GTK+ and Glade3 GUI Programming Tutorial - Part 1

2009年3月27日
23:38
原文地址链接 www.micahcarrick.com/12-24-2007/gtk-glade-tutorial-part-1.html
作者 Micah Carrick
翻译：Binn.X Wee
博客链接：http://blog.csdn.net/xbwee
本文链接：http://blog.csdn.net/xbwee/archive/2009/03/28/4032652.aspx

Quick Overview of GTK+ Concepts

如果你没有任何GTK+ 的编程经验， 那么， 对于我将要阐述的一些概念你也许会听着犯迷糊。不过，不用担心， 在遇到这些概念的时候我会详细讲解，以便你能很好的阅读后面的内容。学完这一部分， 对GTK+ 的基本概念有所了解后， 你也许就能有效的利用Glade进行开发了。

首先， GTK+ 并不是一门编程语言， 而是一个开发工具套件， 或者说是一个开发库， 用来进行跨平台GUI应用程序的开发，Linux， OSX， Windows或其它任何平台都能使用GTK+。GTK+ 就好比Windows上的MFC 和Win32 API， JAVA 上的Swing和SWT， 或者Qt（KDE 使用的Linux下GUI开发套件）。

尽管 GTK+ 是用纯C语言编写的， 但是提供了其它各种语言的捆绑， 允许程序员选择自己喜欢的开发语言来开发 GTK+ 应用程序， 比如 C++, Python， Perl， PHP， Ruby等等。

GTK+ 开发套件基于三个主要的库： Glib， Pango， 和 ATK，当然我们只需关心如何使用 GTK+ 即可， GTK+ 自己负责与这三个库打交道。 Glib 封装了大部分可移植的 C 库函数（允许你的代码移植到 Windows 和 Linux 上运行）。使用 C 或 C++ 时，将大量使用 Glib 库函数， 在我们用 C 语言的具体实现过程中我会详细解释它们。高级语言如 Python 和 Ruby 却不用担心 Glib 的使用， 因为它们有自己的标准库提供了相应的功能。

GTK+ 及相关的库时按照面向对象设计思想来实现的， 至于这时如何实现的现在并不重要， 不同的编程语言有不同的实现方法， 重要的是要知道 GTK+ 使用面向对象编程技术即可（是的， 即使是 C 实现的）。

每一个 GTK+ 的GUI元素都是由一个或许多个 “widgets”对象构成的。 所有的widgets都从基类GtkWidget派生。例如， 应用程序的主窗口是GtkWindow类widget， 窗口的工具条是GtkToolbar类widget。 一个GtkWindow是一个GtkWidget， 但一个GtkWidget兵不是一个GtkWindow， 子类widgets 继承自父类并扩展了父类的功能而成为一个新类， 这就是标准的面向对象 编程OOP(Object Oriented Programming)思想。

我们可以查阅GTK+参考手册找到widgets直接的继承关系。 对于GtkWindow它的继承链看起来像这样：

```
GObject
   +----GInitiallyUnowned
         +----GtkObject
               +----GtkWidget
                     +----GtkContainer
                           +----GtkBin
                                 +----GtkWindow
```

因此， GtkWindow继承自GtkBin， GtkBin继承自GtkContainer， 等等。在第一个程序中，你不需要担心GtkWidget对象。 各widget之间的继承链之所以重要是因为当你查找某个widget的函数， 属性和信号时， 你应该知道它的父类的函数，属性和信号也被此widget继承了，可以直接使用。在第二部分讲述此实例的代码时， 你能更清楚的认识到这一点。

我们来看命名规则， 命名规则带来的好处是非常便于使用。我们能够清楚 的看出对象或函数是哪个库中的。以Gtk开头的所有对象都是在GTK+中定义的。 稍后我们会看到类似GladeXML以Glade开头的是Libglade库对象或函数， GError以G开头的在GLib库定义。所有Widgets类都遵循标准camelcase命名习惯。所有操作函数都以下划线组合小写字母单词命名。如gtk_window_set_title（）设置GtkWindow对象的标题属性。

你需要的所有参考文档都可以从以下网站获得：library.gnome.org/devel/references,

不过， 使用Devhelp更方便， 它甚至可以作为一个包来分发。 Devhelp可以浏览或搜索任何安装在你系统上的库的相关文档， 当然前提是你必须安装了这些文档。

Introduction to Glade3

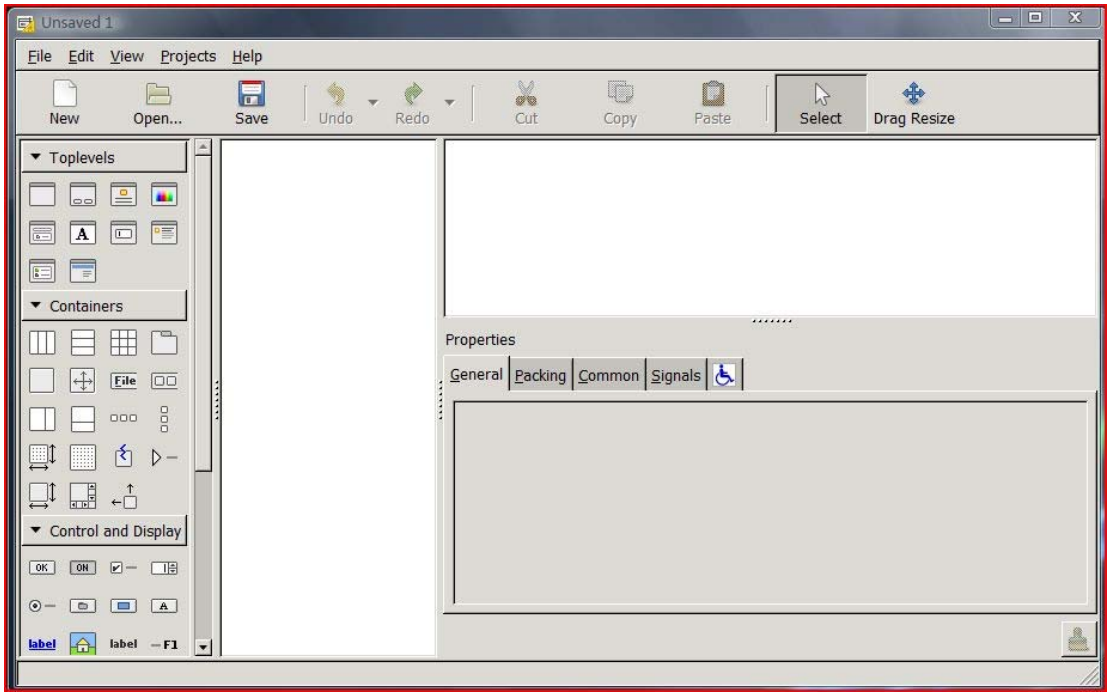
Glade 是一种开发GTK+ 应用程序的 RAD（Rapid Application Development）工具。 Glade自身就是一个GTK+应用程序，因为它就是用GTK+ 开发出来的^_^ Glade用来简化 UI 控件的设计和布局操作， 进行快速开发。（译者注：当然，还不仅如此， Glade的设计初衷是把界面设计与应用程序代码相分离， 界面的修改不会影响到应用程序代码）Glade设计的界面保存为 glade格式文件，它实际上是一种XML文件。

Glade 起先能根据创建的 GUI 自动生成 C 语言代码（你仍然能找到此类相关的实例）， 后来可以利用Libglade库在运行时动态创建界面， 到了Glade3 ，这些方法都不赞成使用了。这是因为， Glade需要做的唯一的事就是生成一个描述如何创建GUI的glade文件。这给编程人员提供了更多的灵活性和弹性， 避免了用户界面部分微小的改变就要重新编译整个应用程序， 同时其语言无关性， 几乎所有的编程语言都可以使用Glade。

Glade3 进行了重新设计， 与之前的版本如 Glade2 有巨大的改变。2006年Glade3.0发布， 你可以自由获取最新版本进行开发。软件包管理器如 aptitude等应该都有Glade3的安装包， 不过请注意：有个数字 3， 因为"glade"是老版本的Glade2，"Glade-3" 或"Glade3"才是新版本。 你也可以从glade.gnome.org下载。

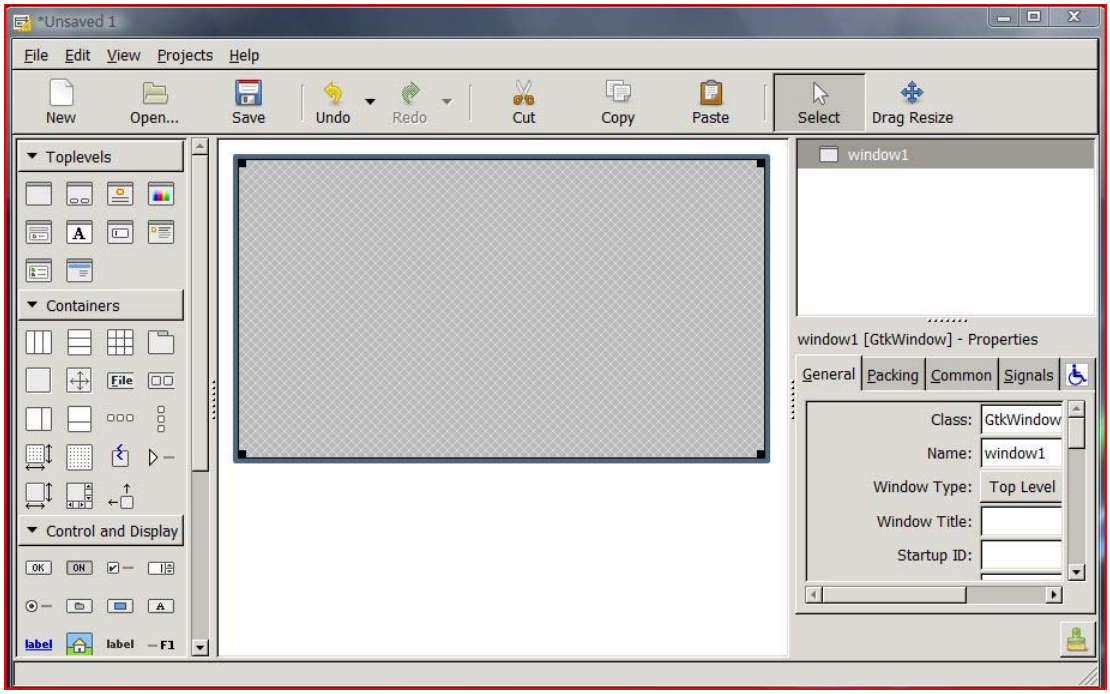
Getting Familiar with the Glade Interface

启动Glade3， 让我们来看看其主界面：



左边的是"Palette" 就像是一个图形编辑程序， 可以用它上面的GtkWidgets来设计你的用户界面。中间部分（刚启动时是空白一片）是"Editor" 所见即所得的编辑器。在右边， 上部是"Inspector"， 下部是widget "Properties" 。Inspector以树形显示当前创建的控件的布局， 可以对控件进行选择。我们通过 Properties中各项内容来设置widgets的属性， 包括设置widgets的信号回调函数。

我们先创建一个顶层窗口并保存。点击Palette上"Toplevels"分组框中的 GtkWindow图标， 你会看到一个灰色窗口出现在Glade中间的 Editor 区域。这是GtkWindow的工作区：



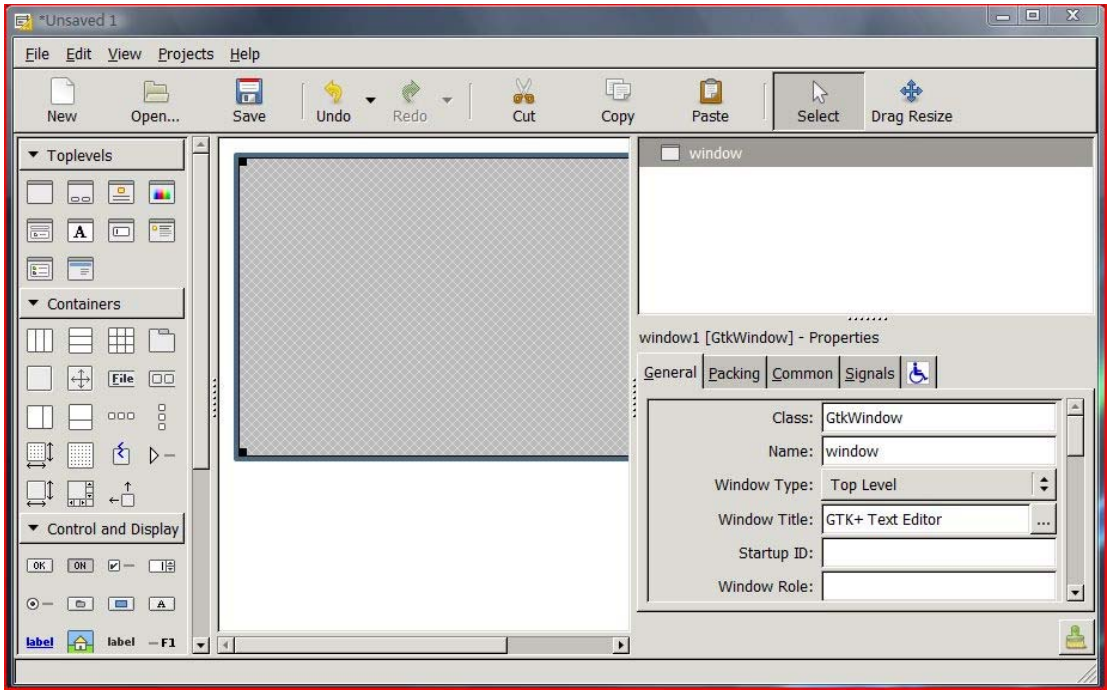
窗口管理器（如GNOME）会自动加上窗口标题， 关闭按钮等， 因此我们编辑时看不见。使用Glade时， 我们总是需要首先创建一个顶层窗口，典型的是创建一个GtkWindow。
以 "tutorial.glade" 文件名保存工程。这个文件是一个XML文件， 你可以在文本编辑器中打开它：

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE glade-interface SYSTEM "glade-2.0.dtd">
<!--Generated with glade3 3.4.0 on Tue Nov 20 14:05:37 2007 -->
<glade-interface>
  <widget class="GtkWindow" id="window1">
    <property name="events">GDK_POINTER_MOTION_MASK | GDK_POINTER_MOTION_HINT_MASK |
GDK_BUTTON_PRESS_MASK | GDK_BUTTON_RELEASE_MASK</property>
    <child>
      <placeholder/>
    </child>
  </widget>
</glade-interface>
```

你看，这就是一个简单的XML文件， 在part2 中我们会用C语言调用Libglade库来解析这个XML文件并在运行时生成 UI 。XML文件很容易用Python应用程序或其它任何语言来解析。Glade能在修改过程中自动保存到该文件。退出文本编辑器，回到Glade我们继续。

Manipulating Widget Properties

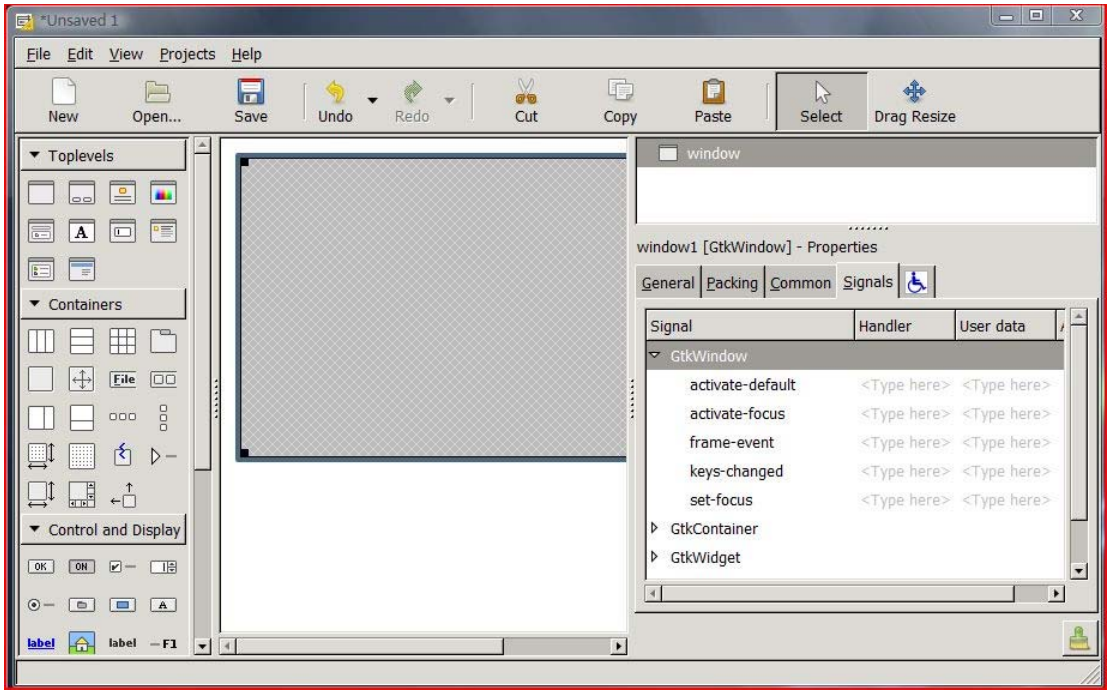
现在，Glade的Editor区显示的是一个空的 GtkWindow widget。我们来修改它的属性。在Properties面板，你会看到4个选项卡：'General', 'Packing', 'Common', 和 'Signals'。我们先来谈谈前面的两个。GtkWidgets有许多属性，这些属性定义了它们的功能和现实方式。
如果你查阅一下GTK+的开发参考文档， 找到GtkWidget的"Properties"一项， 列出了GtkWindow的特有属性， 这些在Glade属性面板的"General"选项卡中， 并且每个widget的属性都会不一样。widget属性名称是我们的应用程序直接获取的信息， 把此GtkWindow的"name"由"window1" 修改为"window"。添加"GTK+ Text Editor"到"Window Title"属性：



我们稍后讲述 "Packing"， 先来看看"Common"， 这里也包括属性设置， 不过我们不能在开发人员参考文档中相应的widget属性下看到它们，这是因为这些属性是继承自父类的属性。在参考文档的"Object Hierarchy"里你将会看到GtkWindow的父类GtkContainer， 连接到GtkContainer属性项你将会看到一个"border-width"， 而在Glade的属性面板中GtkWindow继承了这个属性， 你可以在"Common"选项卡底部找到。我们以后会讲到GtkContainer， 到这里， 你应该清楚地知道对象继承链是多么重要了。因为大部分widgets都从GtkContainer继承， 因此Glade把它的属性放到了"Common" Tab下。
参考文档的"Object Hierarchy"， GtkContainer由GtkWidget继承。链接到GtkWidget， 你会看到其大部分的属性都列在了Glade属性面板的"Common" tab中。这些属性是所有GTK+ widgets的公共属性， 因为它们都继承自GtkWidget。

Specifying Callback Functions for Signals

当某些对程序员有意义的事情发生时， 控件对象就发出一个信号"signal"。这同Visual Basic中的"events"类似。当用户与界面进行交互时， 界面元素发出相应的信号， 程序员可以决定哪些信号需要捕获并连接到一个回调函数，完成某些任务。
我们遇到的第一个信号，也是你在所有GTK+应用程序中都会碰到的，是由GtkObject发出的"destroy"信号。当一个GtkObject对象销毁时就发出"destroy"信号。这非常重要，因为当用户通过点击一个widget顶部的"X"来关闭时， widget就销毁了。我们需要捕获这个信号并正确地退出我们的应用程序。在我们正式为此GUI写代码时做这件事是最好的， 不过先得在Glade中指定响应"destroy"信号的具体函数。
切换到属性面板的"Signals"tab，你将看到一个树形列表，显示了当前widget及其父类对象的所有信号。 这些与参考文档相符。
在"Handler"列下点击灰色文本"<Type here>"并开始编辑它， 从下拉列表框中选择"on_window_destroy"并按回车键。我们也可以键入任何名字， 不过Glade提供的下拉框列出了通用的回调函数习惯命名。这个值如何使用得看程序员如何连接信号与回调函数。在这里， 我们把GtkWindow的"destroy"信号连接到"on_window_destroy"函数名上。在part2中我们会看到这一点的。



到这里， 我们有了一个GUI， 可以编写代码显示我们的空窗口并在点击了关闭按钮时退出程序， 你可以用C， Python或任何其它语言。在此向导中， 我将会充分地向你展示如何在编写任何代码前就用Glade3建立 起完整的GUI. 不过， 为了满足你的好奇心， 同时也让你了解到要实现这个Glade用户接口，代码将会是多么的简单，请看代码：

In C

```
/*
First run tutorial.glade through gtk-builder-convert with this command:
  gtk-builder-convert tutorial.glade tutorial.xml

Then save this file as main.c and compile it using this command
(those are backticks, not single quotes):
  gcc -Wall -g -o tutorial main.c `pkg-config --cflags --libs gtk+-2.0` -export-dynamic

Then execute it using:
  ./tutorial
*/
#include <gtk/gtk.h>

void
on_window_destroy (GtkObject *object, gpointer user_data)
{
    gtk_main_quit ();
}

int
main (int argc, char *argv[])
{
    GtkBuilder      *builder;
    GtkWidget       *window;

    gtk_init (&argc, &argv);

    builder = gtk_builder_new ();
    gtk_builder_add_from_file (builder, "tutorial.xml", NULL);
    window = GTK_WIDGET (gtk_builder_get_object (builder, "window"));
    gtk_builder_connect_signals (builder, NULL);

    g_object_unref (G_OBJECT (builder));

    gtk_widget_show (window);
    gtk_main ();

    return 0;
```

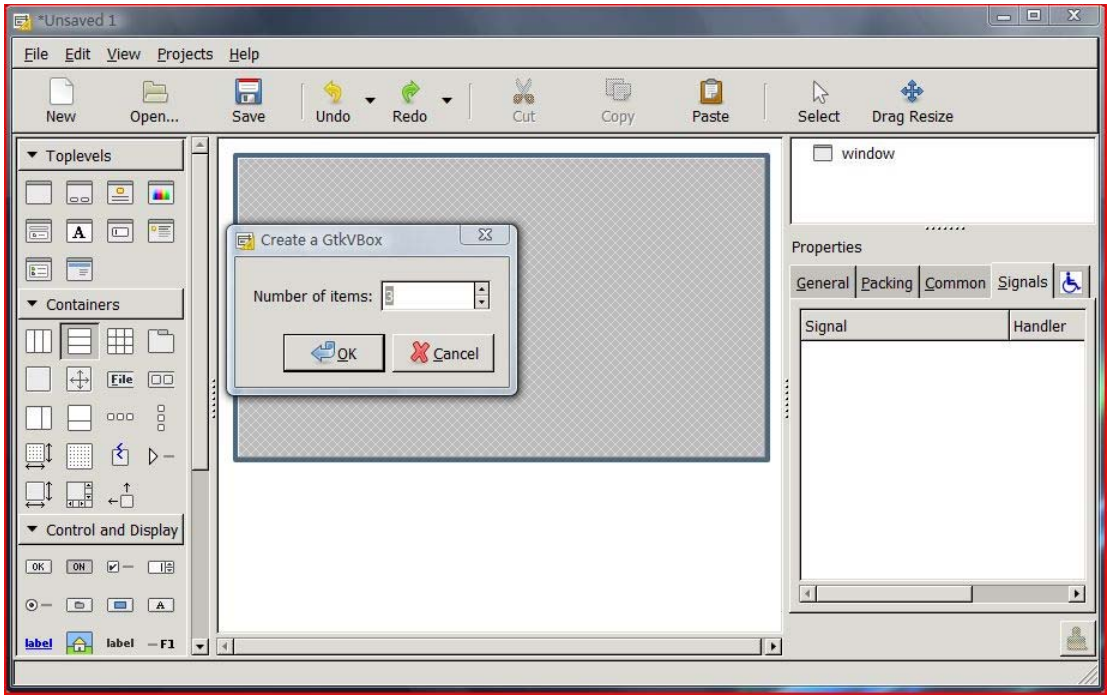
```
}

In Python (note: you must set the 'visible' property of  'window' to "Yes" in the 'Common'
properties tab in Glade)
#!/usr/bin/env python

# First run tutorial.glade through gtk-builder-convert with this command:
# gtk-builder-convert tutorial.glade tutorial.xml
# Then save this file as tutorial.py and make it executable using this command:
# chmod a+x tutorial.py
# And execute it:
# ./tutorial.py

import pygtk
pygtk.require("2.0")
import gtk

class TutorialApp(object):
  def __init__(self):
      builder = gtk.Builder()
      builder.add_from_file("tutorial.xml")
      builder.connect_signals({ "on_window_destroy" : gtk.main_quit })
      self.window = builder.get_object("window")
      self.window.show()

if __name__ == "__main__":
  app = TutorialApp()
  gtk.main()
```

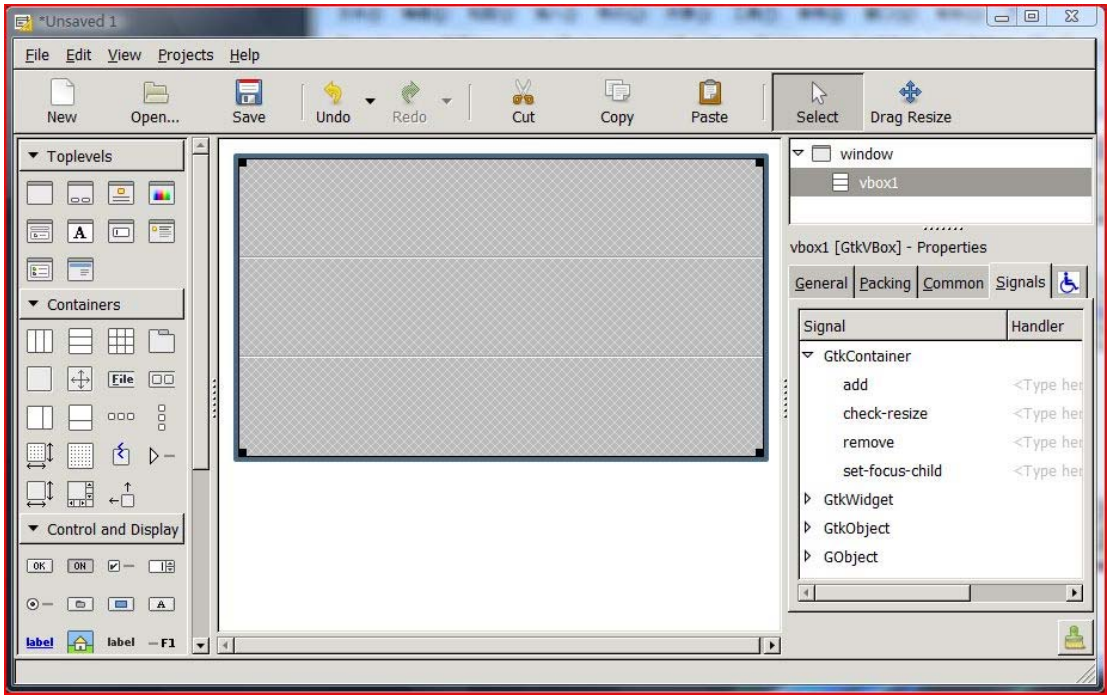
在这部分，我将不会深入讲解这些实现代码， 而把注意力放在Glade3上。不过你已经看到了， 实现一个Glade创建的用户接口是多么的简单。

Adding Widgets to the GtkWindow

查阅参考文档你会看到GtkWindow继承自GtkContainer。继承自GtkContainer的widgets就是一个容器widgets， 也就是说它们可以容纳其它的widgets。这是GTK+编程的一个基本理念。如果你是一个Windows程序员，你会期望着拖一堆的widgets到窗口上，并摆放好它们的位置即可。不过GTK+并不是这样工作地----有更好的理由。
GTK+的widgets可以装填到不同的容器， 容器能装填到其它的容器中。有不同的装填属性设置可以控制widgets在容器内如何分配空间，这样我们就可以创建出十分复杂的GUI界面， 而不用写任何代码来调整widgets大小尺寸和位置。因为GTK+为我们做了这一切。
不过这对于一个GTK+程序员新手来说也许是一个难以理解的概念，让我们用事实来说话！

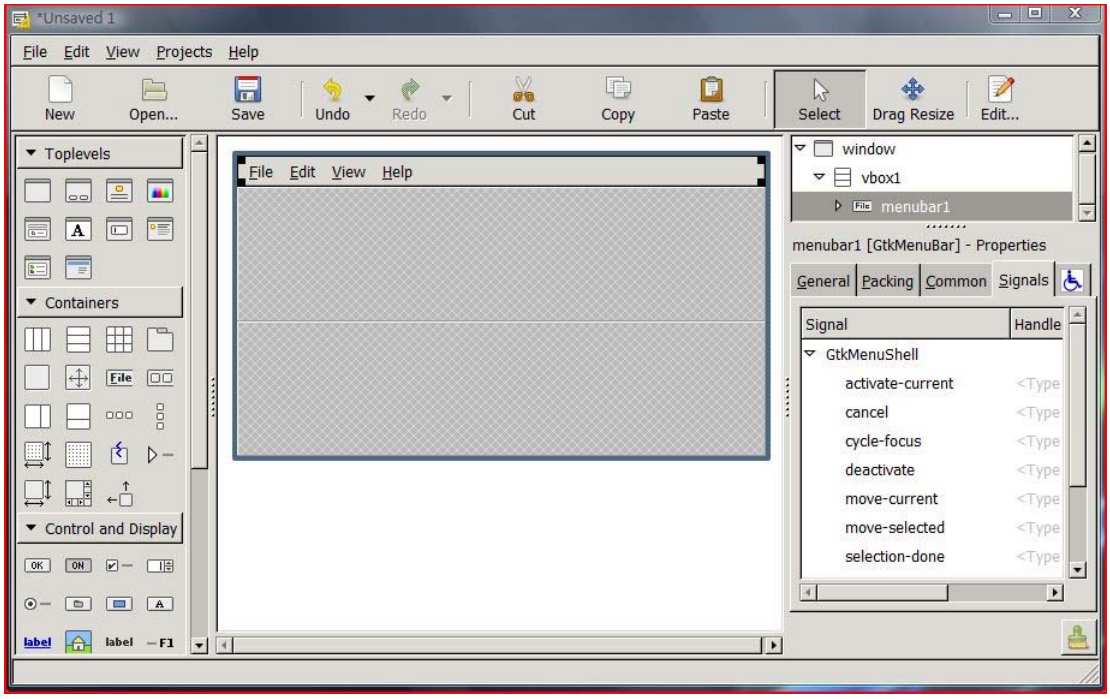
GtkWindow继承自GtkBin容器，GtkBin是只能容纳一个子widget的容器，但是我们的界面需要3个元素：菜单条， 文本编辑区， 状态栏。因此， 我们使用GtkVBox，它可以容纳一个以上的子widgets， 并按照垂直排列。（GtkHBox按照水平排列子widgets）注：这里的"子widgets"是指容器中容纳的属于此容器的widgets
在 Palette面板上"Container"分组框下的 GtkVBox 图标上点击。此时"Select"工具条按钮弹起，并且鼠标在 Glade编辑区上显示为带有 "+" 的 GtkVBox 图标。在灰色的空窗口区点击，就放置了一个GtkVBox，此时弹出一个对话框询问"Number of items"， 设置GtkVBox的行数，我们选择3行。



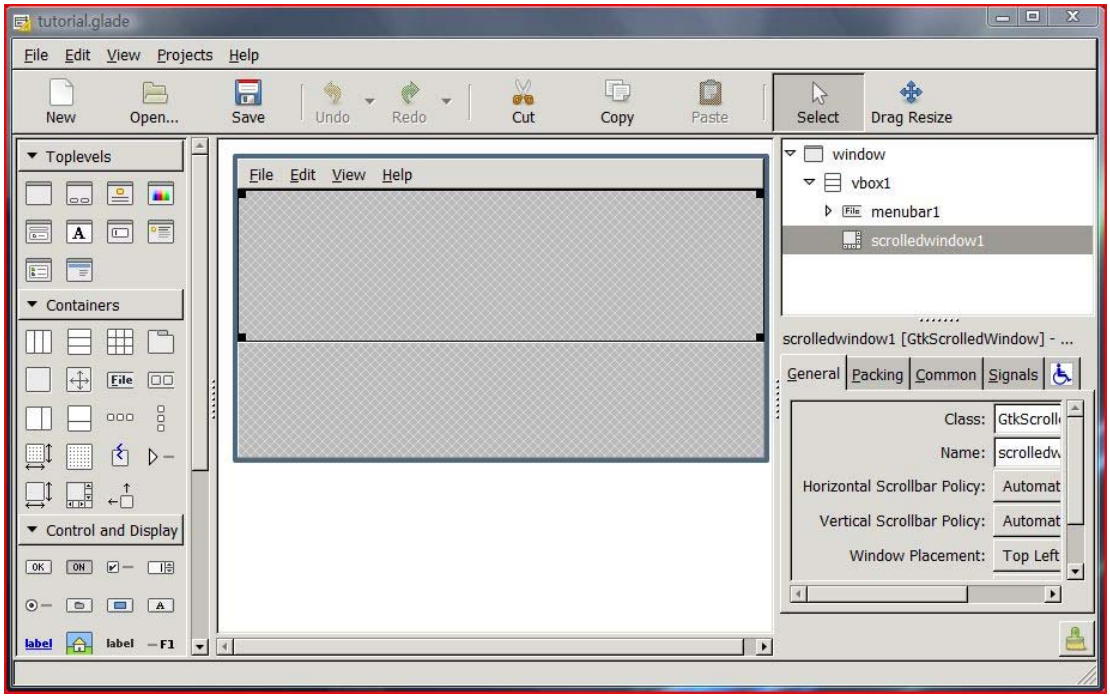


编辑区的GtkWindow现在有三行。Glade窗口顶部的"Select"工具栏图标转换到按下状态，即允许你在编辑区选择任意的widgets。

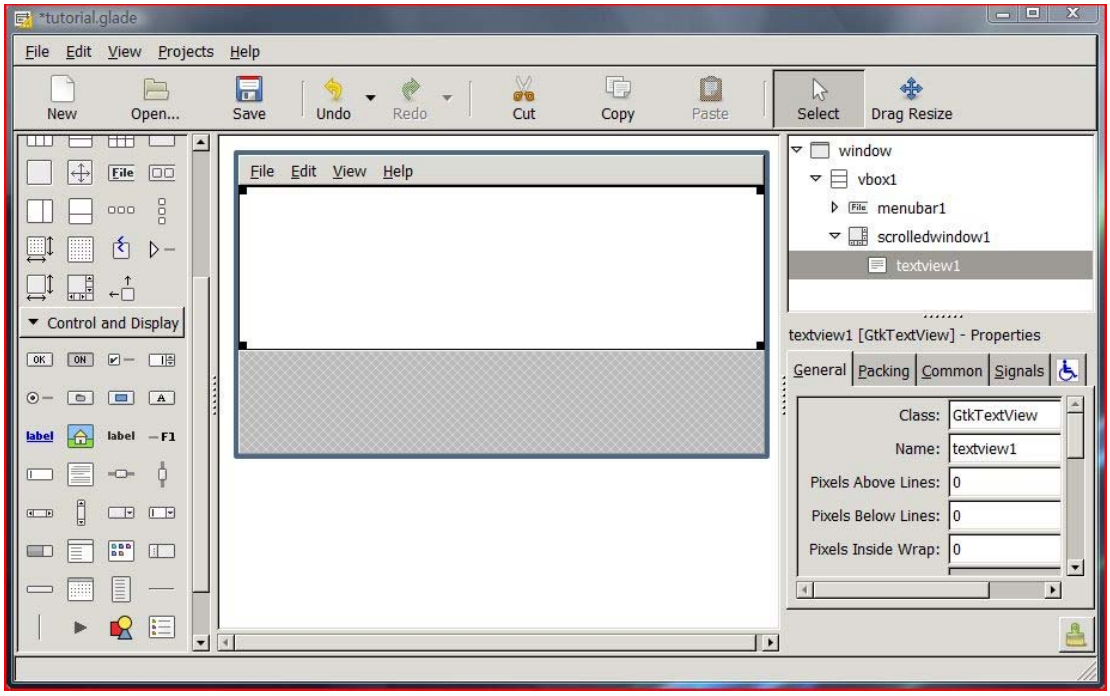
接下来添加一个GtkMenuBar到GtkVBox的最顶上一行，GtkMeneBar在Glade的"Container"分组框下



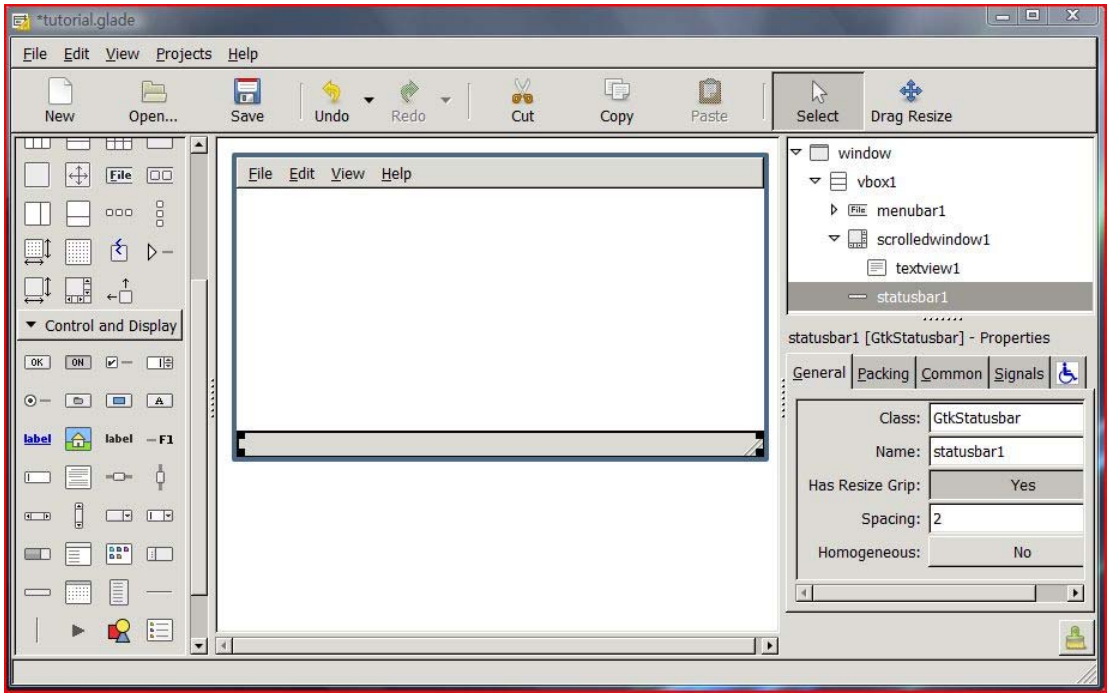
现在，找到"Container"下的GtkScrolledWindow并添加到中间一行。完成这一步后除了中间一行被选中状态外看不出有什么变化。这是因为GtkScrolledWindow没有任何初始化可视元素。它仅仅是一个容器，当它容纳的子widgets变得太大时它提供滚动条。我们的编辑器需要滚动条支持。



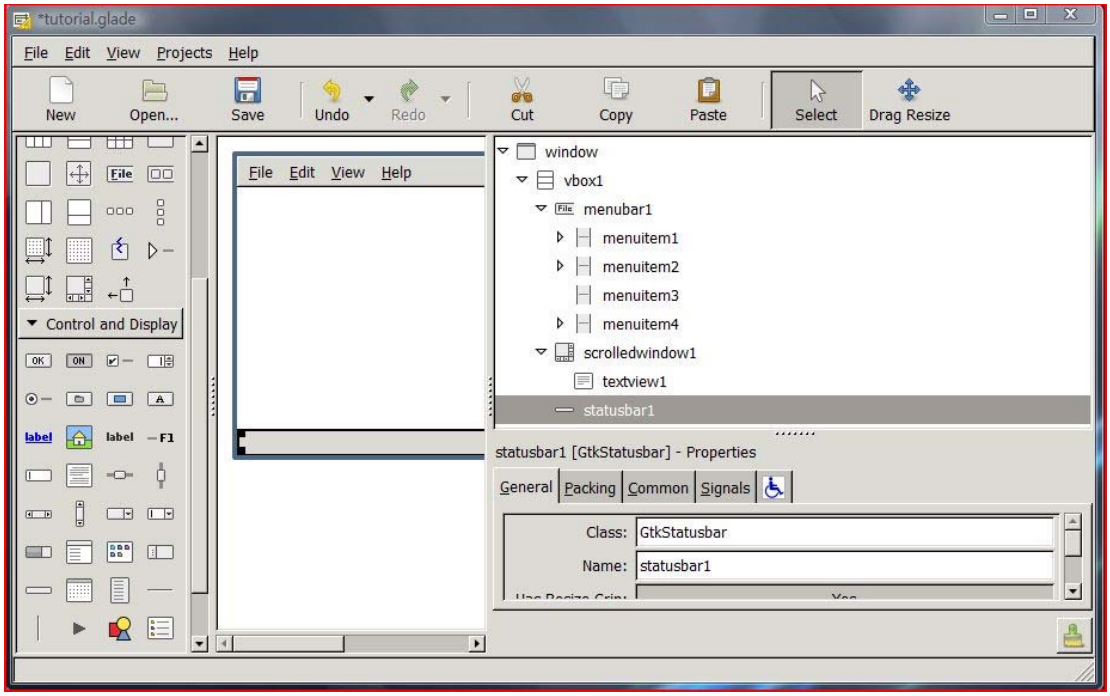
点击"Control and Display"分组框下的GtkTextView并添加到GtkScrolledWindow上（中间一行）。



最后，点击"Control and Display"分组框下的GtkStatusbar并添加到最底部一行。



这就建好了我们文本编辑器的UI布局。在Inspector中你会看到widgets的包容关系。



在Inspector中选择widgets是很方便的，因为当widgets相互重叠时你在编辑区不能选择了。比如你不能在编辑区中点击GtkScrolledWindow因为我们只能看到它包容的子widgets， 你只能在Inspector中选择。

之前我提到装填的概念对一个GTK+程序员新手来说不好理解。因此，我将向你展示不同的装填方式是如何影响你的布局设计的。

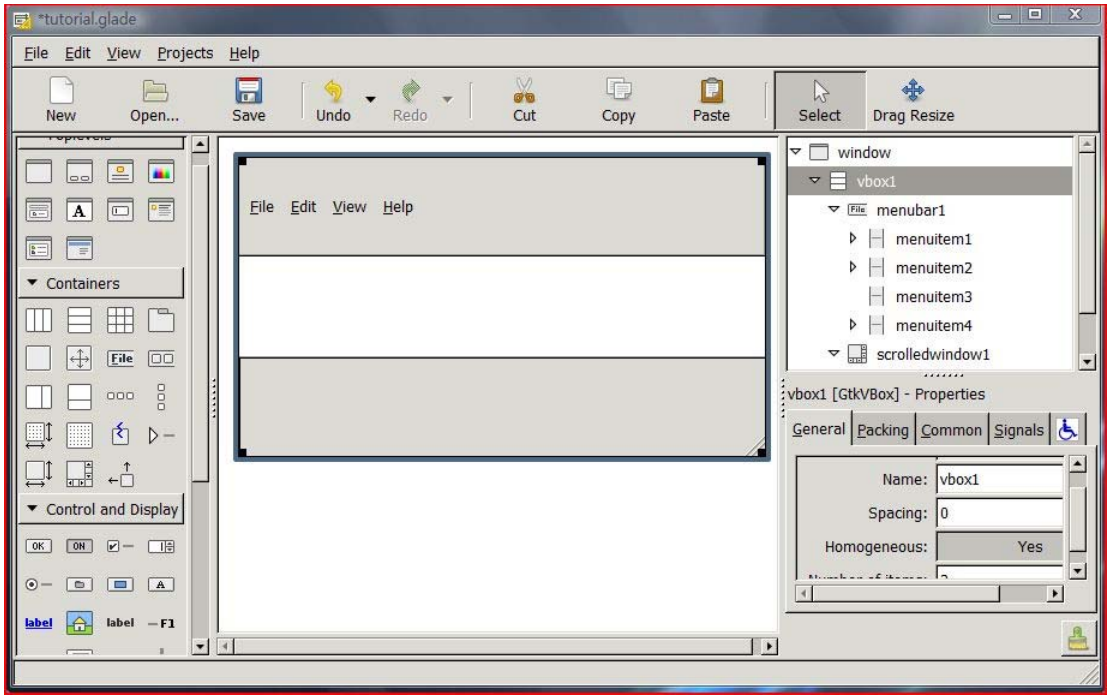
How Packing Effects the Layout

从上面的界面设计过程，你也许会惊叹Glade如此的智能。它是如何知道我们不想状态栏太高？如果你调整窗口大小， 它又是如何知道应该让文本编辑框自动缩放来填充窗口变化的空间？哈哈， Glade靠猜的！它应用了默认设置，我们通常需要如此，不过不总是这样。
了解装填的最好方式是试验各种不同的装填属性，观察Glade如何响应。
你应该了解的关于装填和空间分配：
homogeneous：此属性设置，则告诉GTK+为每个子widgets分配同样大小的空间。
expand：子widgets的装填属性，确定在容器增长时，此子widgets是否获得额外的空间。
fill：子widgets装填属性，确定子widgets是否利用分配到的额外空间。

来看默认装填属性，GtkScrolledWindow的"expand"=TRUE表示当其所在容器增长时它要获得空间分配，"fill"=TRUE表示它将扩充自己来利用额外空间。这是我们想要的效果。

Widget	Property	Value
GtkVBox "vbox1"	homogeneous	FALSE
GtkMenuBar "menubar1"	expand	FALSE
	fill	TRUE
GtkScrolledWindow "scrolledwindow1"	expand	TRUE
	fill	TRUE
GtkStatusbar "statusbar1"	expand	FALSE
	fill	TRUE

现在我们来看homogeneous都干嘛了。在Inspector中选择GtkVBox并设置其"Homogeneous"="Yes"，这表示"vbox1"将分配同样大小的空间给其包含的子widgets。既然其3个子widgets的"fill"=TRUE，那么它们将填满分配到的空间。

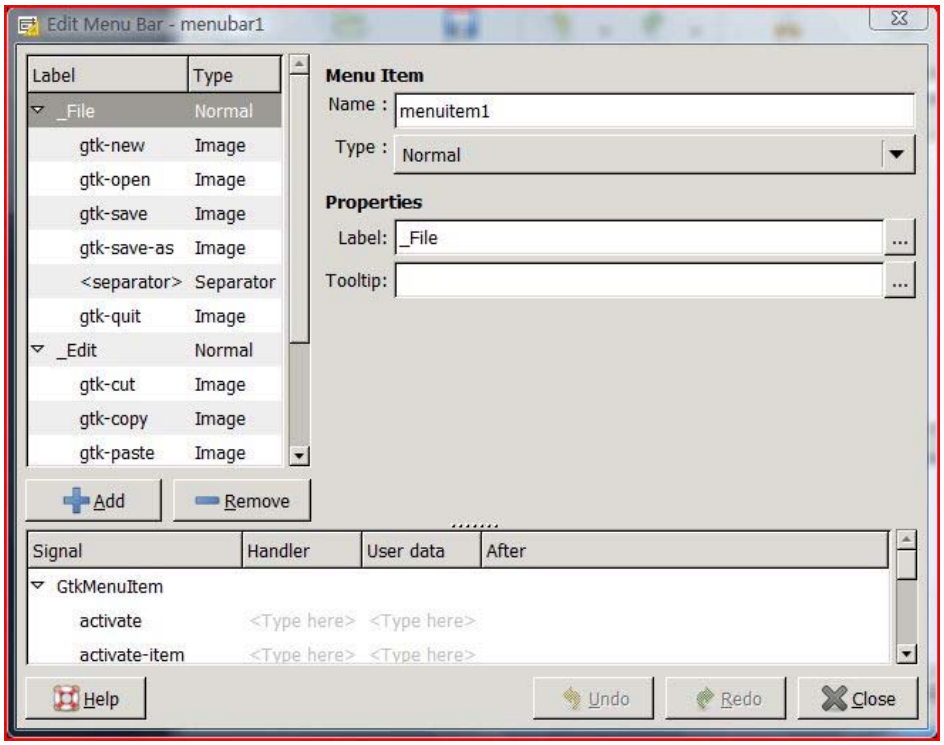


现在设置GtkScrolledWindow的"Expand"=Yes， "Fill"=NO。此时额外空间分配给GtkScrolledWindow，但是它并不扩充自己来利用分配到的额外空间。

我知道你认为这看起来很奇特，不过随着你深入了解Glade， 你将会清楚地知道装填属性是怎么工作的，并且你会惊叹为了改变GUI各元素位置和尺寸竟然只需要做如此少的工作就能完成。

Editing the Menu (or Toolbar)

Glade3拥有一个新的菜单条和工具条编辑器，尽管我们的向导里不使用GtkToolBar，不过它的处理与GtkMenuBar很相似。我们将利用Glade3的菜单编辑器来删除不需要的菜单项并对需要使用的菜单项设置信号处理。
你可以使用Glade的Inspector来删除，可以在Glade的属性面板中设置信号处理，不过Glade3的菜单编辑器更容易做这些工作。
在编辑区或通过Inspector选择GtkMenuBar，单击右键选择"Edit"启动菜单编辑器。



菜单编辑器包括的属性设置很像Glade主界面属性面板，而底部的信号设置与Glade属性面板中的"Signals"tab很像。只不过菜单编辑器中左边是树形列表显示。可以很轻易地添加或删除菜单项。移除"_View"菜单项。我们的向导中不使用这个菜单。对剩余的菜单项，我们需要重新命名，以便于在源代码中能清楚明了地引用它们。每个菜单项修改都一样，所以我只讲"New"，记住，所有这些菜单编辑器能做的工作在Inspector和属性面板中也能完成。

Final Touches to the Main Window

对于"textview1"， "textview2"这种命名的引用十分不利于在源代码中使用，因此需要重新修改以下widgets的名称（记住，名称在属性面板的"General"tab）
1 "textview1"改为"text_view"
2 "statusbar1"改为"statusbar"

为了使其看上去更漂亮一些，我们为GtkScrolledWindow增加阴影和边框
1 在属性面板的"General"tab中把scrolledwindow1的"Shadow Type"改为"Etched in"
2 属性面板的"Common"tab中把scrolledwindow1的"Border Width"改为1
3 在"General"属性中把"text_view"的"Left Margin"设为2
4 在"General"属性中把"text_view"的"Right Margin"设为2

Getting Additional Help Using Glade

对使用Glade过程中的更多问题，可以询问
glade-users mailing list 或
GTK+ Forums.

What Next?

在 GTK+ and Glade3 GUI Programming Tutorial - Part 2 中将选择一个确定的编程语言来实现我们刚刚创建的GUI。

GTK+ and Glade3 GUI Programming Tutorial - Part 2

2009年3月28日
20:35
原文地址链接 www.micahcarrick.com/12-24-2007/gtk-glade-tutorial-part-2.html
作者 Micah Carrick
翻译：Binn.X Wee
博客链接：http://blog.csdn.net/xbwee
本文链接：http://blog.csdn.net/xbwee/archive/2009/03/28/4032815.aspx

Choosing a Programming Language for GTK+ Development

在part1 我们用Glade设计了一个GTK+ 文本编辑器应用程序的图形用户界面（GUI），并创建了一个XML文件 tutorial.glade 这是一个描述GUI的XML文件。
在这一部分，我将讨论可以用来进行 GTK+ 开发的编程语言。本向导系列都将只包含 C 编程和Python编程。

Which is the BEST Language?

先放下操作系统不说。 这个问题本身就是一个错误的问题， 因为对于不同的人，在不同的时期，处于不同的环境，答案都是不一样的。每一种编程语言都有它的有点也同时存在缺点。应该这样来问：什么语言最适合我现在开发的项目？

Language Choice Considerations

选择一门编程语言时最重要的一点是对其它的语言不抱任何成见。你可以选择 X 语言来进行GTK+编程，后来又换用Y语言因为你知道它更适合你的任务。而GTK+的概念对不同的语言来说却都是一样的。
1 经验水平
你将在编程上花费的时间，热情投入的多少以及耐心程度等等是你选择一门编程语言最重要的因素。没有任何编程经验的人需要学习一门语言的基本概念，如语法和特性等，而一个经验丰富的编程人员可以通过比较来快速掌握一门新语言，并更多地关注语言的难点。此外如果你是一个PHP专家，那么用PHP进行GTK+开发是最适合你的。也许你在大学学习过C++并打算用它。也许你一直用Visual Basic 但是打算尝试用C 。
2 相关活动和社区支持
GTK+是用C语言开发的，并且提供了其它多种语言的绑定版本。各绑定语言的开发项目的进展情况是一个重要的因素。你想要选择一个最新发布版本的语言，并且修正了bug（我提到的所有语言都是最新的）。此外众多的使用者和大的社区也很重要，你可以从那获得更多的支持。使用某确定语言来开发GTK+的人越多， 可获取的信息就越多。
3 有效率的程序员 vs 有效率的编程
易于编程开发， 速度方面的效率，可以对底层进行访问控制，这些方面存在一个权衡。对大多数程序来说， 两个编程语言的效率常常被忽视，甚至一个程序员新手根本就不会注意到这一点。因此， 对生产率的提高就成了决定性因素。例如， 如果我需要写一个程序，允许我通过一个GUI来与命令行交互，那么我会选择Python或者Ruby。然而， 如果我要开发一个复杂的，高性能的IDE， 我会选择C或者C++。事实上， 你可以在一个工程中使用不同的开发语言。你可以用C活C++编写内存和处理器密集型实例，而用Python或者Ruby来开发其它的部分。
4 语言魅力
是的， 对开发语言的表现形式和个人的感受也是一个因素。语言整体的开发流程如何对你更重要。这影响着你的思维方式。

A Look at Python vs. C

基于以上的标准， 在我们的向导中，我将分别选择两种语言来讲解。我选择最适合以上标准的C 和Python。它们都具有强大的开发团队和社区支持， 并且在开发Linux和GNOME中用的最多。此外， 对于高效率和生产率来说，它们是两个极端。在接下来的部分，你可以分别尝试它们并自己做个比较。
如果你没哟任何编程经验， 或者仅有一点Visual Basic 或PHP的经验，我建议你看Python。即使你有C ， C++或者JAVA的经验，也许你还想学习一点Python的只是，它是一门令人兴奋地现代编程语言， 更多的乐趣， 令人惊奇的易学易用。对于Linux下RAD开发， Glade + Python是一个强组合。在
www.pygtk.org. 上可以学习到更多 Python 进行GTK+编程的绑定PyGTK。
如果你是一个经验丰富的程序员或者是一个学生， 那么很值得你去学习C或者C++进行GTK+编程开发。学习C语言的GTK+ 可以很轻松地转到其它语言上比如Python。此外， 你可以有更多选择加入已有的项目开发中。我个人是使用C来开发大多数GTK+项目的，尽管可能需要更多的时间。

What's Next ?
在part3， 我将会谈到如何设置你的开发环境并分别用C 和Python来实现我们在part1中创建的Glade GUI。

GTK+ and Glade3 GUI Programming Tutorial - Part 3

2009年3月29日
13:43
原文地址链接 www.micahcarrick.com/12-24-2007/gtk-glade-tutorial-part-3.html
作者 Micah Carrick
翻译：Binn.X Wee
博客链接：http://blog.csdn.net/xbwee
本文链接：http://blog.csdn.net/xbwee/archive/2009/03/29/4034314.aspx

Writing a Basic Program to Implement the Glade File

在这一部分， 我将示范一个非常简单的程序， 用来解析我们在part1中用Glade3创建的GUI文件tutorial.glade， 并显示我们GTK+ 文本编辑器的主窗口。我会先讨论GTK+ 的一些概念然后分别以C语言和Python语言来给出实现代码。如果你选择其中之一， 那么你可以跳过另外一门语言的介绍部分。

Setting Up Your Development Environment

为完成这部分向导的GTK+编程，你需要一个文本编辑器， 一个终端，， GTK+开发库，以及Devhelp。Devhelp是开发人员的参考帮助。如果你是Linux新手，那么有更多的选择等着你。并没有一个所谓标准的编辑器或者是IDE， 许多开发人员都仅仅使用自己喜欢的文本编辑器和终端。尽管也有一些功能齐全的IDE可以选择， 不过你应该首先使用比较简单的文本编辑器加上终端即可， 以免被IDE的特性和自动化功能弄迷糊， 浪费不必要的时间。
我使用gedit， GNOME的默认文本编辑器。gedit有一些有用的插件可以使用。我写了一个 Gedit Symbol Browser Plugin 可以让你快速地跳转到源代码的函数定义。

你需要的开发库取决于你使用C还是使用Python， 对于不同的开发平台及发布平台会有很大不同， 不过我会对此提供一些有用的信息。如果在安装开发包和发布产品时还出现问题的话， 可以到 GTK+ Forums 寻找答案。

在Linux上， 你通常可以使用分发包管理器获得你所需要的所有开发包并解决其依赖性。例如， 在Ubuntu上你只需要使用命令：sudo aptitude install libgtk-2.0-dev， 即可安装好GTK+开发包，以及其相关依赖项。
注意，需要安装的是"development package"， Ubuntu/Debian中以-dev结尾的包， 在Redhat/Fedora中以-devel结尾。这种包包括了使用相应的库来开发应用程序所需要的头文件和其它包含文件。请记住，"package"允许你能过运行应用程序，而"package-dev"或"package-devel"允许你使用库开发应用程序。
另外你还会见到-doc后缀的包， 比如"libgtk2.0-doc"，这是相应库的开发文档， 安装之后你就可以使用 Devhelp--the GNOME developer's help browser. 来浏览相关文档。

如果你使用C，你应该安装以下开发包及其依赖项:build-essential, libgtk2.0-dev, libgtk2.0-doc, libglib2.0-doc, devhelp.
如果你用Python， 你应该安装以下开发包及其依赖项：python2.5-dev, python2.5-doc, python2.5-gtk2, python-gtk2-doc, python-gobject-doc, devhelp.

GtkBuilder and LibGlade

如果你能回忆起来的话， 我们在part1利用Glade创建的tutorial.glade文件是一个描述GUI界面的XML文件。实际的GUI是由我们的应用程序来创建的。因此， 应用程序需要解析XML文件并创建widgets的对象实例。不过这个任务可以使用两个库来完成：

Libglade库， 用来解析glade文件并创建widgets对象实例。使用Libglade库是最常用的方式， 在其它一些开发向导或教程中都能看到。然而， 自此GTK+2.12， 就包含了一个叫GtkBuilder的对象， 它是GTK+自身的一部分并用来取代Libglade库。也因此，在我们的开发向导中我们将使用GtkBuilder。不过， 你得知道的是， 你在网络上看到的教程中凡是使用Libglade库的， 都可以使用GtkBuilder来代替。

在写本系列的时候， GtkBuilder还是一个新东西， 因此Glade尚未支持GtkBuilder格式文档，（译者注：自Glade3.6.0版本， 提供了对两种格式的支持即Libglade和GtkBuilder格式）GtkBuilder格式就是XML格式文件， 但是有一点不同。这意味着GtkBuilder使用glade文件时， 需要进行格式转换。GTK2.12 提供了转换脚本命令来做这件事， 这个脚本连同开发库一起安装。现在你已经装上了^_^
你可以参考关于Libglade/GtkBuilder常见问题的答案：Libglade to GtkBuilder F.A.Q..

使用如下命令来把Libglade格式文件tutorial.glade转换为GtkBuilder格式文件tutorial.xml文件：gtk-builder-convert tutorial.glade tutorial.xml
而tutorial.xml文件是我们的应用程序将要用来进行解析的， 不过我们仍然需要glade文件以便可以使用Glade进行修改。这是必需的，直到Glade3.6版本支持GtkBuilder格式文件为止。（译者注：Mar 16 2009，Glade3.6.0 released 。此版本及以后版本可以直接使用xml文件并且自动保存所做的修改）

The Minimal Application

现在我们开始写代码！首先回顾一下到目前为止我们都做了哪些工作
1 使用Glade，创建了描述用户界面的tutorial.glade文件
2 我们选择了开发语言C和Python
3 准备了一个文本编辑器和一个终端
4 安装了进行GTK+开发所需要的所有库
5 使用gtk-builder-convert命令把tutorial.glade文件转换成了GtkBuilder使用的tutorial.xml文件。
现在， 在深入讲解每一行代码的细节之前， 先来写一个最简单的程序来验证一切都能正常工作， 并熟悉开发流程。因此， 打开你的文本编辑器， 输入以下内容：
C语言

```
#include <gtk/gtk.h>

void
on_window_destroy (GtkObject *object, gpointer user_data)
{
        gtk_main_quit();
}

int
main (int argc, char *argv[])
{
        GtkBuilder              *builder;
        GtkWidget               *window;

        gtk_init (&argc, &argv);

        builder = gtk_builder_new ();
        gtk_builder_add_from_file (builder, "tutorial.xml", NULL);

        window = GTK_WIDGET (gtk_builder_get_object (builder, "window"));
        gtk_builder_connect_signals (builder, NULL);
        g_object_unref (G_OBJECT (builder));
```

```
        gtk_widget_show (window);
        gtk_main ();

        return 0;
}
```
命名为main.c并保存到tutorial.xml所在目录。

Python语言
```
import sys
import gtk

class TutorialTextEditor:

    def on_window_destroy(self, widget, data=None):
        gtk.main_quit()

    def __init__(self):

        builder = gtk.Builder()
        builder.add_from_file("tutorial.xml")

        self.window = builder.get_object("window")
        builder.connect_signals(self)

if __name__ == "__main__":
    editor = TutorialTextEditor()
    editor.window.show()
    gtk.main()
```
命名为tutorial.py并保存到tutorial.xml所在目录。

Compiling and Running the Application

C语言

C是一种编译语言， 需要使用gcc编译器把源代码转换为二进制可执行代码。为了让gcc知道GTK+链接库位置以及编译标识，我们使用pkg-config。当我们安装GTK+开发包时， 一个叫"gtk+-2.0.pc"的配置文件也安装了， 它告诉pkg-config我们系统上安装的GTK+库版本以及包含文件位置等信息。

输入以下命令； pkg-config --modversion gtk+-2.0

终端输出将是你安装的GTK+版本号。我的系统上显示为2.12.0 。现在来看编译GTK+应用程序时需要的编译器标识： pkg-config --cflags gtk+-2.0

输出将是一堆的-I开关选项指出编译器使用的包含文件。这能让gcc知道到哪去找我们应用程序中"#include"所列出的包含文件。

每当使用了"#include"并引用了非标准C库头文件时， 都需要使用"I/path/to/library"选项传给gcc。这些库可以装在不同的地方，这根据分发要求，操作系统或使用者意愿来定。而pkg-config为我们掌控这一切。

编译我们的程序。终端输入以下命令（确保你当前目录为main.c和tutorial.xml所在目录）：

```
gcc -Wall -g -o tutorial main.c -export-dynamic `pkg-config --cflags --libs gtk+-2.0`
```

其中"-Wall"选项告诉gcc显示警告信息。"-g"选项产生调试信息， 当你使用调试器如gdb进行单步调试时这非常有用。"-o tutorial"告诉gcc输出的可执行文件名。"main.c"是gcc将对其进行编译的源文件。"-export-dynamic"关系到我们如何连接信号与回调函数，这在以后讲解。最后出现的是pkg-config命令。

注意， pkg-config命令是用反引号``而不是单引号''。反引号在数字键1的旁边。这告诉shell先执行pkg-config --cflags --libs gtk+-2.0然后将其结果输出复制到gcc命令的末尾。pkg-config命令可以在任何系统上使用，而不用考虑库的安装位置。

编译之后， 我们可以运行它：./tutorial

你会看到一些警告信息"Gtk-WARNING**：Could not find signal handler 'xxxxxx'"， 别担心，这些信息告诉我们在glade文件中定义的一些信号在程序中没有相关的处理程序。讲到代码时我在讲解这些。不过你应该看到显示了一个窗口，通过点击"X"来关闭它。

如果由于某种原因你不能编译并执行你的程序，可以把错误信息贴到 GTK+ Forums. 以寻求帮助。
Python语言
因为Python是解释性语言，所以我们不需要编译程序。只是调用Python解释器， 源代码中第一行就是做这件事情。为了运行我们的程序， 使用以下命令改变文件访问权限：
chmod a+x tutorial.py
然后可运行：./tutorial.py

Stepping Through the Code

注意：你应该在我讲到某个函数时顺便查阅GTK+开发文档。它会是你最好的朋友。安装Devhelp并使用它。考虑到也许你不能安装Devhelp的情况，在我讲到一个新函数时将会提供在线文档的链接。

Including the GTK+ Library

C语言
希望你足够了解了C编程并能知道第一行"#include <gtk/gtk.h>"是怎么回事。否则，你应该先看看C基础编程向导。包含了gtk.h，我们就同时间接地包含了多需的其它头文件。事实上， 我们包含了所有GTK+库及其依赖的GLib库部分头文件。想知道具体有哪些，打开此文件看一看！
Python语言
希望你足够了解了Python编程并能知道前两行"#import sys"和"#import gtk"是怎么回事。否则，你应该先看看Python基础编程向导。现在我们可以访问所有的gtk.x类了。

Initializing the GTK+ Library

Python自动显示初始化了GTK+库， 而C你必须在调用任何GTK+函数前初始化GTK+库
gtk_init(&argc， &argv);

Building the Interface with GtkBuilder

在不使用任何辅助GUI工具开发GTK+应用程序时， 程序员需要编写程序来创建每个widgets并调用相关函数设置其属性，然后装填到容器中。每个步骤都需要许多行代码才能完成， 非常的繁琐。考虑我们在part1中建立的用户GUI界面， 超过20个widgets（包括菜单项）编写代码来创建这些界面需要将近百行代码才能创建并设置好每一个属性。
幸好我们有Glade和GtkBuilder。仅仅只需要2行代码，GtkBuilder就能解析tutorial.xml文件，创建所有定义的widgets并应用其属性，以及建立widgets之间包容父子关系。然后，我们就能利用GtkBuilder引用widgets并控制其行为特性。
C语言
```
builder = gtk_builder_new();
gtk_builder_add_from_file(builder, "tutorial.xml", NULL);
```

第一个变量是在main（）中定义的GtkBuilder类对象指针。我们使用gtk_builder_new()来创建实例。所有的GTK+对象都以这种方式创建。
这时builder还没有任何UI元素，我们使用gtk_builder_add_from_file()来解析XML文件，并把其内容添加到builder对象。此函数的第三个参数我们传递了NULL，因为现在不需要使用GError。我们没有进行任何异常和错误处理，因此一旦有任何异常或错误出现， 我们

的程序只能崩溃。异常与错误处理我们后面再讲。
在调用了gtk_builder_new()创建了对象实例之后，所有的其它gtk_builder_xxx函数都是以创建好的builder对象作为第一个参数。这就是GTK+用C实现的面向对象技术。其它所有GTK+对象都是如此方式。

Python语言
```
builder=gtk.Builder()
builder.add_from_file("tutorial.xml")
```

Getting References to Widgets From GtkBuilder

创建好了所有的widgets之后我们就可以引用它们了。我们只需要引用一部分，因为其它的已经能很好地完成它们的工作不再需要更多的处理了。例如，GtkVBox，容纳了菜单，文本编辑框，状态栏，已经完成了布局工作不需要代码处理了。我们可以在应用程序生命期引用任意一个widgets并保存在变量中以备用。在此开发向导中我们仅仅需要引用命名为"window"的GtkWindow对象，以便显示它。
C语言
```
Window=GTK_WIDGET(gtk_builder_get_object(builder, "window"));
```
首先， gtk_builder_get_object()第一个参数是获取的对象所在的builder对象，第二个参数是获取对象的名称， 这个名称必须与我们在Glade中"name"属性一致。函数返回一个GObject对象指针， 保存在window变量中。参考文档中对象层次结构指出GtkWidget从GObject继承， 因此一个GtkWindow就是一个GObject，也是一个GtkWidget。这是OOP的基本概念，对于GTK+编程很重要。
因此GTK_WIDGET()宏进行类型转换。你可以用转换宏把GTK+的widgets转换为它的任意一个子类型， 所有GTK+类都有相应的类型转换宏。GTK_WIDGET(something)就如同(GtkWidget*)something所起的作用一样。
最后， main（）函数中我们声明一个GtkWidget类型的window变量而不是GtkWidget类型，这纯属是习惯而已。我们也可以把它声明为GtkWindow*也对。所有GTK+的widgets都继承自GtkWidget因此我们可以声明指向任何widgets的指针为GtkWidget类型。许多函数都传递GtkWidget*类型的参数，并且许多函数返回的也是GtkWidget*类型指针。所以声明为GtkWidget类型然后必要时使用转换宏转换为其它widgets类型。
Python语言
```
self.window = builder.get_object("window")
```

Connecting Callback Functions to Signals

在part1我们指定了许多信号的处理函数， 当有事件发生时GTK+就发送相应的信号， 这是GUI编程的基础概念。我们的应用程序需要知道用户何时做了何事，然后对此进行响应。你将会看到，我们的程序就是循环等待事件的发生。我们使用GtkBuilder来连接在Glade中定义的信号和相应的回调函数。GtkBuilder会自动查询程序符号表然后正确地连接信号与处理函数。
在part1中， 我们定义了"on_window_destroy"函数来响应"window"的"destroy"信号。当一个GObject对象销毁时，它会发送"destroy"信号，我们的应用程序就无限循环等待事件发生， 当用户关闭主窗口（点击主窗口标题栏的"X"按钮）应用程序需要能够终止循环并退出。连接一个回调函数到GtkWindow的"destroy"信号，就能知道何时终止程序。因此，"destroy"信号是几乎所有的GTK+应用程序中都会处理的。
注意：本实例中用来连接信号与处理程序的函数与Libglade中的glade_xml_signal_autoconnect()函数等价。
C语言
```
gtk_builder_connect_signals(builder, NULL);
```
此函数总是需要传递builder对象作为第一个参数，第二个参数是用户数据。这很有用，

不过现在设为NULL即可。这个函数会使用GModule， GLib的一部分， 动态加载模块来查询应用程序符号表（函数名， 变量名等等），寻找应用程序中能够与Glade中指定的回调函数名相符的函数，然后连接到信号。
在Glade中我们为GtkWindow的"destroy"信号指定了回调函数名为"on_window_destroy"，因此gtk_builder_connect_signals()会在程序中寻找名为"on_window_destroy"的处理函数，如果找到则连接到"signal"信号。函数原型必须一致才能连接，包括函数名，参数个数类型，返回类型等。"destroy"信号属于GtkObject类，因此可以在开发文档中查找GtkObject目录下的"destroy"signal找到相应的回调函数原型，根据此原型我们可以定义如下处理函数:

```
void
on_window_destroy (GtkObject *object, gpointer user_data)
{
        gtk_main_quit();
}
```

现在，gtk_builder_connect_signals()将会找到它并确认与Glade中指定的函数匹配，因此就把此函数与"destroy"信号连接。当GtkWindow对象"window"销毁时将会调用上述函数。此函数仅仅是调用了gtk_main_quit()来结束循环并退出应用程序。
因为我们不再使用GtkBuilder对象了，所以可以将其销毁并释放为XML文件分配的空间：g_object_unref(G_OBJECT(builder))。
你注意到我们使用G_OBJECT宏将GtkBuilder*转换为GOblet*， 这是必须的因为函数g_object_unref()接受GOblet*类型参数。而GtkBuilder是从GOblet继承的。

Python语言

```
builder.connect_signals(self)
```

Showing the Application Window

在进入GTK+主循环之前，显示我们的GtkWindow类widget，否则它是不可见的。
C语言

```
gtk_widget_show(window);
```

此函数设置了widgets的GTK_VISIBLE标识，告诉GTK+可以显示此widget了。
Python语言

```
editor.window.show()
```

Entering the GTK+ Main Loop

GTK+的主循环是一个无限循环，一旦创建了GUI并设置好了应用程序，就可以进入主循环等待事件的发生。在主循环中发生了很多魔幻的事情，作为一个新手，可以简单的把它看成是一个循环，做了诸如检查状态，更新UI，为事件发送信号等事情。
一旦进入主循环，我们的应用程序就不做任何事了（GTK+在做），当用户缩放窗口，最小化，点击，按键等等时， GTK+检查每一个事件并发送相应的信号。不过，我们的应用程序仅仅对"window"的"destroy"信号进行响应。其它一概不管。
C语言

```
gtk_main();
```

Python语言

```
gtk.main()
```

总结:
1 应用程序使用GtkBuilder从XML文件创建GUI
2 应用程序得到主窗口小部件的引用
3 应用程序把"on_window_destroy"处理函数连接到"destroy"信号

4 应用程序显示窗口
5 应用程序进入GTK+主循环
6 用户点击"X"按钮关闭窗口，导致GTK+主循环发送"destroy"信号
7 信号处理函数"on_window_destroy"退出GTK+主循环
8 应用程序正常终结

What's Next?
在接下来的部分， 我将会完成我们的GTK+文本编辑器余留的功能，并处理所需的信号。对代码不再详细阐述。