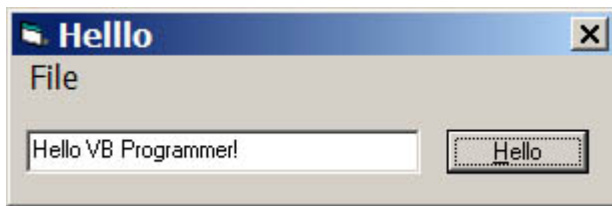


VB Programmer's Intro to Linux Programming with GTK+

August 19th, 2006



From Visual Basic...



... to C/GTK+

I was browsing around in various Linux programming forums the other day and I noticed that I was seeing posts from people quite frequently who were looking to learn Linux programming but had only a little experience with Visual Basic. I was programming in Visual Basic for several years before I started in Linux, and more importantly, I had done very little with C/C++. I made the switch to Linux and have been quite satisfied with programming in C and GTK+. This article is designed to help VB programmers make the switch to Linux application development.

Visual Basic is referred to as a Rapid Application Development (RAD) language. It's power is it's simplicity. You can literally write a program with a GUI for windows in seconds. Migration to Linux as a VB developer can be intimidating. This article will walk through the process of creating a simple "hello world" style application from the point of view of a Visual Basic programmer. The Visual Basic version of the program is shown in the image above on the left. The GTK+ version is shown in the image above on the right. They both are a simple, non-resizable windows with a text box, a command button, and a File -> Quit menu. The Quit menu also has a control key *accelerator* in each program (pressing CTRL+Q invokes that quit menu item).

The GTK+ version uses the Glade interface designer to build the GTK+ GUI. Glade is similar to the form editor in the Visual Basic IDE. The packages you need installed on your Linux system for this article are: gtk2, gtk2-devel, glade2, libglade2, and libglade-devel.

Download the C/GTK+ Project: vb2gtk_gtk.tar.gz

Download the Visual Basic Project: vb2gtk_vb.zip

New Vocabulary, Same Ol' Stuff

One of the intimidating factors of moving to Linux and a new programming language, is all the new terms. As a visual basic programmer, we drop controls on a form. I can't find "control" or "form" anywhere in this GTK+ documentation. What gives! Different names, same thing. You may not know the terms, but you certainly know the concepts.

Visual Basic

Controls - The controls that we drop on our form in VB are often OCX controls or part of some standard library. We can drop a *CommandButton* control or a *TextBox* control onto our form.

Forms - In VB, our windows are called forms and also containers for the controls.

GTK+

Widgets - In GTK+, the "controls" we see in the interface are all *widgets*, or more specifically, *GtkWidgets*. We have a button widget which is a *GtkButton* widget, and we have a text entry widget which is a *GtkEntry* widget.

Window - In GTK+, a window is just another widget, a *GtkWindow* widget. Just like in VB,

the `GtkWindow` is a container of other widgets.

Events - In VB, we write code to handle an "event" that occurs for a particular control using a subroutine or just 'Sub'. For example, we might handle the "Click" event for a *CommandButton* in the subroutine 'Sub Command1_Click()'.

Signals - In GTK+, events are called *signals*. We attach a signal that occurs for a given widget to a *callback function*. For example, we might attach the "Clicked" signal for a *GtkButton* to a callback function we write called 'on_button1_clicked()'.

Subs and Functions - In VB, we put code that returns a value into a 'Function', and code that does not return a value into a 'Sub'.

Functions - In GTK+, when we write code in C (or C++) we just use functions. However, functions can return a type of *void*, which indicates that it does not return a value and thus serves the same purpose as a 'Sub'.

ByRef and ByVal - In VB, we have the option of passing arguments to our functions using the `ByRef` or `ByVal` keyword. If you're not familiar with this, `ByRef` passes address to where the variable is in memory so that a) you don't have to make copies of large variables and b) you can modify the variable as it exists outside of the function. The `ByVal` keyword makes a copy of the variable and thus it is in the scope of the function it is defined in only.

Pointers - In GTK+, or actually, in C and C++, you can pass variable or *pointers* to variables--a pointer is the address in memory where the variable's data is located. This is a very important fundamental of C programming, and is essentially what you are doing in Visual Basic with the `ByRef` keyword. A pointer in C/C++ will be denoted by an asterick (*).

Methods and Properties - In VB, our controls and other *Objects* exhibited Properties (such as 'Caption', 'ForeColor', etc) and Methods (such as 'Add()', 'Move()', etc).

Functions - C doesn't naturally support Objects as such. In GTK+, the same *Object Oriented* functionality is achieved through functions and *structures*. We get and set the "property" of an Object (such as a widget) through function calls. The methods are also function calls (as they were in VB). This changes the code syntax a bit. We'll see how this works in relation to VB later in the article.

Introduction to Glade

If you've ever looked at a Visual Basic 5/6 form file in a text editor, you'll see that it is simply a listing of the controls that make up the form, with each control that is "in" a container control nested between 'Begin' and 'End' keywords. The file `Form1.frm` used in the Visual Basic project for this article looks like this (I have removed the code, this is just the layout of the form):

```
Begin VB.Form Form1
BorderStyle = 3 'Fixed Dialog
Caption = "Hellllo"
ClientHeight = 750
ClientLeft = 150
ClientTop = 810
ClientWidth = 4500
```

```

LinkTopic = "Form1"
MaxButton = 0 'False
MinButton = 0 'False
ScaleHeight = 750
ScaleWidth = 4500
ShowInTaskbar = 0 'False
StartupPosition = 3 'Windows Default
Begin VB.CommandButton Command1
Caption = "&Hello"
Height = 330
Left = 3255
TabIndex = 1
Top = 210
Width = 1065
End
Begin VB.TextBox Text1
Height = 330
Left = 105
TabIndex = 0
Top = 210
Width = 2955
End
Begin VB.Menu mnuFile
Caption = "&File"
Begin VB.Menu mnuExit
Caption = "E&xit"
Shortcut = ^Q
End
End
End

```

This is all the information that is needed for Visual Basic to "generate" a GUI for your application. You then go in and write code to handle "Events" that occur for various controls within the application.

We'll be doing the same thing for GTK+ programming. The Glade Interface Designer works much like the WYSIWYG form layout editor in VB. Glade will save the information in a file with a .glade suffix, in this case, we're going to call it gui.glade. This Glade file is actually an XML file which looks like this:

```

<?xml version="1.0" standalone="no"?> <!--*- mode: xml -*-->
<!DOCTYPE glade-interface SYSTEM "http://glade.gnome.org/glade-2.0.dtd">
<glade-interface>

<widget class="GtkWindow" id="window1">
<property name="visible">True</property>
<property name="title" translatable="yes">window1</property>
<property name="type">GTK_WINDOW_TOPLEVEL</property>
<property name="window_position">GTK_WIN_POS_NONE</property>
<property name="modal">False</property>
<property name="resizable">False</property>

```

```

<property name="destroy_with_parent">False</property>
<property name="decorated">True</property>
<property name="skip_taskbar_hint">False</property>
<property name="skip_pager_hint">False</property>
<property name="type_hint">GDK_WINDOW_TYPE_HINT_NORMAL</property>
<property name="gravity">GDK_GRAVITY_NORTH_WEST</property>
<property name="focus_on_map">True</property>
<signal name="delete_event" handler="on_window1_delete_event" last_modification_time="Wed, 08
Mar 2006 05:03:00 GMT"/>
<signal name="destroy" handler="on_window1_destroy" last_modification_time="Wed, 08 Mar 2006
05:03:22 GMT"/>

<child>
<widget class="GtkVBox" id="vbox1">
<property name="visible">True</property>
<property name="homogeneous">False</property>
<property name="spacing">0</property>

<child>
<widget class="GtkMenuBar" id="menubar1">
<property name="visible">True</property>

<child>
<widget class="GtkMenuItem" id="file_menuitem">
<property name="visible">True</property>
<property name="label" translatable="yes">_File</property>
<property name="use_underline">True</property>

<child>
<widget class="GtkMenu" id="file_menuitem_menu">

<child>
<widget class="GtkImageMenuItem" id="quit_menuitem">
<property name="visible">True</property>
<property name="tooltip" translatable="yes">Quit Application</property>
<property name="label" translatable="yes">_Quit</property>
<property name="use_underline">True</property>
<signal name="activate" handler="on_quit_activate" last_modification_time="Wed, 08 Mar 2006
04:52:22 GMT"/>
<accelerator key="Q" modifiers="GDK_CONTROL_MASK" signal="activate"/>

<child internal-child="image">
<widget class="GtkImage" id="image1">
<property name="visible">True</property>
<property name="stock">gtk-quit</property>
<property name="icon_size">1</property>
<property name="xalign">0.5</property>
<property name="yalign">0.5</property>
<property name="xpad">0</property>
<property name="ypad">0</property>
</widget>
</child>

```

```

</widget>
</child>
</widget>
</child>
</widget>
</child>
</widget>
<packing>
<property name="padding">0</property>
<property name="expand">False</property>
<property name="fill">False</property>
</packing>
</child>

<child>
<widget class="GtkHBox" id="hbox1">
<property name="border_width">10</property>
<property name="visible">True</property>
<property name="homogeneous">False</property>
<property name="spacing">0</property>

<child>
<widget class="GtkEntry" id="entry1">
<property name="visible">True</property>
<property name="can_focus">True</property>
<property name="editable">True</property>
<property name="visibility">True</property>
<property name="max_length">0</property>
<property name="text" translatable="yes"></property>
<property name="has_frame">True</property>
<property name="invisible_char">*</property>
<property name="activates_default">False</property>
</widget>
<packing>
<property name="padding">0</property>
<property name="expand">True</property>
<property name="fill">True</property>
</packing>
</child>

<child>
<widget class="GtkButton" id="button1">
<property name="visible">True</property>
<property name="can_focus">True</property>
<property name="label" translatable="yes">_Hello</property>
<property name="use_underline">True</property>
<property name="relief">GTK_RELIEF_NORMAL</property>
<property name="focus_on_click">True</property>
<signal name="clicked" handler="on_button1_clicked" last_modification_time="Wed, 08 Mar 2006
05:01:25 GMT"/>

```

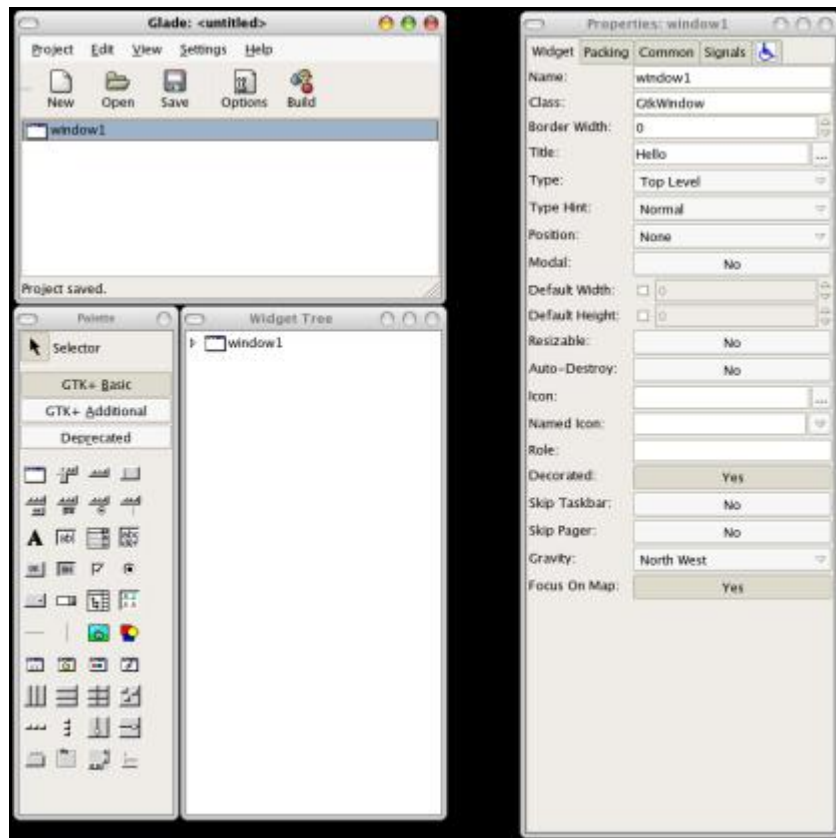
```

</widget>
<packing>
<property name="padding">0</property>
<property name="expand">False</property>
<property name="fill">False</property>
</packing>
</child>
</widget>
<packing>
<property name="padding">0</property>
<property name="expand">True</property>
<property name="fill">True</property>
</packing>
</child>
</widget>
</child>
</widget>
</glade-interface>

```

The format of the VB form is much different from the XML used by Glade, however, it should be fairly obvious that they are doing the same thing--defining the widgets (controls) and their layout on their parent window (form). We use the program Glade to create this XML file and are actually going to be using the libglade library to read this XML file from within our C program and generate the GUI at runtime.

If you're using the Gnome desktop and have installed Glade, it will be in the Gnome main menu under 'Programming' -> 'Glade Interface Designer'. There are 5 windows associated with Glade, only 4 of which we are going to concern ourselves with now. They can be hidden/show from the Glade's 'View' menu. They are the main Glade window, the *Palette*, the *Widget Tree*, and the *Properties* window. Make sure you have these windows visible and lay them out on your desktop so they are all visible. A screenshot of my workspace is shown below (You won't have 'window1' listed just yet).



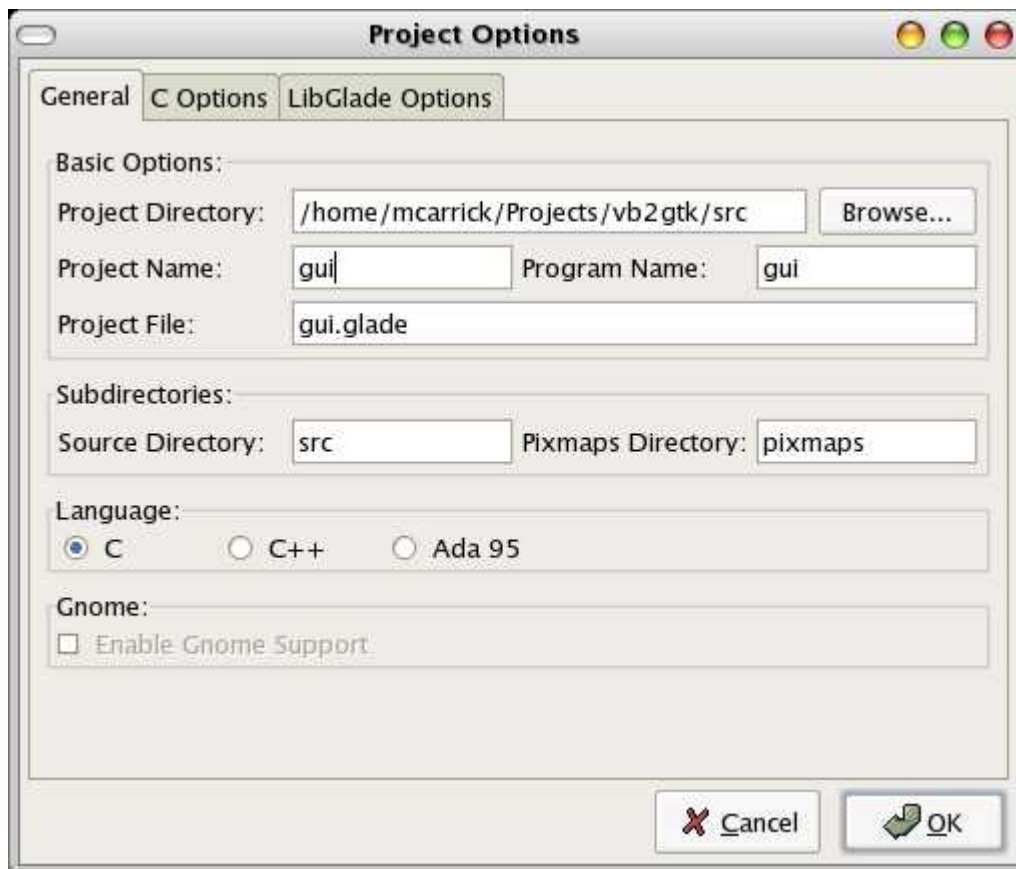
You should already see some familiarities between Glade and Visual Basic. The *Palette* likely reminds you of the VB Toolbox, the *Properties* window corresponds to the *Properties* window of VB, and the main window along with the *Widget Tree* is similar to the VB Project Explorer window.


- Main Window - Lists each window widget in the project and allows you to open/save projects, show/hide Glade windows, etc.
- Palette - A "toolbox" of available GTK+ widgets you can drop in to another container widget, such as the *GtkWindow* widget.
- Widget Tree - A listing of all the widgets in the project in a tree-like structure to show the parent-child relationship of each widget.
- Properties - Allows you to set various properties for the selected widget. The 2 new concepts here are going to be "packing" and "signals" which we'll talk about later in the article.

Laying Out the GUI in Glade

First thing we're going to do, is start a new project. Like most programs, this can be done three ways: Click 'New' from the 'File' menu, click the 'New' toolbar button, or press CTRL+N on the keyboard. When you do this, you are prompted with two choices. One is to create a new GTK+ Project and the other is to create a new GNOME project. For this example, we're going to create a GTK+ Project (For an introductory tutorial on Gnome programming, see my tutorial: [Simple Gnome Application Using libglade and C/GTK+](#)).

Next, we're going to save our project. Since we're using libglade, we don't need to worry about any of the "C Options" etc. We just need to specify the "Project Directory", "Project Name", and "Project File". The project file is going to be the XML file that we use in our C code. My screen shot is shown below:




Now we're going to layout our GUI. I'm not going to explicitly tell you all the time to save. So just make sure you punch CTRL+S every time you're at a good spot and don't want to lose what you've done. First thing we need is a GtkWindow widget. We add a GtkWindow widget by clicking the GtkWindow icon  in the Glade Palette. A window pops up titled "window1". This is going to be our main application window. We are now ready to drop widgets into the main window.

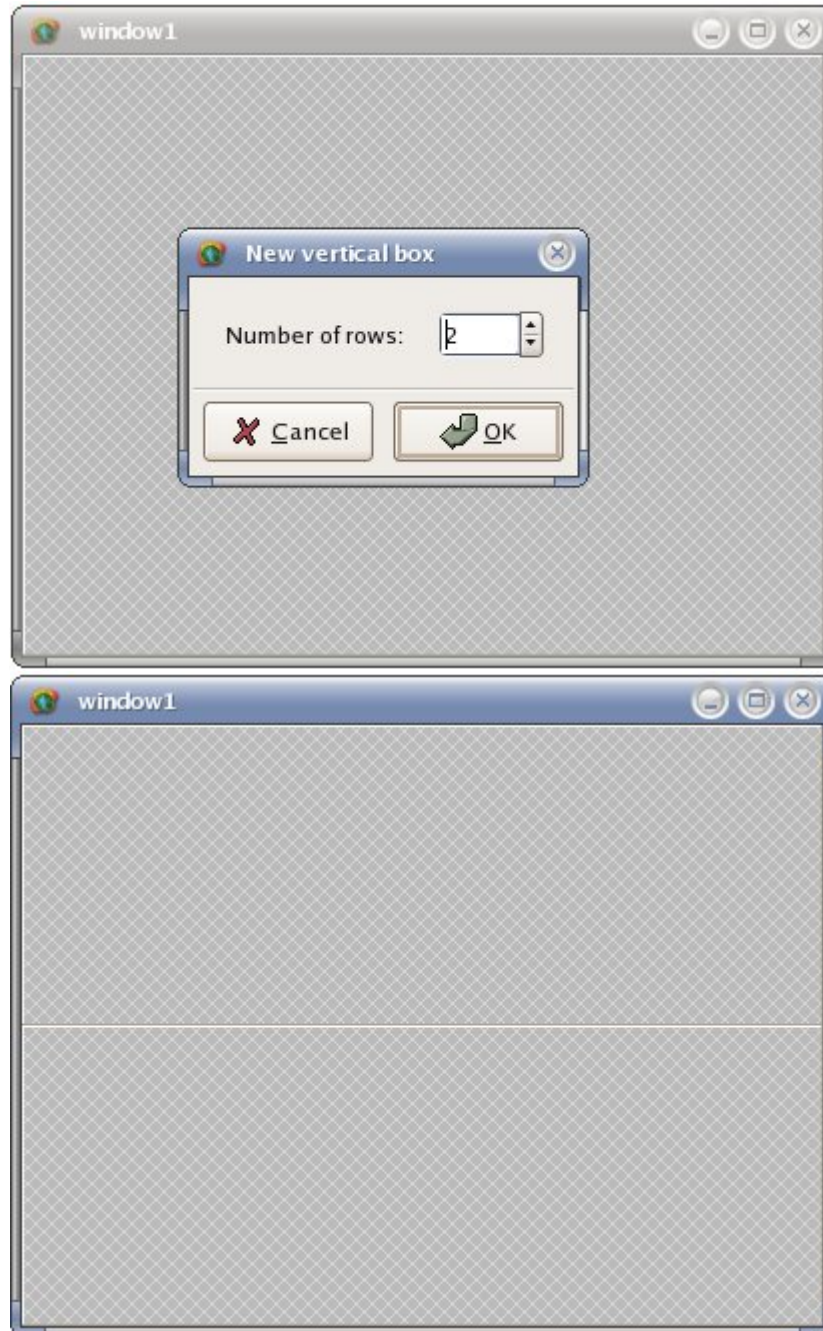
Before I go any further, I want to talk about "packing". This is one area where GTK+ differs a bit and it initially seemed annoying to me. However, it saves a lot of time--among other things. So, In VB we would drop our controls on a form and that's where they would stay there. We explicitly wrote code to handle resizing of the controls. With GTK+, the widgets are *packed* with various properties into their containers which indicate how they will be sized/moved with the parent. Therefore, container widgets are very important in GTK, where as we had very few in VB (maybe the FrameControl and PictureBox). We're going to use a GtkHBox, which is a series of horizontal box containers, and the other is a GtkVBox, which is a series of vertical box containers.



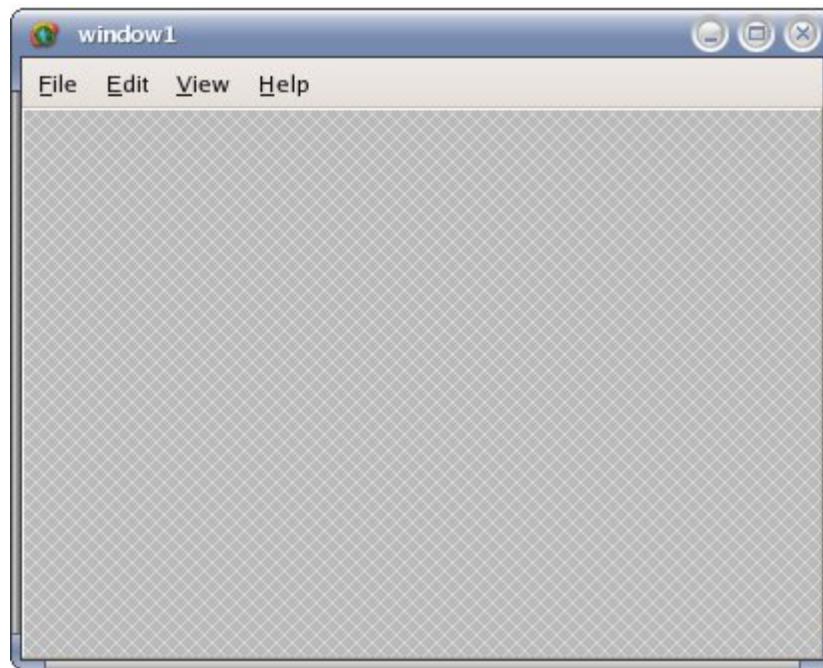
The image above shows the window we're creating sectioned off. The red outline is a **GtkVBox** with a "size" of 2. That is, 2 container boxes stacked on top of each other. The top one will contain our menu widget. The bottom one contains a **GtkHBox** with a size of 2. That is, 2 container boxes side-by-side. Then, inside of that GtkHBox we have our text entry widget and our button widget. The reason we are laying out these widgets inside of container widgets is to control the resizing or "packing" behavior of

the window and its widgets.

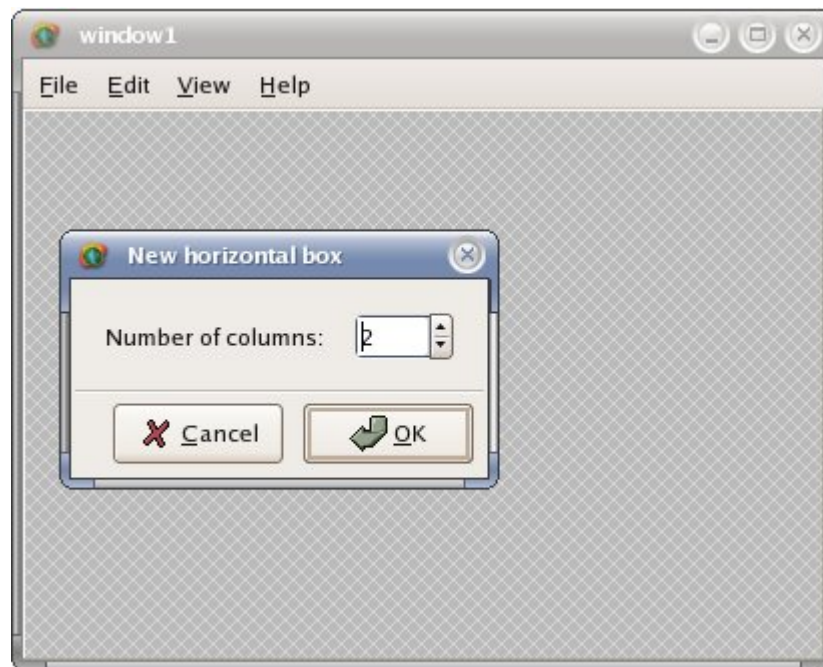
So, the first widget we're going to add is the GtkVBox widget. Do this by clicking the GtkVBox icon  in the Glade Palette. Then, click in the gray, empty area of "window1" to place the GtkVBox there. You will be prompted for the "Number of Rows". We want 2 rows in this GtkVBox, so change it from the default of 3, to 2 and click "Ok". The window should now have 2 panes, but each of those panes is still empty.

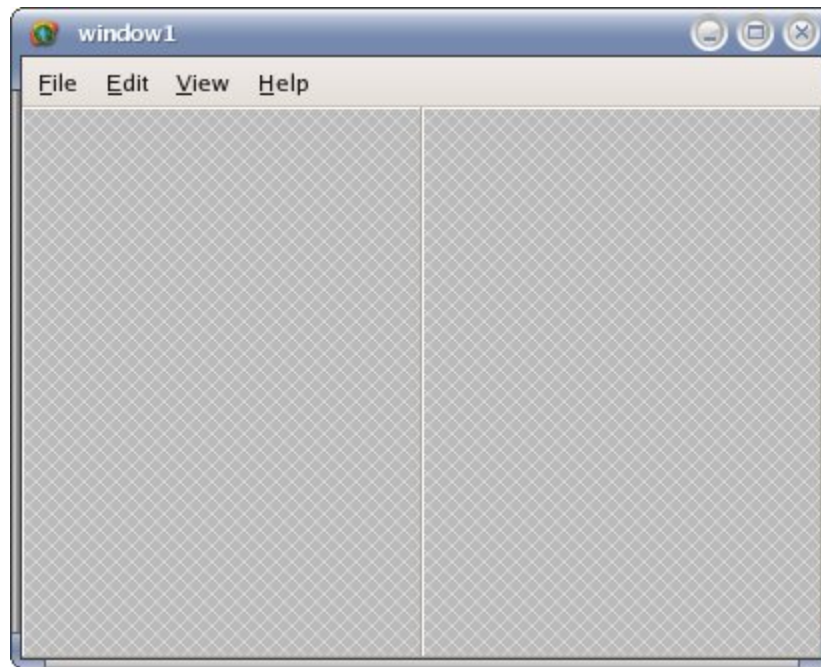


In the top pane, we're going to add a GtkMenuBar. Do this by clicking the GtkMenuBar icon in the Glade Palette. Then, click in the empty area of the top pane in "window1". The GtkVBox will *snap* around the GtkMenuBar, leaving the bottom section blank. Since there is a widget in the GtkVBox's top pane, it is exhibiting the packing behaviour around that object. We'll do some experimenting later to illustrate this more.

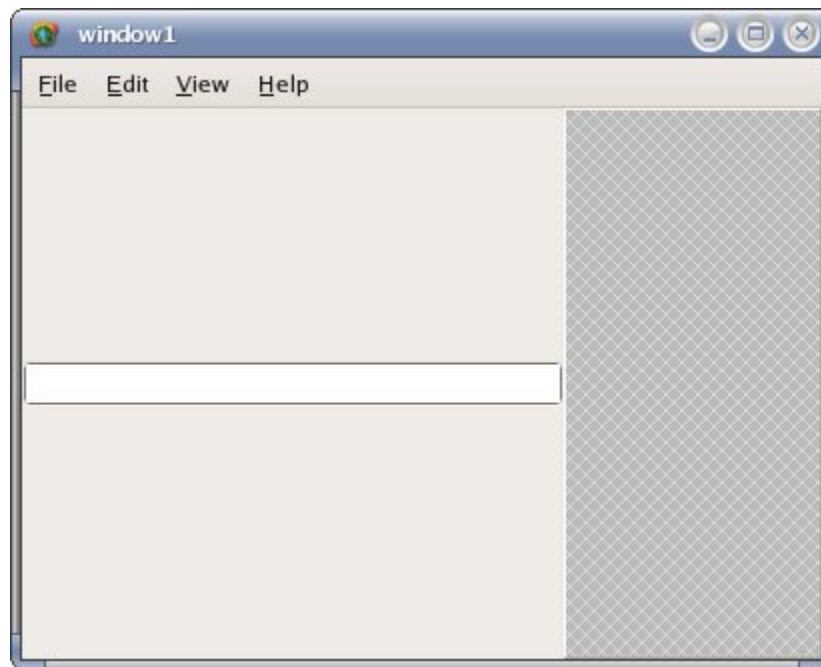


Now, in the lower pane of the GtkVBox, we need to add the GtkHBox. Click the GtkHBox icon in the Glade Palette. Then, click in the empty area of the bottom pane in "window1". You are prompted for the "Number of Columns" to which you want to input 2 and click "Ok". The result is that bottom empty section of the window gets sectioned off into 2 empty sections.





Next, add the GtkEntry widget to the left container. Click the GtkEntry icon and then click in the left empty pane in "window1". Again, the entry snaps in to place filling up the entire area (Don't worry about the window size yet).



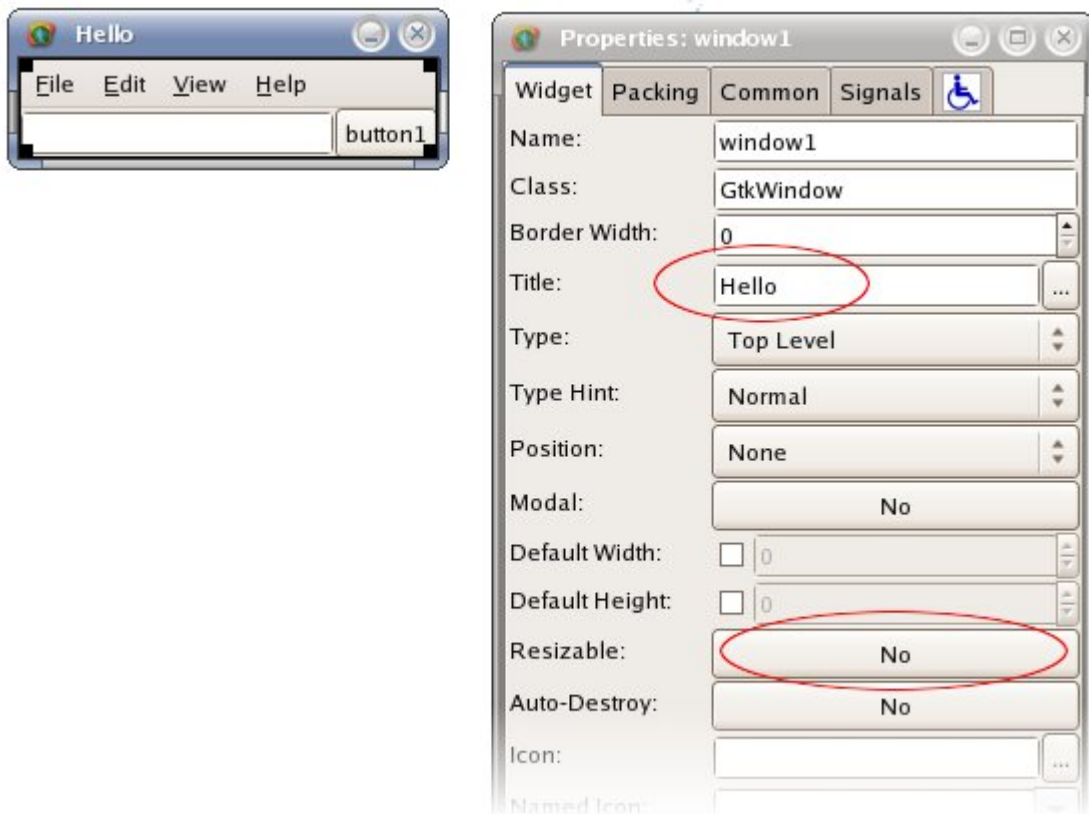
And finally, let's add the GtkButton. Click the GtkButton icon in the Glade Palette and click in the remaining empty space in "window1". The button snaps to fill in the remaining space. Might not look real pretty just yet, but hold on. We're getting there.



What we want to do now, is look at the Widget Tree window. This is the easiest way to "select" a widget in your interface. For example, to set the properties of the GtkHBox, you can't easily get to it from it's physical layout in window1. But using the widget tree makes it easy. It also shows us the parent-child relationships of our widgets.

Now, this particular application is going to be a fixed sized window. Click on "window1" in the widget tree, and then look at the Glade Properties window. This is very similar to the VB properties window. The first thing I want to point your attention to, is the box labeled "Class". For the window, it says GtkWindow. This is the official name of the widget's class, it's object name. This is what you use to lookup the widget's properties, functions, and signals in the GTK API documentation.

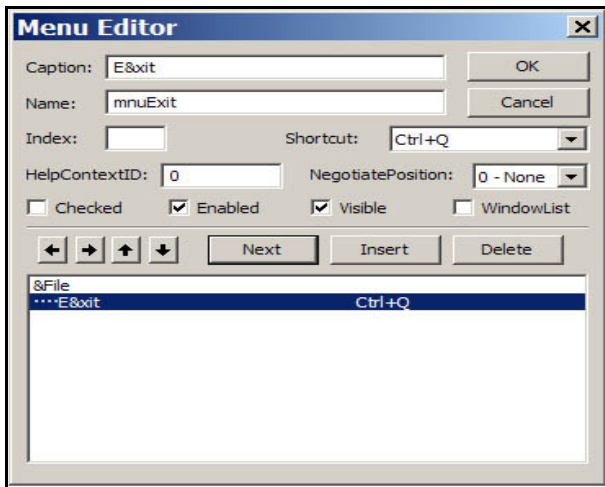
There are 2 things we are going to do here in the Properties for window1. First, change the "Title" from "window1" to "Hello". Next, change the "Resizable" property from "Yes" to "No". Notice what happens? The window snaps around it's widgets.



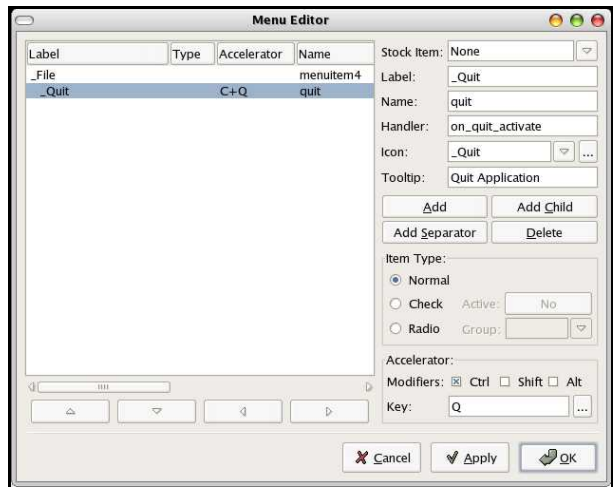
It may be starting to look like the final window--but not quite. First, I like a little breathing room around my widgets and the edge of the window. There are several different properties of widgets which allow you to tweak the layout of the widget. In the properties window under both 'Widget' and 'Packing' you can find 'Border Width', 'Padding', and 'Spacing'. Which parameters are available are dependent on the widget you have selected. For this article, I selected the GtkHBox and changed the 'Border Width' in the 'Widget' tab to 10. I encourage you to (first save your project) play with the different packing options for the different widgets to get a feel for how they behave. You should especially take time experimenting with the following:

- Homogeneous
- Expand
- Fill
- Padding
- Spacing
- Border Width

The last thing to do with our interface is modify the menu items. When we dropped the GtkMenu in, it came pre-loaded with items. We only need the file menu and it only needs the quit menuitem. So, let's open up the Glade menu editor. We do so by selecting "menubar1" in the Widget Tree, and then clicking the "Edit Menus..." button in the 'Widget' tab of the Properties window.



Vis



Gla

ual Basic menu editor

de menu editor

The menu editor is very similar to the VB menu editor. There are a few differences to point out.

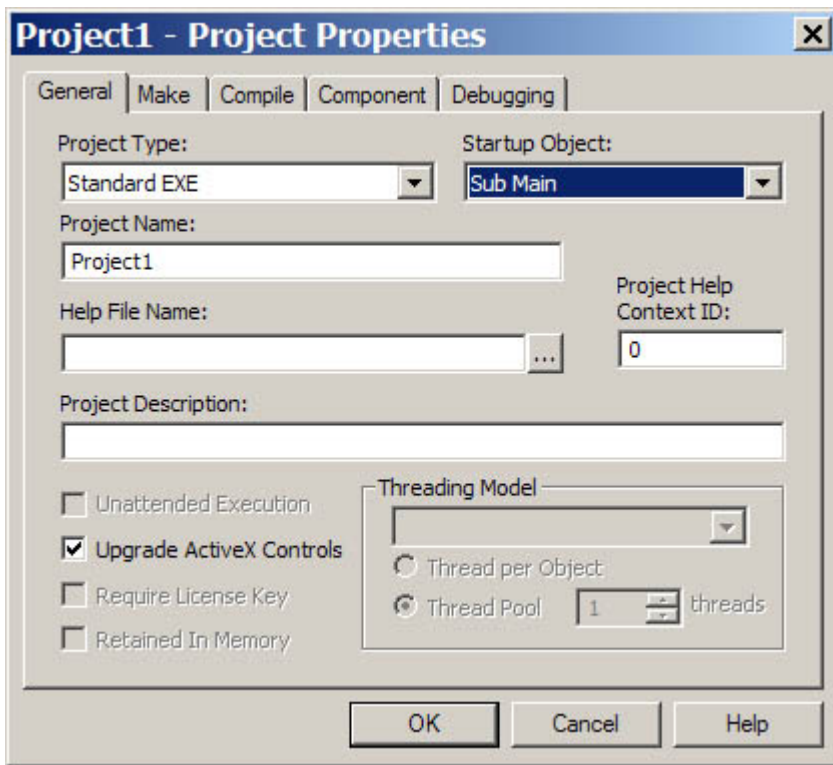
- In VB the '&' ampersand character was used to signify a letter in a caption that was the activator for that menu or control. In Glade, the '_' underscore character does this. So &File in VB ends up being _File in Glade.
- The "Shortcut" for menu items in VB (pressing a sequence of keys such as CTRL+S activates the menu) is called an *Accellerator* in Glade.
- GTK+ provides a number of "Stock" buttons for toolbars, buttons, and menu items. These are commonly used actions such as new, open, save, ok, cancel, etc. You can have a stock menu item, or simply use a stock image. When a user changes their GTK+ theme, they may also be changing the "Stock" icons to match the theme. You should use stock images whenever you can for this reason.
- Glade allows you to specify the "Handler" for a menu item. This is the name of the function that handles the code (which was automatically chosen in VB). We'll learn more about this later.

Now for this example, we just want the "File" and "Quit" menu items. So you can delete everything else. Additionally, I changed the 'quit' menu item so that it is NOT a stock menu item, but instead only uses a stock image (see the screenshot above, "Stock Item" is "None" and the images has _Quit selected, a stock icon). I did this because the stock menuitem does not use the CTRL+Q accellerator. Additionally, I renamed it from 'quit1' to just 'quit', and the Handler from "on_quit1_activate" to "on_quit_activate". This is just me being obsessive compulsive.

Writing The Code

I'm going to discuss the GTK+ code as it relates to Visual Basic code. The VB code I list is completely fake just to illustrate the sytactical differences between Visual Basic and GTK+.

If you have only minimal experience with Visual Basic, you may not know about the 'Startup Object' option in the Project Properties window (shown in the image below). What this does, is rather than the project starting with **Form1** (or whatever your main form is), it starts in a **Sub** named **Main** which you can put in a module. Typically, you perform startup routines, argument handling, etc., and then show the form when you're ready using **Form1.Show**.



The Visual Basic project I have setup for this article has employed the **Sub Main** method. The reason I have pointed this out is because when we write our C program, we have a **main** function which is also our entry point, and then we show our main window using a function call much like the VB **Show** method.

I'm going to be discussing the code in **main.c**, which in this program, houses all of the code for the entire application. The top of the file contains code to sort of setup our program. In C, there is a bit more work we have to do that VB would have done for us. Let's take a look:

```
/* include files: GTK+ library and libglade */
#include <gtk/gtk.h>;
#include <glade/glade.h>;
```

Here we're "including" header files which define the functions, variables, classes, etc. available to use in our program. In this case, we're including the GTK+ library and the Glade library. These libraries can in turn include other libraries.

```
/* path to glade interface file relative to where executable is */
#define GLADE_FILE "gui.glade"
```

This is a constant which we also had in Visual Basic, and like VB, we use capital letters to indicate it. The difference here, is the '#' character is a *compiler directive*. When the C compiler generates the executable, every occurrence of **GLADE_FILE** in the code is replaced with "gui.glade". This way, if I rename my glade file or store it in a different location, I only have to modify 1 line in the source code.

```
/* Declaring a global XML object. Part of libglade (glade.h) */
GladeXML *gxml;
```

This is a global variable which shouldn't be a new concept. But I will take this opportunity to point out how variables are declared in C vs. VB. First of all, you didn't have to declare variables in VB (unless you specified **Option Explicit**). Secondly, VB had a "variant" type. We're not so privileged here. We

MUST specify the type of the variable and declare any variable we use. In visual basic, the above might look like:

```
Dim gxml As GladeXML
```

You shouldn't have any trouble finding information on declaring variables in C. I also have some "function prototypes" at the top of the file. Any time you call a function in a C program, that function has to have been declared and defined. When we included the header files, all the GTK+ and glade functions were declared. For my own user functions, I have to either locate the functions before they are called in the source, or declare them using function prototypes as I have done here. Again, this is a basic C concept and you can find information about function declarations/function prototypes all over the internet or in any beginner book on C.

```
/* function prototypes let the compiler know what functions are coming later */  
void on_window1_destroy_cb (GtkWidget *widget, gpointer user_data);  
void on_quit_activate_cb (GtkWidget *widget, gpointer user_data);  
void on_button1_clicked_cb (GtkWidget *widget, gpointer user_data);
```

Just to help you along, in VB these functions would look more like:

```
Sub on_window1_destroy_cb (ByRef widget As GtkWidget, ByVal user_data As gpointer)
```

Now let's look at the main code.

```
int  
main (int argc, char *argv[])  
{
```

The program begins execution at the function main(). In VB, this would look like:

```
Function main (ByVal argc As Integer, ByRef argv As Variant) As Integer
```

The argc and argv variables are the command line options and I'm not going to discuss them here.

```
/* A widget - part of GTK+ (gtk.h) */  
GtkWidget *main_window;
```

Declaring a variable which will be the main window. So in VB we might have access to Form1 automatically, in GTK/C we need to declare it. The VB equivalent might look like:

```
Dim main_window As GtkWidget;
```

Next, we have to initialize the GTK+ library. This is just a basic function call, however, note those ampersands (&). Those are related to pointers and the VB ByRef keyword. If the function wants a ByRef argument, that is, a pointer, then we have to send it a pointer (which is the address to the variable). If our variable was not declared as a pointer (with the *), then it has to be passed with the & character to say "pass the address of the variable".

```
/* initialize the GTK+ library */  
gtk_init (&argc, &argv);
```

Next I'm assigning a value to the gxml variable I declared globally:

```
/* create an instance of the GladeXML object. */  
gxml = glade_xml_new (GLADE_FILE, NULL, NULL);
```


Now, I mentioned earlier that GTK+ does all the object oriented type of stuff through functions. This is an example. Before this line, gxml wasn't assigned any value. I have to initialize it. While initializing it, I'm assigning it 3 "parameters". These are the glade filename, the root widget from which to start building, and the domain respectively. For now we only need to specify the glade file. In VB we do this with objects (such as listitems and treeview nodes). In Visual Basic, the declaration and assignment of the gxml variabl might look more like this:

```
Dim gxml As New GladeXML
gxml.FileName = GLADE_FILE
gxml.Root = vbNull
gxml.Domain = vbNull
```

The next thing is connecting functions to handle signals in the GUI. In visual basic, this was done automatically where a particular Sub (such as onClick) was used to handle an event. So the below 3 lines of code is telling libglade to call a particular function when a particular event occurs. The 3rd call to glade_xml_signal_connect below is telling libglade to call the "on_button1_clicked_cb" function when the button in our GUI is clicked.

```
glade_xml_signal_connect (gxml, "on_window1_destroy",
    G_CALLBACK(on_window1_destroy_cb));

glade_xml_signal_connect (gxml, "on_quit_activate",
    G_CALLBACK(on_quit_activate_cb));

glade_xml_signal_connect (gxml, "on_button1_clicked",
    G_CALLBACK(on_button1_clicked_cb));
```

Sorry, this article is still under construction. I've posted what I have so far as I don't know when I'll have time to finish it.