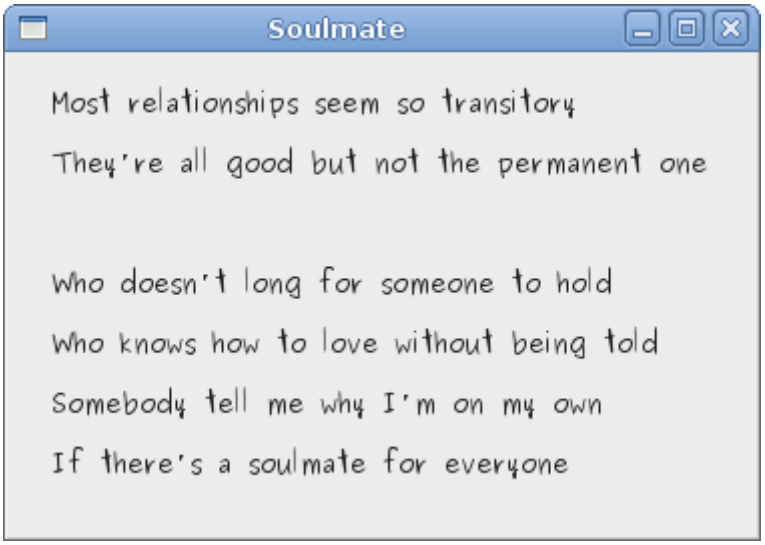


Figure: Soulmate

在 PyGTK 编程库的这章中，我们使用 Cairo 图形库进行绘制工作。

13 用 Cairo 库绘图 II

在教程的这个部分，我们将继续用 Cairo 图形库进行绘图。

Donut

在下面的例子中，我们通过旋转一串椭圆来创建一个复杂的形状。

Code: donut.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# This program creates a donut
# with cairo library
#
# author: jan bodnar
# website: zetcode.com
# last edited: February 2009


import gtk
import math

class PyApp(gtk.Window):

    def __init__(self):
        super(PyApp, self).__init__()

        self.set_title("Donut")
```

```
        self.set_size_request(350, 250)
        self.set_position(gtk.WIN_POS_CENTER)

        self.connect("destroy", gtk.main_quit)

        darea = gtk.DrawingArea()
        darea.connect("expose-event", self.expose)
        self.add(darea)

        self.show_all()

    def expose(self, widget, event):

        cr = widget.window.cairo_create()

        cr.set_line_width(0.5)

        w = self.allocation.width
        h = self.allocation.height

        cr.translate(w/2, h/2)
        cr.arc(0, 0, 120, 0, 2*math.pi)
        cr.stroke()


        for i in range(36):
            cr.save()
            cr.rotate(i*math.pi/36)
            cr.scale(0.3, 1)
            cr.arc(0, 0, 120, 0, 2*math.pi)
            cr.restore()
            cr.stroke()


PyApp()
gtk.main()
```

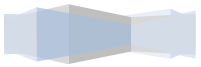
在这个例子中，我们创建了一个环状线圈。它的形状有点像一种甜点，因此名字叫做 donut。

```
cr.translate(w/2, h/2)
cr.arc(0, 0, 120, 0, 2*math.pi)
cr.stroke()
```

最开始，有一个椭圆。

```
for i in range(36):
    cr.save()
    cr.rotate(i*math.pi/36)
    cr.scale(0.3, 1)
    cr.arc(0, 0, 120, 0, 2*math.pi)
    cr.restore()
    cr.stroke()
```

几个旋转之后，就有了一个环状线圈。我们使每个旋转孤立，并且通过 **save()**和 **restore()** 方法一个接一个的进行缩放操作。



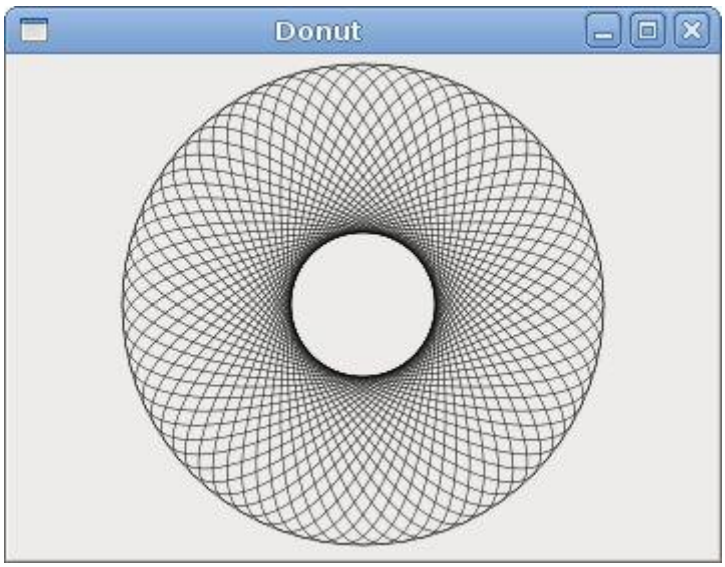


Figure: Donut

Gradients

在计算机图形中，梯度（Gradient）是一种光滑的渐变混合，从亮到暗或者从一种颜色到另一种。在 2D 绘图程序和喷绘程序中，梯度常被用来创建色彩斑斓的背景和类似模仿的光和影一样的效果。（引用 answers.com）

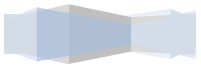
Code: gradients.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# This program works with
# gradients in cairo
#
# author: jan bodnar
# website: zetcode.com
# last edited: February 2009

import gtk
import cairo

class PyApp(gtk.Window):

    def __init__(self):
        super(PyApp, self).__init__()

        self.set_title("Gradients")
        self.set_size_request(340, 390)
        self.set_position(gtk.WIN_POS_CENTER)

        self.connect("destroy", gtk.main_quit)
```

```
        darea = gtk.DrawingArea()
        darea.connect("expose-event", self.expose)
        self.add(darea)

        self.show_all()

    def expose(self, widget, event):

        cr = widget.window.cairo_create()
        lg1 = cairo.LinearGradient(0.0, 0.0, 350.0, 350.0)

        count = 1

        i = 0.1
        while i < 1.0:
            if count % 2:
                lg1.add_color_stop_rgba(i, 0, 0, 0, 1)
            else:
                lg1.add_color_stop_rgba(i, 1, 0, 0, 1)
            i = i + 0.1
            count = count + 1


        cr.rectangle(20, 20, 300, 100)
        cr.set_source(lg1)
        cr.fill()

        lg2 = cairo.LinearGradient(0.0, 0.0, 350.0, 0)

        count = 1

        i = 0.05
        while i < 0.95:
            if count % 2:
                lg2.add_color_stop_rgba(i, 0, 0, 0, 1)
            else:
                lg2.add_color_stop_rgba(i, 0, 0, 1, 1)
            i = i + 0.025
            count = count + 1

        cr.rectangle(20, 140, 300, 100)
        cr.set_source(lg2)
        cr.fill()

        lg3 = cairo.LinearGradient(20.0, 260.0,  20.0, 360.0)
        lg3.add_color_stop_rgba(0.1, 0, 0, 0, 1)
        lg3.add_color_stop_rgba(0.5, 1, 1, 0, 1)
        lg3.add_color_stop_rgba(0.9, 0, 0, 0, 1)

        cr.rectangle(20, 260, 300, 100)
        cr.set_source(lg3)
        cr.fill()
```



```
PyApp()
gtk.main()
```



在我们的例子中，我们用三种不同的梯度来绘制了三个矩形。

```
lg1 = cairo.LinearGradient(0.0, 0.0, 350.0, 350.0)
```

这里我们创建了一个线形的梯度图案。参数指定了线沿着我们想画的梯度的方向。在这里它是垂直方向的线。

```
lg3 = cairo.LinearGradient(20.0, 260.0,  20.0, 360.0)
lg3.add_color_stop_rgba(0.1, 0, 0, 0, 1)
lg3.add_color_stop_rgba(0.5, 1, 1, 0, 1)
lg3.add_color_stop_rgba(0.9, 0, 0, 0, 1)
```

为了产生我们的梯度图案，我们定义了颜色停顿。在这里，梯度是黑色和黄色的混合。通过给一个黄色停顿增加两个黑色，我们创建了一个水平的梯度图案。这些停顿实际上是什么意思呢？在这里，我们以黑色开始，它将在 1/10 大小的位置上停顿，然后我们逐渐用黄色进行喷绘，它将在到达图形的中间达到最亮。黄色在 9/10 大小的位置停止。这里我们重新开始用黑色进行喷绘，知道结尾。

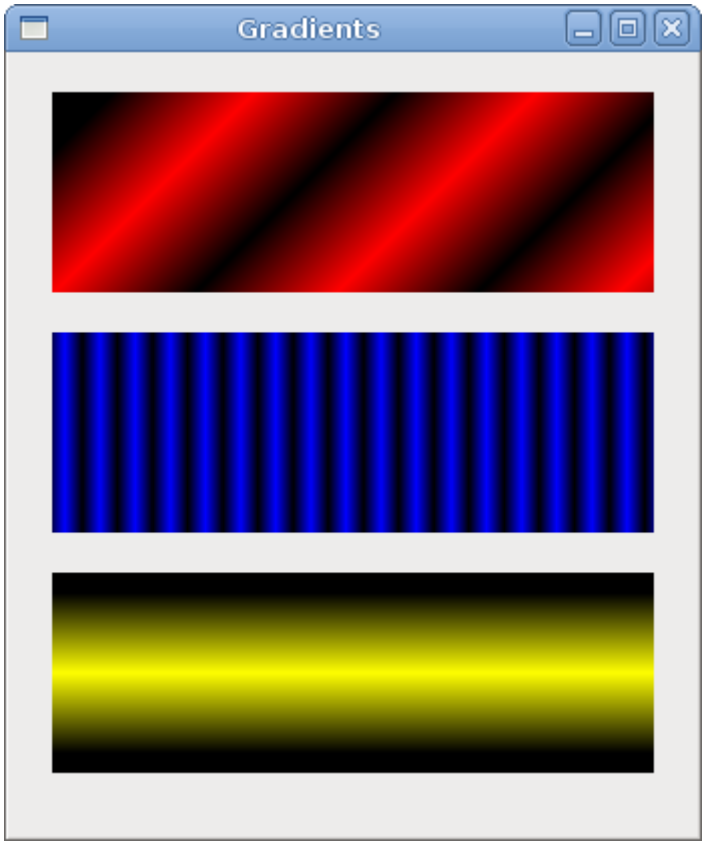


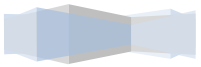
Figure: Gradients

Puff

在下面的例子中，我们将创建膨胀（puff）效果。例子显示正在变大的居中文本。最后从一些端点逐渐淡出。这是非常通用的效果，你可以在 flash 动画中经常看到这种效果。

Code: puff.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# This program creates a puff
# effect
#
# author: jan bodnar
# website: zetcode.com
# last edited: February 2009


import gtk
import glib
import cairo


class PyApp(gtk.Window):

    def __init__(self):
        super(PyApp, self).__init__()

        self.set_title("Puff")
        self.resize(350, 200)
        self.set_position(gtk.WIN_POS_CENTER)

        self.connect("destroy", gtk.main_quit)

        self.darea = gtk.DrawingArea()
        self.darea.connect("expose-event", self.expose)
        self.add(self.darea)

        self.timer = True
        self.alpha = 1.0
        self.size = 1.0

        glib.timeout_add(14, self.on_timer)

        self.show_all()

    def on_timer(self):
        if not self.timer: return False

        self.darea.queue_draw()
        return True


    def expose(self, widget, event):

        cr = widget.window.cairo_create()

        w = self.allocation.width
        h = self.allocation.height

        cr.set_source_rgb(0.5, 0, 0)
        cr.paint()
```

```
        cr.select_font_face("Courier",          cairo.FONT_SLANT_NORMAL,
cairo.FONT_WEIGHT_BOLD)

        self.size = self.size + 0.8

        if self.size > 20:
            self.alpha = self.alpha - 0.01

        cr.set_font_size(self.size)
        cr.set_source_rgb(1, 1, 1)

        (x, y, width, height, dx, dy) = cr.text_extents("ZetCode")

        cr.move_to(w/2 - width/2, h/2)
        cr.text_path("ZetCode")
        cr.clip()
        cr.stroke()
        cr.paint_with_alpha(self.alpha)

        if self.alpha <= 0:
            self.timer = False


PyApp()
gtk.main()
```

这个例子在窗口创建了一个增大和淡出的文本。

```
glib.timeout_add(14, self.on_timer)
```

每隔 14ms **on_timer()**方法会被调用。

```
def on_timer(self):
    if not self.timer: return False

    self.darea.queue_draw()
    return True
```

在 **on_timer()**方法中 ,我们在绘图区上调用了 **queue_draw()**方法 ,该方法将触发 expose 信号。

```
cr.set_source_rgb(0.5, 0, 0)
cr.paint()
```

我们设置背景颜色为暗红色。

```
self.size = self.size + 0.8
```

每个循环，字体的大小将会增加 0.8 个单位。

```
if self.size > 20:
    self.alpha = self.alpha - 0.01
```


当字体的尺寸大于 20 的之后，淡出过程就开始了。

```
(x, y, width, height, dx, dy) = cr.text_extents("ZetCode")
```

我们获得了文本的公制尺度。

```
cr.move_to(w/2 - width/2, h/2)
```

我们用文本的公制尺度来使文本居于窗口的中心。

```
cr.text_path("ZetCode")
cr.clip()
```

我们获得了文本的路径，并且将当前的修剪部位设置为这个路径。

```
cr.stroke()
cr.paint_with_alpha(self.alpha)
```

我们绘出当前的路径，并且将 alpha 值加入到这个过程中去。

Figure: Puff

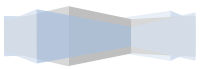
Reflection

在下面的例子中，我们将显示一个反射的图像。这个漂亮的效果会使人产生一种幻觉，就像一幅图像在水里被反射一样。

Code: reflection.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

# ZetCode PyGTK tutorial
#
# This program creates an
# image reflection
#
# author: Jan Bodnar
# website: zetcode.com
# last edited: April 2011


import gtk
import cairo
import sys
```

```
class PyApp(gtk.Window):

    def __init__(self):
        super(PyApp, self).__init__()

        self.set_title("Reflection")
        self.resize(300, 350)
        self.set_position(gtk.WIN_POS_CENTER)

        self.connect("destroy", gtk.main_quit)

        darea = gtk.DrawingArea()
        darea.connect("expose-event", self.expose)
        self.add(darea)

        try:
            self.surface                                                    =
cairo.ImageSurface.create_from_png("slanec.png")
        except Exception, e:
            print e.message
            sys.exit(1)


        self.imageWidth = self.surface.get_width()
        self.imageHeight = self.surface.get_height()
        self.gap = 40
        self.border = 20

        self.show_all()

    def expose(self, widget, event):

        cr = widget.window.cairo_create()


        w = self.allocation.width
        h = self.allocation.height

        lg = cairo.LinearGradient(w/2, 0, w/2, h*3)
        lg.add_color_stop_rgba(0, 0, 0, 0, 1)
        lg.add_color_stop_rgba(h, 0.2, 0.2, 0.2, 1)

        cr.set_source(lg)
        cr.paint()

        cr.set_source_surface(self.surface, self.border, self.border)
        cr.paint()

        alpha = 0.7
        step = 1.0 / self.imageHeight

        cr.translate(0, 2 * self.imageHeight + self.gap)
        cr.scale(1, -1)

        i = 0
```

```
        while(i < self.imageHeight):

            cr.rectangle(self.border, self.imageHeight-i, self.imageWidth,
1)

            i = i + 1

            cr.save()
            cr.clip()
            cr.set_source_surface(self.surface, self.border, self.border)
            alpha = alpha - step
            cr.paint_with_alpha(alpha)
            cr.restore()


PyApp()
gtk.main()
```

这个例子展示了一个反射的城堡。

```
lg = cairo.LinearGradient(w/2, 0, w/2, h*3)
lg.add_color_stop_rgba(0, 0, 0, 0, 1)
lg.add_color_stop_rgba(h, 0.2, 0.2, 0.2, 1)

cr.set_source(lg)
cr.paint()
```

背景是用一种梯度的喷绘填充的。这个喷绘是一种从黑色到暗灰色的平滑的组合。

```
cr.translate(0, 2 * self.imageHeight + self.gap)
cr.scale(1, -1)
```

这行代码将图像翻转，使其转变为在原始图像的下面。Translation 操作是很必要的，因为 scaling 操作使这个图像上部下来，然后再 translate 这个图像。要理解这里发生了什么，请找一张照片，将它放在桌子上，现在翻转（flip）它。

```
cr.rectangle(self.border, self.imageHeight-i, self.imageWidth, 1)

i = i + 1

cr.save()
cr.clip()
cr.set_source_surface(self.surface, self.border, self.border)
alpha = alpha - step
cr.paint_with_alpha(alpha)
cr.restore()
```

这是代码的关键部分。我们使第二幅图像变得透明，但是这个透明度并不是恒定的。图像逐渐淡出，这就是用梯度来完成的。

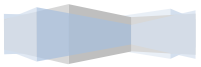


Figure: Reflection

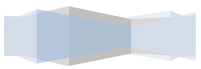
Waiting

在这个例子中，我们使用透明度效果来创建一个等待的演示实例。我们将画 8 条线，它们将逐渐淡出，这样就创建了一个幻觉——一条线在运动。这个效果通常用来通知用户，一个冗长的任务正在后台运行。一个典型的例子是从因特网上打开一个视频。

Code: waiting.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# This program creates an
# waiting effect
#
# author: jan bodnar
# website: zetcode.com
# last edited: February 2009


import gtk
import glib
import math
import cairo


trs = (
    ( 0.0, 0.15, 0.30, 0.5, 0.65, 0.80, 0.9, 1.0 ),
    ( 1.0, 0.0,  0.15, 0.30, 0.5, 0.65, 0.8, 0.9 ),
```

```
    ( 0.9, 1.0,  0.0,  0.15, 0.3, 0.5, 0.65, 0.8 ),
    ( 0.8, 0.9,  1.0,  0.0,  0.15, 0.3, 0.5, 0.65 ),
    ( 0.65, 0.8, 0.9,  1.0,  0.0,  0.15, 0.3, 0.5 ),
    ( 0.5, 0.65, 0.8, 0.9, 1.0,  0.0,  0.15, 0.3 ),
    ( 0.3, 0.5, 0.65, 0.8, 0.9, 1.0,  0.0,  0.15 ),
    ( 0.15, 0.3, 0.5, 0.65, 0.8, 0.9, 1.0,  0.0, )
)


class PyApp(gtk.Window):

    def __init__(self):
        super(PyApp, self).__init__()

        self.set_title("Waiting")
        self.set_size_request(250, 150)
        self.set_position(gtk.WIN_POS_CENTER)

        self.connect("destroy", gtk.main_quit)

        self.darea = gtk.DrawingArea()
        self.darea.connect("expose-event", self.expose)
        self.add(self.darea)

        self.count = 0

        glib.timeout_add(100, self.on_timer)

        self.show_all()

    def on_timer(self):
        self.count = self.count + 1
        self.darea.queue_draw()
        return True


    def expose(self, widget, event):

        cr = widget.window.cairo_create()

        cr.set_line_width(3)
        cr.set_line_cap(cairo.LINE_CAP_ROUND)

        w = self.allocation.width
        h = self.allocation.height

        cr.translate(w/2, h/2)

        for i in range(8):
            cr.set_source_rgba(0, 0, 0, trs[self.count%8][i])
            cr.move_to(0.0, -10.0)
            cr.line_to(0.0, -40.0)
            cr.rotate(math.pi/4)
            cr.stroke()


PyApp()
gtk.main()
```

我们用 8 个不同的 alpha 值画了 8 条线。

```
glib.timeout_add(100, self.on_timer)
```

我们使用了一个 timer 函数来创建动画。

```
trs = (
    ( 0.0, 0.15, 0.30, 0.5, 0.65, 0.80, 0.9, 1.0 ),
    ...
)
```

这是一个二维的元组，其值为用于该例子的透明度值。这里有八行，每行是一种状态。8 条线每条将会陆续地使用这些值。

```
cr.set_line_width(3)
cr.set_line_cap(cairo.LINE_CAP_ROUND)
```

我们是线宽稍微大点，这样才会有更好的视觉效果。我们用环形的 cap 来画这些线。他们看起来更好。

```
cr.set_source_rgba(0, 0, 0, trs[self.count%8][i]
```

这里我们定义了一条线的透明度值。

```
cr.move_to(0.0, -10.0)
cr.line_to(0.0, -40.0)
cr.rotate(math.pi/4)
cr.stroke()
```

这些代码行将绘制出八条线中的每条。

Figure: Waiting

在教程的这章中，我们用 Cairo 图形库做了一些更高级的绘画工作。

14 PyGTK 中的 Snake 游戏

在教程的这个部分，我们将创建一个 Snake 游戏的克隆版。

Snake game

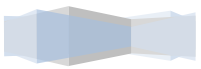
Snake（贪吃蛇）是一个古老的经典的视频游戏。它是在 20 世纪 70 年代后期第一次被创建。后来它被引入到 PC 机上。在这个游戏中，玩家控制一条蛇，目标即是尽可能地吃足够多的苹果。每次这个蛇吃下一个苹果，它的身体就会长长。这条蛇必须避免撞上墙和它自己的身体。这个游戏有时候被称为 **Nibbles**。

Development

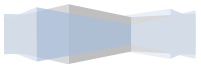
蛇的每个节点的大小被设计为 10 像素。蛇被方向键（cursor keys）控制。最初蛇有 3 个节点。游戏会立刻开始。如果游戏完成了，我们将在面板的中间显示“Game Over”的信息。

Code: snake.py

```
#!/usr/bin/python

# ZetCode PyGTK tutorial
#
# This is a simple snake game
# clone
#
# author: jan bodnar
# website: zetcode.com
# last edited: February 2009

import sys
import gtk
import cairo
import random
import glib


WIDTH = 300
HEIGHT = 270
DOT_SIZE = 10
ALL_DOTS = WIDTH * HEIGHT / (DOT_SIZE * DOT_SIZE)
RAND_POS = 26
DELAY = 100

x = [0] * ALL_DOTS
y = [0] * ALL_DOTS


class Board(gtk.DrawingArea):

    def __init__(self):
        super(Board, self).__init__()

        self.modify_bg(gtk.STATE_NORMAL, gtk.gdk.Color(0, 0, 0))
        self.set_size_request(WIDTH, HEIGHT)
```

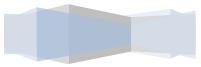



```
        self.connect("expose-event", self.expose)

        self.init_game()

    def on_timer(self):

        if self.inGame:
            self.check_apple()
            self.check_collision()
            self.move()
            self.queue_draw()
            return True
        else:
            return False

    def init_game(self):

        self.left = False
        self.right = True
        self.up = False
        self.down = False
        self.inGame = True
        self.dots = 3

        for i in range(self.dots):
            x[i] = 50 - i * 10
            y[i] = 50

        try:
            self.dot = cairo.ImageSurface.create_from_png("dot.png")
            self.head = cairo.ImageSurface.create_from_png("head.png")
            self.apple = cairo.ImageSurface.create_from_png("apple.png")
        except Exception, e:
            print e.message
            sys.exit(1)

        self.locate_apple()
        glib.timeout_add(DELAY, self.on_timer)



    def expose(self, widget, event):

        cr = widget.window.cairo_create()

        if self.inGame:
            cr.set_source_rgb(0, 0, 0)
            cr.paint()

            cr.set_source_surface(self.apple, self.apple_x, self.apple_y)
            cr.paint()

            for z in range(self.dots):
                if (z == 0):
                    cr.set_source_surface(self.head, x[z], y[z])
                    cr.paint()
                else:
```

```
                cr.set_source_surface(self.dot, x[z], y[z])
                cr.paint()
        else:
            self.game_over(cr)


    def game_over(self, cr):

        w = self.allocation.width / 2
        h = self.allocation.height / 2

        (x, y, width, height, dx, dy) = cr.text_extents("Game Over")

        cr.set_source_rgb(65535, 65535, 65535)
        cr.move_to(w - width/2, h)
        cr.show_text("Game Over")
        self.inGame = False


    def check_apple(self):

        if x[0] == self.apple_x and y[0] == self.apple_y:
            self.dots = self.dots + 1
            self.locate_apple()


    def move(self):

        z = self.dots

        while z > 0:
            x[z] = x[(z - 1)]
            y[z] = y[(z - 1)]
            z = z - 1

        if self.left:
            x[0] -= DOT_SIZE

        if self.right:
            x[0] += DOT_SIZE

        if self.up:
            y[0] -= DOT_SIZE

        if self.down:
            y[0] += DOT_SIZE


    def check_collision(self):

        z = self.dots

        while z > 0:
            if z > 4 and x[0] == x[z] and y[0] == y[z]:
                self.inGame = False
```

```
            z = z - 1

        if y[0] > HEIGHT - DOT_SIZE:
            self.inGame = False

        if y[0] < 0:
            self.inGame = False

        if x[0] > WIDTH - DOT_SIZE:
            self.inGame = False

        if x[0] < 0:
            self.inGame = False


    def locate_apple(self):

        r = random.randint(0, RAND_POS)
        self.apple_x = r * DOT_SIZE
        r = random.randint(0, RAND_POS)
        self.apple_y = r * DOT_SIZE


    def on_key_down(self, event):

        key = event.keyval

        if key == gtk.keysyms.Left and not self.right:
            self.left = True
            self.up = False
            self.down = False


        if key == gtk.keysyms.Right and not self.left:
            self.right = True
            self.up = False
            self.down = False


        if key == gtk.keysyms.Up and not self.down:
            self.up = True
            self.right = False
            self.left = False


        if key == gtk.keysyms.Down and not self.up:
            self.down = True
            self.right = False
            self.left = False


class Snake(gtk.Window):

    def __init__(self):
        super(Snake, self).__init__()

        self.set_title('Snake')
        self.set_size_request(WIDTH, HEIGHT)
```

```
        self.set_resizable(False)
        self.set_position(gtk.WIN_POS_CENTER)

        self.board = Board()
        self.connect("key-press-event", self.on_key_down)
        self.add(self.board)

        self.connect("destroy", gtk.main_quit)
        self.show_all()


    def on_key_down(self, widget, event):

        key = event.keyval
        self.board.on_key_down(event)


Snake()
gtk.main()
```

首先，我们定义了一些在游戏中要使用的全局常量和变量。

WIDTH 和 **HEIGHT** 常量确定了游戏面板的大小。**DOT_SIZE** 就是苹果和蛇的节点的大小。**ALL_DOTS** 常量是定义了在面板上可能有的点的最大数字。**RAND_POS** 常量被用来计算一个苹果的随机位置。DELAY 常量是设定游戏的速度。

```
x = [0] * ALL_DOTS
y = [0] * ALL_DOTS
```

这两个列表存储蛇的所有可能的节点的 x, y 坐标。

init_game()方法初始化变量，加载图像，开始一个超时函数。

```
self.left = False
self.right = True
self.up = False
self.down = False
self.inGame = True
self.dots = 3
```

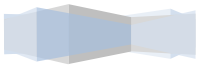
当我们的游戏开始后，蛇有三个节点，并且头朝向右边。

在 **move()**方法中，我们有游戏的关键算法。为了理解它，请看看蛇是怎样在运动。你控制蛇的头部，通过方向键，你可以改变它的方向。剩下的节点在链上移动一个位置。第二个节点移动到第一个节点原来所在的位置，第三个节点移动到第二个节点原来所在的位置等等。

```
while z > 0:
    x[z] = x[(z - 1)]
    y[z] = y[(z - 1)]
    z = z - 1
```

这里的代码将节点移动到链上。

```
if self.left:
    x[0] -= DOT_SIZE
```

将头部移向左边。

在 **checkCollision()**方法中，我们来判断蛇是否撞到它自己或者其中一面墙。

```
while z > 0:
    if z > 4 and x[0] == x[z] and y[0] == y[z]:
        self.inGame = False
    z = z - 1
```

如果蛇用头部撞上了它的一个节点，游戏结束。

```
if y[0] > HEIGHT - DOT_SIZE:
    self.inGame = False
```

如果蛇撞上了面板的底部，游戏结束。

locate_apple()方法是在面板中随机地给苹果定位。

```
r = random.randint(0, RAND_POS)
```

我们获得一个从 0 到 RAND_POS-1 的随机数。

```
self.apple_x = r * DOT_SIZE
...
self.apple_y = r * DOT_SIZE
```

这几行设置苹果这个对象的 x, y 坐标。

```
    self.connect("key-press-event", self.on_key_down)
    ...

def on_key_down(self, widget, event):

    key = event.keyval
    self.board.on_key_down(event)
```

我们在 Snake 类中捕获摁键事件，并且对面板对象委派一个处理过程。

在 Board 类的 **on_key_down()**方法中，我们判断玩家点击了哪个按键。

```
if key == gtk.keysyms.Left and not self.right:
    self.left = True
    self.up = False
    self.down = False
```

如果我们按下左方向键，我们设置 **self.left** 变量为真，这个变量被用在 **move()**方法中来改变蛇对象的坐标。我们也应该注意到，当蛇想右边前行的时候，我们不能立刻改为向左。

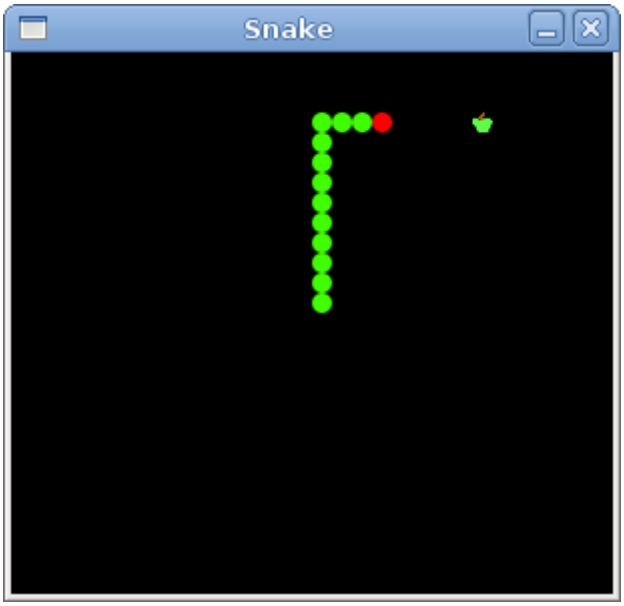


Figure: Snake

这是用 PyGTK 编程库制作的贪吃蛇电脑游戏。

15 PyGTK 中的自定义部件

你是否曾经看到一个程序就在想，一个特别的 GUI 项目是怎样被创建的呢？或许每个程序员都想知道。然后你就看看你最喜欢的 GUI 库提供的部件列表。但是你找不到它。工具包一般只提供最常见的部件，例如按钮，文本部件，滑动器等等。没有哪个工具包提供所有可能的部件。

实际上有两种工具包——简朴的工具包和重量级的工具包。FLTK 工具包是一种简朴的工具包。它仅仅提供最基本的部件和呈现方法，但是程序员可以自己创建一个更加复杂的部件。wxWidgets 是一个重量级的工具包，它有相当多的部件。但是它也不提供更加专业的部件。例如一个速度仪表部件，一个用来检测 CD 能被烧制的容量的部件(例如 :在 nero 中就有)。工具包也通常不会有图表。

程序员必须通过他们自己来创建一个这样的部件。他们可以通过工具包中提供的绘画工具做到。这里有两种可行的方法。一是，程序员可以更改或者增强现有的部件；或者二是，程序员从零起创建一个部件。

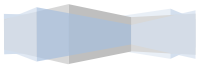
Burning widget

这是一个我们从零起创建的一个部件的例子。这个部件能够在各种媒体烧制程序中见到，就像 Nero Burning ROM。

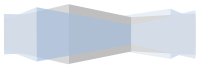
Code: burning.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
```

```
# ZetCode PyGTK tutorial
#
# This example creates a burning
# custom widget
#
# author: Jan Bodnar
# website: zetcode.com
# last edited: April 2011


import gtk
import cairo

class Burning(gtk.DrawingArea):

    def __init__(self, parent):

        self.par = parent
        super(Burning, self).__init__()

        self.num = ( "75", "150", "225", "300",
            "375", "450", "525", "600", "675" )

        self.set_size_request(-1, 30)
        self.connect("expose-event", self.expose)


    def expose(self, widget, event):

        cr = widget.window.cairo_create()
        cr.set_line_width(0.8)

        cr.select_font_face("Courier",
            cairo.FONT_SLANT_NORMAL, cairo.FONT_WEIGHT_NORMAL)
        cr.set_font_size(11)

        width = self.allocation.width

        self.cur_width = self.par.get_cur_value()

        step = round(width / 10.0)

        till = (width / 750.0) * self.cur_width
        full = (width / 750.0) * 700

        if (self.cur_width >= 700):

            cr.set_source_rgb(1.0, 1.0, 0.72)
            cr.rectangle(0, 0, full, 30)
            cr.save()
            cr.clip()
            cr.paint()
            cr.restore()

            cr.set_source_rgb(1.0, 0.68, 0.68)
            cr.rectangle(full, 0, till-full, 30)
            cr.save()
```

```
            cr.clip()
            cr.paint()
            cr.restore()

        else:
            cr.set_source_rgb(1.0, 1.0, 0.72)
            cr.rectangle(0, 0, till, 30)
            cr.save()
            cr.clip()
            cr.paint()
            cr.restore()


        cr.set_source_rgb(0.35, 0.31, 0.24)

        for i in range(1, len(self.num) + 1):
            cr.move_to(i*step, 0)
            cr.line_to(i*step, 5)
            cr.stroke()

            (x, y, width, height, dx, dy) = cr.text_extents(self.num[i-1])
            cr.move_to(i*step-width/2, 15)
            cr.text_path(self.num[i-1])
            cr.stroke()


class PyApp(gtk.Window):

    def __init__(self):
        super(PyApp, self).__init__()

        self.set_title("Burning")
        self.set_size_request(350, 200)
        self.set_position(gtk.WIN_POS_CENTER)
        self.connect("destroy", gtk.main_quit)

        self.cur_value = 0

        vbox = gtk.VBox(False, 2)

        scale = gtk.HScale()
        scale.set_range(0, 750)
        scale.set_digits(0)
        scale.set_size_request(160, 40)
        scale.set_value(self.cur_value)
        scale.connect("value-changed", self.on_changed)

        fix = gtk.Fixed()
        fix.put(scale, 50, 50)

        vbox.pack_start(fix)

        self.burning = Burning(self)
        vbox.pack_start(self.burning, False, False, 0)

        self.add(vbox)
        self.show_all()
```

```
    def on_changed(self, widget):
        self.cur_value = widget.get_value()
        self.burning.queue_draw()


    def get_cur_value(self):
        return self.cur_value


PyApp()
gtk.main()
```

我们在窗口的底部放置了以 **DrawingArea** 区域，准备手动绘制整个部件。所有重要的代码都属于 Burning 类的 **expose()**方法。这个部件以图像的形式展现了一个媒体总的容量和空余可用的空间。这个部件被一个标尺部件所控制。我们自定义部件的最小值为 0，最大值为 750。如果我们到达值 700，我们就开始用红色绘制它。这个一般在提示超容量烧录了。

```
self.num = ( "75", "150", "225", "300",
    "375", "450", "525", "600", "675" )
```

这些数字会在 burning 部件上显示。它们指示的是一个媒体的容量。

```
self.cur_width = self.par.get_cur_value()
```

这两行是从标尺部件上获取当前的数据。我们从父部件获得了父部件，然后我们获得了当前值。

```
till = (width / 750.0) * self.cur_width
full = (width / 750.0) * 700
```

till 参数是判定将要绘制的总共大小。这个值来自滑动器部件。它是整个区域的一个比例。fill 参数是判定我们从哪里开始用红色来绘制。

```
cr.set_source_rgb(1.0, 1.0, 0.72)
cr.rectangle(0, 0, till, 30)
cr.save()
cr.clip()
cr.paint()
cr.restore()
```

这些代码这里在媒体满值之前绘制了一个黄色的矩形。

```
(x, y, width, height, dx, dy) = cr.text_extents(self.num[i-1])
cr.move_to(i*step-width/2, 15)
cr.text_path(self.num[i-1])
cr.stroke()
```

这些代码这里在 burning 部件上绘制数字。我们将正确的计算 **TextExtents** 到文本的位置。

```
def on_changed(self, widget):
    self.cur_value = widget.get_value()
    self.burning.queue_draw()
```


我们从标尺部件获得值，将其存储在 **cur_value** 变量中留待后用。我们重画 burning 部件。

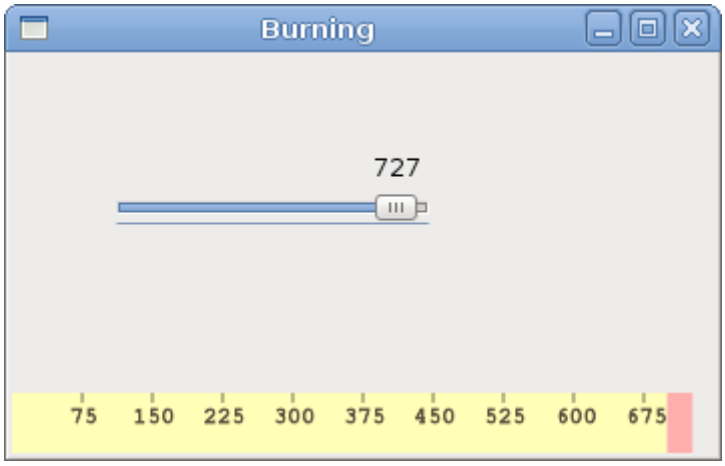


Figure: Burning Widget

在这章中，我们用 PyGTK 创建了一个自定义的部件。