28th September 2009

Glade3 tutorial (6) - Signals

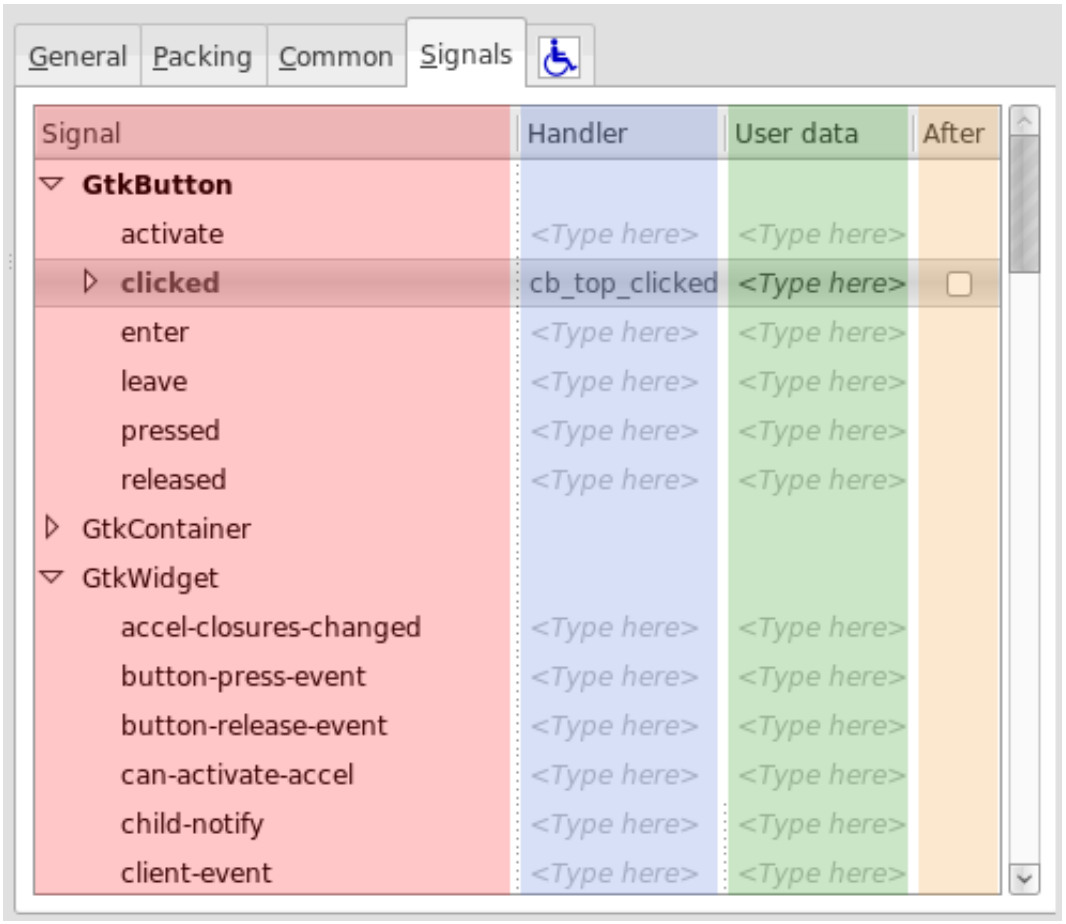
Hello.

As promised, we'll finally connect some signals to handlers and write some C code that will do something useful.

Contents

Signals tab in Glade

If you select one of the buttons, signals tab will look like this (I color-coded columns):



Inside "Signal" column are listed signals, grouped by widget type. "Handler" column is where you insert

name of the function that should be hooked to this signal. "User data" column can hold object name from this glade file that should be sent as data parameter to callback function (note: if field is not empty, signal will be connected the same way as if you were to call g_signal_connect_swapped macro). Last, "After" column holds check button that controls how you callback is connected: if checked, your signal will be connected like you were to call g_signal_connect_after, which means that your function will be called after the default signal handler.

For 99% of time, you can safely ignore last two columns, since they are rarely used, but it's still nice to know why exactly they stand there for.

Signal connection theory

Signal connection is done in two stages:

1. Names of the functions that should be evoked when signal is emitted are specified in "Signals" tab in Glade.
2. Mapping of *function name -> function address* is done at runtime by either using GModule or manually by user.

First stage is simple: just fill appropriate fields inside "Signals" tab and save the project.

Second stage is more complex to understand, because there are two different ways of mapping function names to function addresses. First one (the simple one) is to use gtk_builder_connect_signals function, which will automatically search for function with proper names in your main executable. I won't go into details of how this is done, but keep in mind that GModule is required for this operation. Second method of mapping names to addresses is to use gtk_builder_connect_signals_full, where you must provide function that will do the mapping.

In this tutorial, we'll use automatic method, since this is what most people will end up using in their applications.

Passing custom data to callbacks

Any modestly complex application will require some data exchange to and from callback functions. So how can we achieve this?

As already mentioned, one way of passing data to callback function is to specify object name in Glade. This is simple, but fairly limited method, since only object from your UI file can be passed like this. More flexible way of passing data to callbacks is by using data parameter of gtk_builder_connect_signals. Whatever you specify as a last parameter will be passed to all of the connected callbacks as a last parameter.

"But there is only one argument available, and I would need more!" Well, this problem is usually solved by defining a structure that holds all of the data that may be needed in any of the callbacks. Pointer to an instance of that structure is then passed as a last parameter to gtk_builder_connect_signals.

Callback handler definition

When using gtk_builder_connect_signals to connect signals, you most take some additional steps to ensure that functions are found by GModule. Exact steps needed are platform specific, but Glib developers kindly provided means to write portable code.

Before function declaration/definition, you should place G_MODULE_EXPORT macro. So if we have callback handler for button's "clicked" signal defined like this in non-glade application:

```
static void
cb_clicked( GtkButton *button,
            gpointer   data )
{
    /* CODE HERE */
}
```

we need to modify it to this in order for GModule to be able to find it:

```
G_MODULE_EXPORT void
cb_clicked( GtkButton *button,
            gpointer   data )
{
    /* CODE HERE */
}
```

Other step needed to make code function properly on all platforms is to link executable with proper linker flags. Thanks to Glib developers, all that we need to do is add "gmodule-2.0" to pkg-config parameters. If you compiled your normal applications using compile lines similar to those:

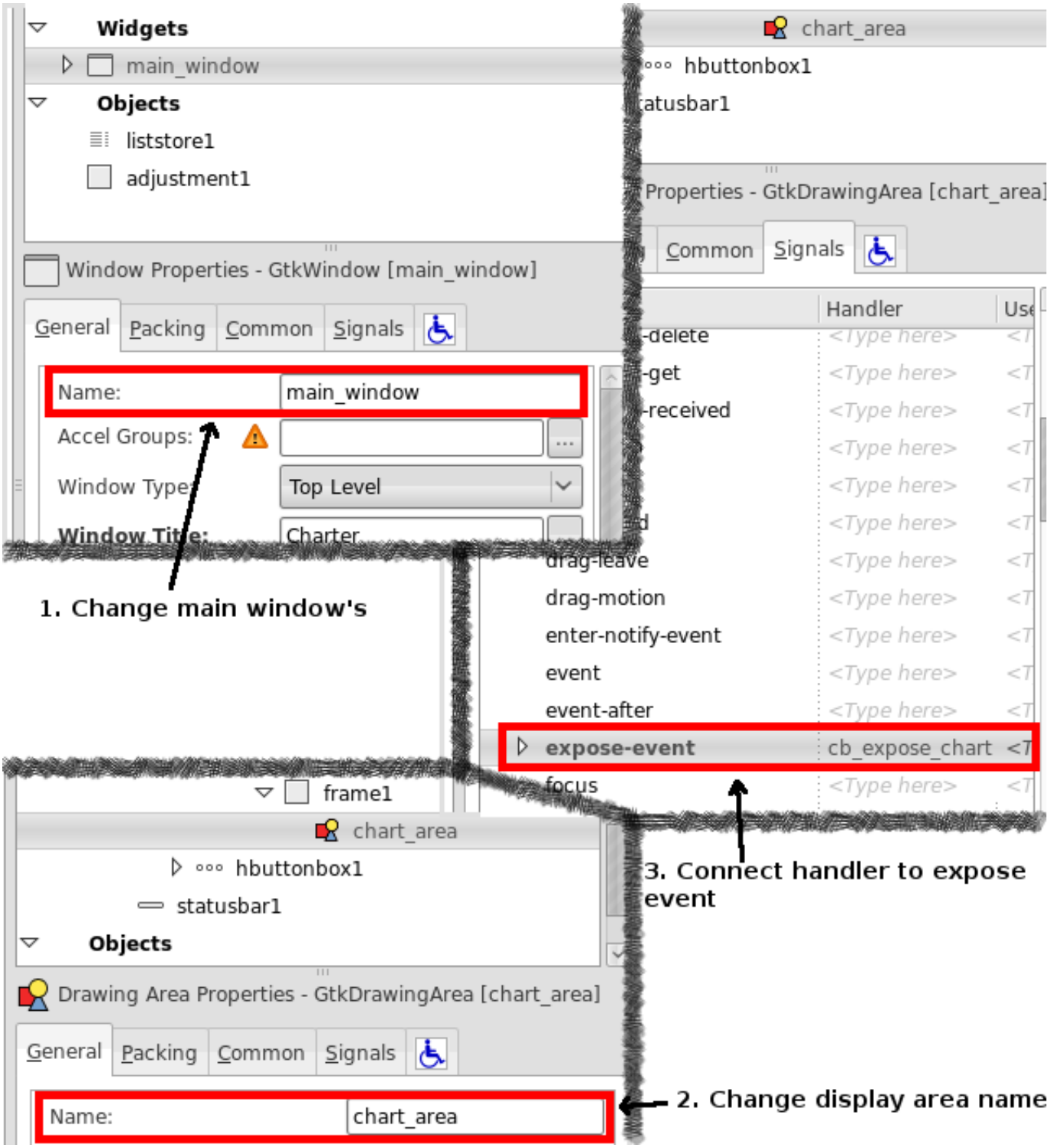
```
gcc -c object1.c $(pkg-config --clfags gtk+-2.0)
gcc -c object2.c $(pkg-config --cflags gtk+-2.0)
gcc -o app object1.o object2.o $(pkg-config --libs gtk+-2.0)
```

then adopted new compile lines will look like this:

```
gcc -c object1.c $(pkg-config --clfags gtk+-2.0 gmodule-2.0)
gcc -c object2.c $(pkg-config --cflags gtk+-2.0 gmodule-2.0)
gcc -o app object1.o object2.o $(pkg-config --libs gtk+-2.0 gmodule-2.0)
```

Preparing glade file

Before we can start writing code, we need to do some minor modifications to glade file: rename "window1" to "main_window", rename "drawingarea1" to "chart_area" and connect "expose-event" to cb_expose_chart function.



Note though, renaming is not needed, but it's more convenient if we rename widgets that we need. With these changes in place, we're ready to start creating code.

Code

This is where Glade cannot help us anymore. Let's start by creating empty folder that will hold all of our files. Now copy latest glade file you saved here and rename it to "charter.glade".

Our code will be initially divided into 3 source files:

1. charter.c will hold main function
2. support.h will hold some convenience macros and main data structure definition

3. callbacks.c will hold functions that we connected in Glade

We'll start with support.h file.

```
#ifndef __SUPPORT_H__
#define __SUPPORT_H__

#include <gtk/gtk.h>

/* Convenience macros for obtaining objects from UI file */
#define CH_GET_OBJECT( builder, name, type, data ) \
    data->name = type( gtk_builder_get_object( builder, #name ) )
#define CH_GET_WIDGET( builder, name, data ) \
    CH_GET_OBJECT( builder, name, GTK_WIDGET, data )

/* Main data structure definition */
typedef struct _ChData ChData;
struct _ChData
{
    /* Widgets */
    GtkWidget *main_window;  /* Main application window */
    GtkWidget *chart_area;   /* Chart drawing area */
};

#endif /* __SUPPORT_H__ */
```

First thing we do is include GTK+ header file, which will provide type and function declarations. Next we define two macros that will make obtaining object pointers less verbose. Consult your favourite C preprocessor manual for more information about how this two macros work. Last thing in this file is structure definition. This structure will serve as our main data storage. We'll add more fields to it gradually. For starters, main window and chart area pointers will suffice.

Next file that we'll create is charter.c.

```
#include "support.h"

#define UI_FILE "charter.glade"

int
main( int    argc,
      char **argv )
{
    ChData     *data;
    GtkBuilder *builder;
    GError     *error = NULL;

    /* Init GTK+ */
    gtk_init( &argc, &argv );

    /* Create new GtkBuilder object */
    builder = gtk_builder_new();
    if( ! gtk_builder_add_from_file( builder, UI_FILE, &error ) )
    {
```

```
        g_warning( "%s", error->message );
        g_free( error );
        return( 1 );
    }

    /* Allocate data structure */
    data = g_slice_new( ChData );

    /* Get objects from UI */
#define GW( name ) CH_GET_WIDGET( builder, name, data )
    GW( main_window );
    GW( chart_area );
#undef GW

    /* Connect signals */
    gtk_builder_connect_signals( builder, data );

    /* Destroy builder, since we don't need it anymore */
    g_object_unref( G_OBJECT( builder ) );

    /* Show window. All other widgets are automatically shown by GtkBuilder */
    gtk_widget_show( data->main_window );

    /* Start main loop */
    gtk_main();

    /* Free any allocated data */
    g_slice_free( ChData, data );

    return( 0 );
}
```

This file is almost exactly the same as tut.c file from second part of this tutorial. One important difference is creation of data structure that is passed to all of the callbacks.

Last file is callbacks.c. We'll be adding callback handlers into this file as we connect them inside Glade. There is only one function that we need to write: cb_expose_chart. But before we start writing anything, we need to check what prototype expose handler should have. Navigate to GtkWidget API reference and search for "expose-event" signal. (Or, if you're too lazy to search for yourself, click this link [http://library.gnome.org/devel/gtk/stable/GtkWidget.html#GtkWidget-expose-event] .)

In reference, you'll see that expose event handler should be defined as:

```
gboolean
function( GtkWidget      *widget,
          GdkEventExpose *event,
          gpointer        user_data )
```

Having this knowledge, we can now create callback file.

```
#include "support.h"
```

```
G_MODULE_EXPORT gboolean
cb_expose_chart( GtkWidget      *widget,
                 GdkEventExpose *event,
                 ChData         *data )
{
    cairo_t *cr;

    /* Create cairo context from GdkWindow */
    cr = gdk_cairo_create( event->window );

    /* Paint whole area in green color */
    cairo_set_source_rgb( cr, 0, 1, 0 );
    cairo_paint( cr );

    /* Destroy cairo context */
    cairo_destroy( cr );

    /* Return TRUE, since we handled this event */
    return( TRUE );
}
```

You can see that we added G_MODULE_EXPORT in front of function definition. We also modified last parameter to avoid some casting. You can ignore function body for now, since cairo drawing will be explained more when we'll start drawing our charts.

Compiling application

Last thing we need to do is compile application. This will in our case be done in three steps:

1. compile charter.c into charter.o
2. compile callbacks.c into callbacks.o
3. link both object files into charter

This translates into terminal command lines:

```
gcc -c charter.c -o charter.o $(pkg-config --cflags gtk+-2.0 gmodule-2.0)
gcc -c callbacks.c -o callbacks.o $(pkg-config --cflags gtk+-2.0 gmodule-2.0)
gcc -o charter charter.o callbacks.o $(pkg-config --libs gtk+-2.0 gmodule-2.0)
```

Since this can be quite demanding to write for each recompilation, I created Makefile that should make things a bit easier.

```
CC     = gcc
CFLAGS = `pkg-config --cflags gtk+-2.0 gmodule-2.0`
LIBS   = `pkg-config --libs   gtk+-2.0 gmodule-2.0`
DEBUG  = -Wall -g

OBJECTS = charter.o callbacks.o

.PHONY: clean

all: charter
```

```
charter: $(OBJECTS)
    $(CC) $(DEBUG) $(LIBS) $(OBJECTS) -o $@

charter.o: charter.c support.h
    $(CC) $(DEBUG) $(CFLAGS) -c $< -o $@

callbacks.o: callbacks.c support.h
    $(CC) $(DEBUG) $(CFLAGS) -c $< -o $@

clean:
    rm -f *.o charter
```

Consult you make manual for detailed information about this file. And we're done for today.

Getting sources

This is something new. I created a git repository at Gtihub.com to host files from this tutorial. I'll pack a code from each part of the tutorial and upload it on Github.com, but if you prefer to use git for obtaining code, I'll also tag commits. Here is my Github repository [http://github.com/tadeboro/Glade3-tutorial] and here are files for this tutorial part [http://cloud.github.com/downloads/tadeboro/Glade3-tutorial/glade_tut_06.tar.bz2] .

I also started converting posts from this tutorial into a DocBook format, which should hopefully make this tutorial more flexible and available in more formats.

Stay healthy.

Posted 28th September 2009 by Tadej Borovšak

Labels: Glade, GTK+, tutorial

22 View comments

Anonymous 28 September, 2009 01:52

very nice, thank you. and to connect more signals, is it correct to call gtk_builder_connect_signals() multiple successive times? like:
gtk_builder_connect_signals(builder, data);
gtk_builder_connect_signals(builder, window3);
etc.
?

Reply

tadeboro 28 September, 2009 09:23

Hi.

gtk_builder_connect_signals will connect handlers to all of the signals you set with glade, so there is no need to call this function more than once on a single GtkBuilder object.

I guess you would like to send different data to different signals. This cannot be done using **gtk_builder_connect_signals**. You'll either have to use **gtk_builder_connect_signals_full**, which will give you total control over the connection process, or manually connect signals whose handlers should get special data.

Tadej

Reply

Cenwen 28 September, 2009 22:17

Hi you never stop for my more bigf pleasure. a wonderfull tutorial yet. i'm only the 3 rd. Imagination is a beautifull software but could you explain in another post how do you do on this logiciel to realise two tabs and the rest like at right on the top ?

Reply

Todor 07 November, 2009 01:38

On both Ubuntu Jaunty and Karmik G_MODULE_EXPORT - as found in the header gmodule.h - expands exactly to nothing, unless you are linking on Win32/64. I have an action, which in the glade file is connected to on_actionAbout_activate and defining it as:

G_MODULE_EXPORT void on_actionAbout_activate(GtkAction* action, gpointer data) { ... }

gives the error:

(app:PID): Gtk-WARNING **: Could not find signal handler 'on_actionAbout_activate'

with or without G_MODULE_EXPORT in front. I needed to pass the option *-Wl,--export-dynamic* to the linker for it to work.

Reply

tadeboro 07 November, 2009 09:03

@Todor:

G_MODULE_EXPORT is indeed only needed on Windows platform, but in order to make your code portable, I would advice to use it whenever you write your code to be used with GtkBuilder (on GNU/Linux, preprocessor will strip it out, so no overhead is added to final binary).

-Wl, --export-dynamic is only needed on GNU/Linux, Windows platforms will do without it. But manually adding this is not portable, this is why I suggest to add "gmodule-2.0" to pkg-config, since it'll provide proper linker flags for your platform.

Tadej

Reply

Todor 08 November, 2009 05:22

Hmm, interesting! Haven't caught that one myself...

pkg-config --cflags --libs gtk+-2.0 links in gmodule-2.0 automatically, but without the linker flag??? My output is:

-D_REENTRANT -I/usr/include/gtk-2.0 -I/usr/lib/gtk-2.0/include -I/usr/include/atk-1.0 -I/usr/include/cairo -I/usr/include/pango-1.0 -I/usr/include/pixman-1 -I/usr/include/freetype2 -I/usr/include/directfb -I/usr/include/libpng12 -I/usr/include/glib-2.0 -I/usr/lib/glib-2.0/include -lgtk-x11-2.0 -lgdk-x11-2.0 -latk-1.0 -lpangoft2-1.0 -lgdk_pixbuf-2.0 -lm -lpangocairo-1.0 -lgio-2.0 -lcairo -lpango-1.0 -lfreetype -lfontconfig -lgobject-2.0 -lgmodule-2.0 -lglib-2.0

Only explicitly adding gmodule-2.0 adds the *-Wl,--export-dynamic* linker flag.

 # pkg-config --cflags --libs gmodule-2.0
-I/usr/include/glib-2.0 -I/usr/lib/glib-2.0/include -Wl,--export-dynamic -lgmodule-2.0 -lglib-2.0

And BTW and OT: thanks for answering my other comment on Imagination about the custom cell renderer, you/it helped a lot! Now I have a thumbnail widget which is checkable, a.k.a a permanent selection :-)

Cheers!

Reply

Slartius 13 November, 2009 14:10

Hi, nice tutorial.

In your tutorial (and also in files created with glade-2, the callback functions for events have the form of:
cb_event (GtkWidget *widget, GdkEvent *event, gpointer user_data).

But when I did some testing using glade-3 I got 1st and 3rd parameters switched.
For instance, for button press event in drawing area I used this drawing area as user data, while for button release event I used the main window. In both callbacks I print out the sizes of the calling widget and the one, passed with the user data.
In both cases the size of the widget passed as user data is displayed first and of the calling widget second, which is exactly vice versa as it should be.

I was wondering if maybe you have an explanation for this strange behavior, or in other words, what am I doing wrong. And by the way, I was using gtk_builder_connect_signals(...) function.

Slatrius

Reply

tadeboro 13 November, 2009 18:21

Hello.

If you look again at second paragraph under **Signals tab in Glade** section, you'll notice this: *note: if field is not empty, signal will be connected the same way as if you were to call g_signal_connect_swapped macro*. And g_signal_connect_swapped does exactly what you experience: swap user data and widget parameter.

Now things should make sense;)

Tadej

Reply

Marcos 09 March, 2010 23:41

So, what is this "Data User" column for? I only managed to get a Segmentation Fault when is set.

Reply

ChristianCDMC 04 April, 2010 18:13

hi tadeboro, i have a problem 'cause i cant do anythig with the application after i compiled, form example when i press any button nothing hapend. coul you help me please-

thanks

Reply

Anonymous 22 June, 2010 05:30

If you copy the source code using the 'copy to clipboard' link in the code block, the resulting Makefile will fail to compile with an error "Makefile:13: *** missing separator. Stop."

This is because there needs to be an actual tab instead of spaces at the start of the $(CC) and rm -f lines.

Once I changed these to tabs instead of spaces, this compiled correctly.

(Just a note in case anyone else was having problems.)

Reply

Replies

Anonymous 30 October, 2012 05:31

of course this is ancient :p but yesh.. thanks a lot.. I probably would have been hacking away senselessly without that lil tip. *Thinks of genie*

but on second compile line I get

gcc -Wall -g `pkg-config --cflags gtk+-2.0 gmodule-2.0` -c callbacks.c -o callbacks.o
callbacks.c:1:1: error: unknown type name ‘gboolean’
callbacks.c:2:11: error: unknown type name ‘GtkWidget’
callbacks.c:3:11: error: unknown type name ‘GdkEventExpose’
callbacks.c:4:11: error: unknown type name ‘gpointer’
make: *** [callbacks.o] Error 1

(not the makefile but the gcc command by itself) but ok, I just did it now without troubleshooting.

Reply

Bob Hazard 08 July, 2010 03:45

This tutorial is great you should do more of them, especially screencasts.

I have converted the C code into Vala and put a link to to your blog here:

http://vala.posterous.com/using-gtkbuilder-to-load-a-gui-created-by-gla

Reply

Replies

Anonymous 30 October, 2012 05:33

yay .. thanks.. I am trying to learn vala *would actually love to give genie a go but not enough documentation*.

So much appreciated... and so is this tutorial. Thank you very much mister borovsak. I did java programming back in last century and am a tad 'frightened' when approaching new

things but this was useful as hell. :)

Reply

Anonymous 27 July, 2010 15:27

Hi
if I have two glade files, both glade files also has the same event name "on_button_clicked", does it causes an ambiguous function call?

Reply

അപരൻ 11 November, 2010 16:02

This tutorial is awesome. Thanks a lot bro...
Stay healthy...

Reply

Anonymous 06 December, 2010 20:10

important: if you compile Glade3 with C++, you have to add ' extern "C" ' before the declaration of the callback function. Like this:

extern "C" void
on_button4_clicked (GtkButton *button, gpointer user_data)
{

Reply

Anonymous 30 September, 2011 14:11

I have been following several glade tutorials, and so far haven't been able to successfully connect the signals with gtk_builder_connect_signals.

At first I was using Eclipse CDT so I separated all --cflags into individual lines, deleted all instances of -I and then pasted the include paths to Project -> Properties -> C/C++ Build -> Settings -> GCC C Compiler -> Includes, and added -pthread to Miscellaneous options, just in case. This is how all options looked like:

-I/usr/include/atk-1.0 -I/usr/include/pango-1.0 -I/usr/include/gio-unix-2.0/ -I/usr/include/glib-2.0 -I/usr/include/freetype2 -I/usr/include/libpng12 -I/usr/include/gtk-2.0 -I/usr/lib/gtk-2.0/include -I/usr/include/cairo -I/usr/include/gdk-pixbuf-2.0 -I/usr/lib/i386-linux-gnu/glib-2.0/include -I/usr/include/pixman-1 -O0 -g3 -Wall -c -fmessage-length=0 -pthread

A little messy, but it compiles, after I did the same for the --libs output (only this was for linker options, of course). This is how all options look like for linker:

-pthread -Wl,--export-dynamic

It didn't list all the -L and -l switches, but since it links with all --libs output -Lxxxx and -Lxxxx added and doesn't if I delete them, I persume they are just not being displayed.

And when I compile the project, I get the dreaded Gtk-WARNING **: Could not find signal handler. The I tried manually compiling and linking.

$ gcc -c drag_drop.c -o drag_drop.o `pkg-config --cflags gtk+-2.0 gmodule-2.0`

$ gcc -o drag_drop drag_drop.o `pkg-config --libs gtk+-2.0 gmodule-2.0`

But it's the same old story:
$ ./drag_drop

(drag_drop:7238): Gtk-WARNING **: Could not find signal handler 'on_main_window_destroy'

The callback function is named in Glade as:
on_main_window_destroy

It is declared in drag_drop.h as:
static void on_main_window_destroy (GtkObject*, gpointer);

and is defined right after main() as:
static void on_main_window_destroy (GtkObject *object, gpointer userdata) { ...

It's all the same if I just define it before main() or declare it in drag_drop.c (I believe I'm using the right terms for declaration and definition, I might be wrong though) - the gmodule just can't find my function...

If anyone knows where is the problem, please let me know.
(I'm on a Ubuntu 11.04 machine).

The command used to obtain CFLAGS variables:
$ pkg-config --cflags gtk+-2.0 gmodule-2.0

The command used to obtain needed libraries:
$ pkg-config --libs gtk+-2.0 gmodule-2.0

Reply

Anonymous 30 September, 2011 14:28

Never mind that, I just needed to omit 'static' keyword from callback function definition.

Reply

Anonymous 12 December, 2012 03:27

great tutorial!
Followed all the way through, except the last step: cb_expose_chart since my Glade 3.12 (and GTK3) does not show the mentioned event "expose-event" in the signal section of chart_area. I guess this part is just due to version change, and I have enough to start reading API. Thanks!

Reply

Anonymous 23 December, 2012 17:39

Can anyone explain why there are definitely loss and 1078 errors?

==4497== LEAK SUMMARY:
==4497== definitely lost: 4,156 bytes in 9 blocks
==4497== indirectly lost: 11,744 bytes in 368 blocks
==4497== possibly lost: 618,127 bytes in 3,897 blocks
==4497== still reachable: 846,489 bytes in 6,480 blocks
==4497== suppressed: 0 bytes in 0 blocks
==4497== Reachable blocks (those to which a pointer was found) are not shown.
==4497== To see them, rerun with: --leak-check=full --show-reachable=yes
==4497==
==4497== For counts of detected and suppressed errors, rerun with: -v

==4497== ERROR SUMMARY: 1078 errors from 1054 contexts (suppressed: 2 from 2)

Reply

Anonymous 25 May, 2013 21:40

Thank you for this nice tutorial.
I really enjoyed your casual writing style.

Reply

Enter your comment...

Comment as: Google Accou ▼

Publish **Preview**