wxPython 界面设计入门及进阶

作者:Justin @ JinRui Garden

原文整理自:http://www.leigao.org/blog/archives/434 等

1 wxPython 界面设计利器:wxFormBuilder

之前我曾经介绍过 wxPython 界面设计的工具 BOA(见这里),但是那个主题只有一篇文章,不是因为我太懒,而是 BOA 在稳定性和工具风格上不太合我的胃口。相对于功能追求全面而强大的 BOA,wxFormBuilder 则只专注于 UI 的设计,并生成框架代码,和 GTK 的工具 Glade 非常相似。

wxFormBuilder 并不是为 wxPython 而生,它不仅可以生成 Python 代码,还支持生成 C++ 和 XRC 代码。的确是使用 wxWidget 用户的福音!

wxFormBuilder 对于初始次用的用户来说,确实上手不太容易,因为在它的工程中,无法看到一整个 wx.App()。在 wxFormBuilder 的概念中,UI 的承载容器是:Frame、Panel、Dialog、MenuBar 和 ToolBar 等组件。所以在创建好这些 UI 组件之后,还需要手动创建一个 wx.App() 使这些组件能够展现给用户。

使用 wxFormBuilder 进行设计的步骤是:

1、首先增加上述所说的组件;

2、在组件中放入各种 Sizer;

3、添加各种控件;

4、添加控件响应动作;

5、自动生成代码;然后就可以根据自己的需要,对代码进行重新组织了。

在这个过程中,wxFormBuilder 只负责界面布局、事件绑定和事件接口初始化的部分;剩下的完全由用户来发挥了。

2 wxFormBuilder 入门

在上一篇博客中,我专门介绍了 wxPython 的 UI 设计工具:wxFormBuilder。这篇博客中将对 wxFormBuilder 进行细致深入的介绍。

此文注定是一篇长长的文章,一共分为如下几个部分:1、wxFormBuilder 的安装;2、创建工程前的准备;3、创建第一个工程;4、编写业务逻辑代码。

本文中示例所使用的操作系统是 Ubuntu 10.04 LTS,wxFormBuilder 版本为 3.2.3-beta。

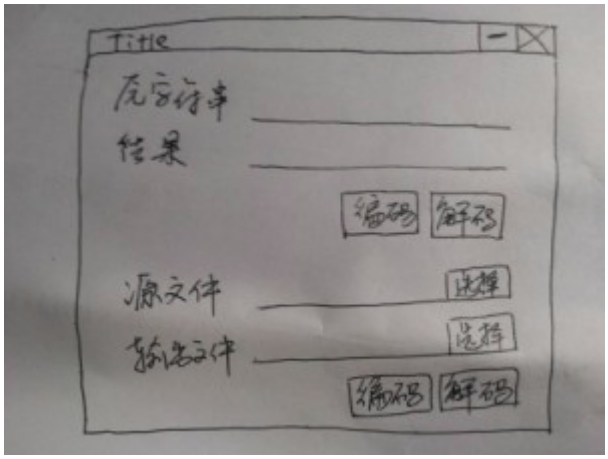
第一部分:wxFormBuilder 的安装在 Ubuntu 中安装 wxFormBuilder 可以选择从 Launchpad 上下载,地址:

https://launchpad.net/~wxformbuilder/+archive/release

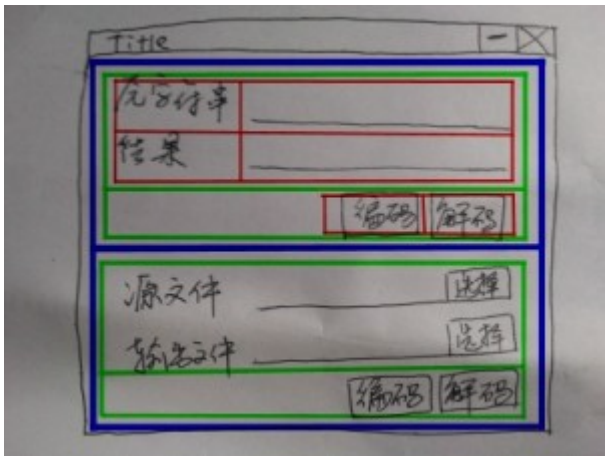
对于其他 Linux 发行版,可以通过系统自带的软件包管理器进行检索;如果无法通过软件包管理器进行安装,可以从 wxFormBuilder 的网站上获取:

http://sourceforge.net/apps/wordpress/wxformbuilder/downloads/

第二部分:创建工程前的准备首先,介绍一下我在这篇博客中将要实现的工程的功能:1、对用户输入的字符串进行 Base64 编解码;2、对用户指定的文件进行 Base64 编解码,将结果保存到另外的文件中。这个工程主要介绍 UI 部分的设计,我画了一份手写稿如下:



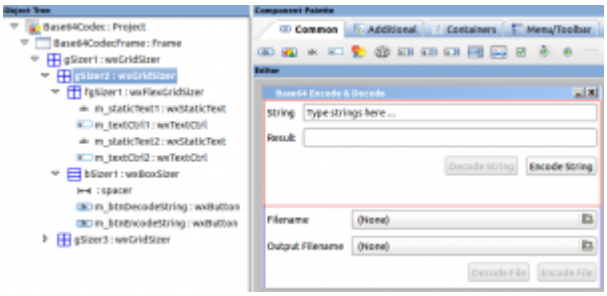
这个窗口中,大体上分为两个部分:上半部分是对字符串进行编解码的操作空间;下半部分是对文件进行编解码的操作空间。上下两个部分是基本对称的。继续观察上部分空间,可以发现,这个空间可以再分解为两个部分:上面的部分是字符串输入与输出的空间,下面的部分是两个操作按钮。继续分解上面的字符串输入与输出空间,这个部分很像汉字中的“田”字,左侧两个窗口中显示的是标签,右侧的两个窗口中显式的是用户输入/输出的字符串。这样一步步分解下来,这个窗口的组成就十分清晰了,在下图中进行了标注:



上面这部分内容,通过对 UI 设计的草稿进行分解,使得 UI 的结构变得很清晰;这些框框在 wxPython 中有专门的名称,叫做“Sizer”。而 sizer 是 wxFormBuilder 设计的基础。

第三部分:创建第一个工程好了,是时候通过 wxFormBuilder 进行创作了。要想将上面图片中的各个按钮、文本输入框等组件组合在一起,首先要有一个“容器”来装载,这个容器可以是一个窗口(Frame)、面板(Panel)或者对话框(Dialog)。所以,首先到“Forms”面板中选择“Frame”容器。wxFormBuilder 是一个 wysiwyg 的 IDE,即刻就可以看到你选择的 Frame。注意:Frame 的属性中,有一个属性是“name”,这个属性就是最终生成代码中 Frame 对象的类名。

OK,有了 Frame 之后呢,就要提到上个章节中的 Sizer 了,可以说掌握 Sizer 是掌握 wxFormBuilder 设计的关键。我首先将设计好的 Sizer 贴上来,然后和上面的图片进行关联讲解,图片如下:



这张截图中,gSizer1 就是上图中的蓝色框体部分,它的类型是 wxGridSizer。wxGridSizer 的图标就像一个“田”字格,可以指定“田”字中行和列的数量。根据草稿的描述,gSizer 应该是一个两行一列(2X1)的 Sizer。gSizer2 与 gSizer3 并列于 gSizer1 中,显然,它们对应的是草稿中的绿色框体了。并且,与 gSizer1 一样,都是 2X1 的 Sizer。接下来是 fgSizer1,这个 Sizer 与前面提到的 Sizer 不同,它属于 wxFlexGridSizer,对应于草稿中的红色“田”字格。剩下还有一个 bSizer1,这是另外一种 Sizer,称为“wxBoxSizer”,也是最简单的一种 Sizer。它的属性只有一个:组件的排列方向(水平还是垂直摆放)。按照上面的步骤,将组件排列好之后,按 F8 生成代码。这样就完成了 UI 部分的设计。

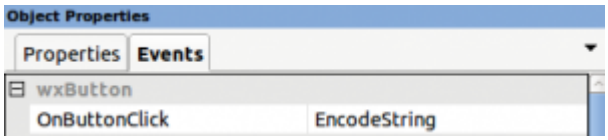
但是请注意,F8 所生成的代码并不能直接运行,因为要展示这个 Frame,需要首先创建一个 wx.App 程序,可以拷贝如下代码到一个 py 文件中,然后放入与上面生成的代码相同的目录中,运行这个 py 程序即可看到我们在上面设计的 Frame。

```
import wx
import Base64CodecMainFrame

# 这是 F8 所生成的代码;
app = wx.App()
main_frm = Base64CodecMainFrame.Base64CodecFrame(None)          # 这个类是我们设计的 Frame
main_frm.Show()
app.MainLoop()
```

运行之后,就可以看到这个 Frame 了,但是工作到此还没有结束,因为 Frame 中有 4 个按钮(Button),用户需要点击这几个按钮来实现不同的功能。以“Encode String”为例,我们希望用户点

击这个按钮之后,能够在“Result”右侧的文本框中输出编码后的结果,这就涉及到 UI 设计的另外一个问题:定义按钮的响应事件,这是实现 UI 交互的关键。选中这个 Button 之后,选择右侧的“事件(Events)”页签,这个页签中显示了这个按钮所能够触发的各种不同的事件,比如点击按钮就是“OnButtonClick”,比如我们希望点击按钮后就可以进行字符串的编码,可以这样设置:



这样将所有按钮的动作设置完成之后,UI 部分的设计才算结束,重新按 F8 生成代码吧。

第四部分:编写业务逻辑代码业务逻辑部分的代码,主要是对上一部分最后定义的按钮动作进行处理。比如我们定义了“EncodeString”接口,那么在生成的代码中,这个接口就需要我们来实现预设定的编码处理逻辑。代码部分如下:

```
def EncodeString( self, event ):
    try:
        import base64
    except:
        self.m_textCtrl2.SetValue("ERROR: Could NOT load module ``base64''")
        return

_istr = self.m_textCtrl1.GetValue()
_ostr = None
    try:
       _ostr = base64.encodestring(_istr)
    except:
       self.m_textCtrl2.SetValue("ERROR: Could NOT decode string: %s" % _istr)
       return

       self.m_textCtrl2.SetValue(_ostr.strip())
       event.Skip()
       return
```

这部分就不做细致的叙述了,完整的代码见附件。

至此,整个 wxPython 工程就完成了。当然,这个工程只是个 Demo,仅作示例用。其中有很多不完善之处,以及设计不合理的地方,具体细节后面继续探讨。

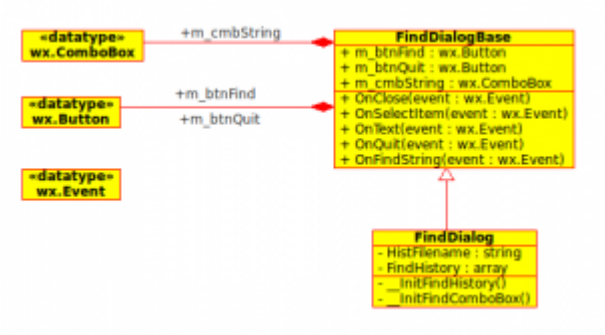
3 wxFormBuilder 进阶:通过类继承实现 UI 和处理逻辑的解偶

在上一篇文章“wxFormBuilder 入门”中,通过 wxFormBuilder 设计了应用程序的界面,并且可以运行了。但是这款应用程序仅仅停留在可以运行的阶段,从设计、维护的角度分析,还存在众多缺陷。在本文中,会结合 wxFormBuilder 工具,谈一谈如何使这款应用更加易于维护。

本文中主要阐述一个问题:如何将 GUI 设计和业务逻辑设计剥离。wxFormBuilder 提供了非常强大的自动生成代码的功能,这也是我们选择 wxFormBuilder 的主要原因。但是使用过后,会发现如

果在通过 wxFormBuilder 设计完 UI 之后,后期 UI 如果稍有变化,那么就必须重新合并 UI 部分和我们定制的业务逻辑代码。

针对这种情况,面向对象思想中,通过“类继承”可以非常容易的避免这种问题带来的麻烦。首先看下图:



在上面的 UML 中我们要设计一个查找窗口(FindDialog),其中包含了两个类:FindDialogBase 与 FindDialog,并且后者继承于前者。在设计的过程中,FindDialogBase 类完全由 wxFormBuilder 设计并自动生成代码。FindDialog 在继承之后,除了显示查找窗口外,还有一个额外的需求:能够记录下用户所有的查找记录。

下面对这个需求进行一下简单的分析:

1、每次用户点击查找进行所有的时候,将用户输入的内容进行记录;

2、当用户关闭查找窗口,或者点击“Quit”结束查找的时候,将记录的条目固化到硬盘上;

3、下次用户调出查找窗口的时候,从硬盘上读取上次保存的条目,并初始化到查找输入框的下拉选单中。

通过以上的分析,自然可以看到,在子类 FindDialog 中需要一个保存用户输入条目的变量,并且需要在初始化(构造)和退出之时加载和保存此变量的值到硬盘上。实现类继承的代码如下:

```
class FindDialog ( FindDialogBase ):
 '''        Implement the Find Dialog;        '''
    def __init__ (self, parent, filename = HIST_FILENAME):
        FindDialogBase.__init__(self, parent)
          # The history filenane;
        self.HistFilename = filename
          # The find history;
        self.FindHistory = []
          # Disable the **Find** Button;
        self.m_btnFind.Enable(False)
          # Initialize the find history;
        self.__InitFindHistory()
          # Initialize the String Combo List;
        self.__InitFindComboBox()
          # Set window focus;
        self.m_cmbString.SetFocus()
```

代码比较简单,就不一一解释了。详细的代码请看附件 FindDialog。

这样通过类的继承,就将 UI 与业务处理逻辑进行了剥离,实现了解偶。比如,在用户的时候过程中,还会不断提出新的需求,比如:

1、增加复选框,用于指定是否大小写敏感;

2、对输入内容进行过滤;

3、向前搜索,向后搜索;

4、……用户的需求会一直不断变化,唯有实现了 UI 和处理逻辑解偶,才能更加从容的应对变化。