29th April 2009

GtkDialog tutorial - part 2

Hello everybody!

Creating second part of this tutorial took a little longer than I anticipated to create, main reason being nasty little bug in Glade3-3.6.x, which produced some "interesting" results. You can find more details here [http://bugzilla.gnome.org/show_bug.cgi?id=580745] .

Contents:

Build using Glade

Now let's create this application already. As usual, try to create glade GUI, based on the code from last post, yourself first (remember, "Repetitio Est ...";), and only then check this video out [http://www.screentoaster.com/watch/stU0tWQk1IR11eQFxUWFNRUVdV/gtkdialog_and_glade] . Yes, you read it right, video;)

Here is the outline of the video (for those who cannot watch it):

- Create main window
  - set border property to 10
  - connect delete-event signal to cb_delete_event
  - connect destroy signal to gtk_main_quit
  - add button
    - set it's stock id to gtk-about
    - connect clicked signal to cb_show_about callback
- Create dialog
  - Add label
    - Set it's label property to "Are you sure you want to quit?
    - Make it modal.
    - Make it transient for main widow.
  - Add button
    - Set it's stock id to gtk-yes
    - Set it's response id to 1
  - Add button
    - Set it's stock id to gtk-no
    - Set it's response id to 2
- Create about dialog
  - Set programe name property to Sample Dialog App
  - Set version property to 0.2
  - Set copyright string to something that suits you
  - Set website URL to tadeboro.blogspot.com

You can download builder file from here [http://tadeboro.googlepages.com/dialog.builder] . WARNING: if

you're using glade3-3.6.x, you should manually rename about button to something like about. See my bug report I mentioned at the beginning of the post for more details. (My uploaded glade project is already fixed.)

What do you think for the first time? What is left for us to do is to write some code to support the GUI. First, we need to load GUI. We did that a few times before so you should be familiar with them by now.

```
    /* Create builder and load interface */
    builder = gtk_builder_new();
    gtk_builder_add_from_file( builder, "dialog.builder", NULL );

    /* Obtain widgets that we need */
    window = GTK_WIDGET( gtk_builder_get_object( builder, "window1" ) );
    data.quit = GTK_WIDGET( gtk_builder_get_object( builder, "dialog1" ) );
    data.about = GTK_WIDGET( gtk_builder_get_object( builder, "aboutdialog1" ) );

    /* Connect callbacks */
    gtk_builder_connect_signals( builder, &data );

    /* Destroy builder */
    g_object_unref( G_OBJECT( builder ) );
```

```
    def __init__( self ):
        builder = gtk.Builder()
        builder.add_from_file( "dialog.builder" )

        self.window       = builder.get_object( "window1" )
        self.about_dialog = builder.get_object( "aboutdialog1" )
        self.quit_dialog  = builder.get_object( "dialog1" )

        builder.connect_signals( self )
```

Writting callback is piece of cake this time, since all that we need to do is to run the right dialog and hide it after use.

```
G_MODULE_EXPORT gboolean
cb_delete_event( GtkWidget *window,
                 GdkEvent  *event,
                 Data      *data )
{
    gint response = 1;

    /* Run dialog */
    response = gtk_dialog_run( GTK_DIALOG( data->quit ) );
    gtk_widget_hide( data->quit );

    return( 1 != response );
}

G_MODULE_EXPORT void
cb_show_about( GtkButton *button,
               Data      *data )
```

```
{
    /* Run dialog */
    gtk_dialog_run( GTK_DIALOG( data->about ) );
    gtk_widget_hide( data->about );
}
```

```
    def cb_delete_event( self, window, event ):
        # Run dialog
        response = self.quit_dialog.run()
        self.quit_dialog.hide()

        return response != 1

    def cb_show_about( self, button ):
        # Run dialog
        self.about_dialog.run()
        self.about_dialog.hide()
```

And that's it! The whole application listings are posted at the end.

I hope you'll join me next time when we'll be tackling ... hmm, I don't know what yet. Suggestions are welcome.

Bye

```
/*
 * Compile me with:
 *  gcc -o dialog1 dialog1.c $(pkg-config --cflags --libs gtk+-2.0 \
 *        gmodule-export-2.0)
 */

#include <gtk/gtk.h>

typedef struct _Data Data;
struct _Data
{
    GtkWidget *quit;
    GtkWidget *about;
};

G_MODULE_EXPORT gboolean
cb_delete_event( GtkWidget *window,
                 GdkEvent  *event,
                 Data      *data )
{
    gint response = 1;

    /* Run dialog */
    response = gtk_dialog_run( GTK_DIALOG( data->quit ) );
    gtk_widget_hide( data->quit );
```

```
    return( 1 != response );
}

G_MODULE_EXPORT void
cb_show_about( GtkButton *button,
               Data      *data )
{
    /* Run dialog */
    gtk_dialog_run( GTK_DIALOG( data->about ) );
    gtk_widget_hide( data->about );
}

int
main( int    argc,
      char **argv )
{
    GtkBuilder *builder;
    GtkWidget  *window;
    Data        data;

    gtk_init( &argc, &argv );

    /* Create builder and load interface */
    builder = gtk_builder_new();
    gtk_builder_add_from_file( builder, "dialog.builder", NULL );

    /* Obtain widgets that we need */
    window = GTK_WIDGET( gtk_builder_get_object( builder, "window1" ) );
    data.quit = GTK_WIDGET( gtk_builder_get_object( builder, "dialog1" ) );
    data.about = GTK_WIDGET( gtk_builder_get_object( builder, "aboutdialog1" ) );

    /* Connect callbacks */
    gtk_builder_connect_signals( builder, &data );

    /* Destroy builder */
    g_object_unref( G_OBJECT( builder ) );

    /* Show main window and start main loop */
    gtk_widget_show( window );
    gtk_main();

    return( 0 );
}
```

```
#!/usr/bin/env python
# vim: set fileencoding=utf-8

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

```
class Sample:
    def gtk_main_quit( self, window ):
        gtk.main_quit()

    def cb_delete_event( self, window, event ):
        # Run dialog
        response = self.quit_dialog.run()
        self.quit_dialog.hide()

        return response != 1

    def cb_show_about( self, button ):
        # Run dialog
        self.about_dialog.run()
        self.about_dialog.hide()

    def __init__( self ):
        builder = gtk.Builder()
        builder.add_from_file( "dialog.builder" )

        self.window       = builder.get_object( "window1" )
        self.about_dialog = builder.get_object( "aboutdialog1" )
        self.quit_dialog  = builder.get_object( "dialog1" )

        builder.connect_signals( self )

if __name__ == "__main__":
    win = Sample()
    win.window.show_all()
    gtk.main()
```

Posted 29th April 2009 by Tadej Borovšak

Labels: GTK+, GtkDialog, tutorial

16 View comments

Anonymous 09 December, 2009 12:25

How is this tutorial useful in anyway ?
I want c code from Glade app and not some crap what i need to do now, why the hell did i even bother to use the gui to build something when i need to do it all manually?????????????

crappy software.....

Reply

tadeboro 09 December, 2009 12:38

@Anonymous: Hello.

I'm sorry if you feel this way, but this is how things work when creating GUI applications using Glade3. Besides, even if Glade would be able to produce some C code for you, you would still need to write majority of code yourself. GUI builders can only create function declarations and provide you with a

skeleton that you need to fill in.

Glade3 sacrificed convenience of creating function declarations in favor of increased maintainability. After all, function declarations are written only once while application needs to be maintained over a longer period of time. Having GUI in a separate file that can be fixed without recompiling your code is really nice.

Tadej

Reply

John 06 January, 2010 15:02

Tutorial was very helpful, thank you!

Big frowns to the guy who wrote the first comment- this guy writes a tutorial and you flame him? Hmm, not impressed.

Again, thanks for this tutorial it has been useful to me! :)

John

Reply

Jack 30 January, 2010 20:12

Here is what I have when using Glade with simple window, label, text entry and button:

```
window
.vbox
..hbox
...label
...entry
..button

/*****************************************************************/
/*****************************************************************/
// FreeBSD 8.0
// Xfce 4.6.1
// GTK 2.16.6
// Glade 3.6.7
//
// gcc `pkg-config --cflags --libs gtk+-2.0` -export-dynamic -o hello hello.c
/*****************************************************************/

#include "gtk/gtk.h"

/*****************************************************************/

#define GLADE
// comment out this option to do it without Glade xml file

/*****************************************************************/

// event handlers must be defined regardless
// if their gadgets are generated in code or Glade

G_MODULE_EXPORT void on_window1_destroy( GtkWidget *widget, gpointer data )
{
gtk_main_quit ();
```

```
}

G_MODULE_EXPORT void on_button1_clicked( GtkWidget *widget, gpointer data )
{
g_print ("Hello!\n");
}

/*************************************************************/

int main( int argc, char *argv[] )
{
gtk_init (&argc, &argv);

#ifdef GLADE
GtkBuilder *builder = gtk_builder_new();
gtk_builder_add_from_file( builder, "hello.glade", NULL );
// need to open the resource

GtkWindow *window1 = GTK_WINDOW( gtk_builder_get_object( builder, "window1" ) );
// need to get reference to main widow
// but no packing individual gadgets
#else
GtkWidget *window1 = gtk_window_new( GTK_WINDOW_TOPLEVEL );
GtkWidget *box1 = gtk_vbox_new( FALSE, 0 );
GtkWidget *box2 = gtk_hbox_new( FALSE, 0 );
GtkWidget *label1 = gtk_label_new( "Enter Name" );
GtkWidget *entry1 = gtk_entry_new();
GtkWidget *button1 = gtk_button_new_with_label( "Hello" );
// this list counld be quite long ...
gtk_container_add( GTK_CONTAINER (window1), box1 );
gtk_box_pack_start( GTK_BOX(box1), box2, TRUE, TRUE, 10 );
gtk_box_pack_start( GTK_BOX(box2), label1, TRUE, TRUE, 10 );
gtk_box_pack_start( GTK_BOX(box2), entry1, TRUE, TRUE, 10 );
gtk_box_pack_start( GTK_BOX(box1), button1, TRUE, TRUE, 10 );
// this list counld be quite long ...
#endif

#ifdef GLADE
gtk_builder_connect_signals( builder, NULL );
// connect all handlers via single call
#else
g_signal_connect( G_OBJECT (window1), "destroy", G_CALLBACK (on_window1_destroy), NULL );
g_signal_connect( G_OBJECT (button1), "clicked", G_CALLBACK (on_button1_clicked), NULL );
// this list counld be quite long ...
#endif

#ifdef GLADE
g_object_unref( G_OBJECT( builder ) );
// one line cleanup for builder
#endif

gtk_widget_show_all( window1 );

gtk_main ();

return 0;
}

/*************************************************************/
/*************************************************************/
```

As far as I see there is an advantage of using Glade vs coding all the gadgets by hand. I think the guy above is frustrated because Glade and GTK are not self explanatory or self guided tools like Visual Studio for VB or NetBeans for Java. However, once mastered, Glade/GTK can lead to lean and easy to maintain code.

Jack

Reply

Max 15 March, 2010 21:56

Very helpful, thanks a lot!!
One question: if I had to redistribute an app made through GtkBuilder, should I redistribute the XML file together with the binaries?

Reply

Anonymous 28 April, 2010 21:06

Thank you!

Reply

Zach 16 June, 2010 02:28

The tutorial and code was a great help. Can't thank you enough!

Reply

Michele 28 August, 2010 05:53

Thank you so much! This is a really nice tutorial, just what I needed to get started!

The web is a nice experience because of people like you :-)

Reply

Anonymous 01 December, 2010 01:46

I really like the way you react to the first comment; And the tutorial is great as always.

Reply

Anonymous 23 December, 2010 16:59

The video is not viewable. Just says buffering, no matter how long I wait.

Pity, I can not figure out how to do the UI with glade reading only the text.

Reply

@samwhiteUK 12 September, 2011 22:54

@Anonymous - you're a twat.

Reply

Anonymous 23 November, 2011 21:43

I can't see the video: access denied, forbidden.

Where can I see the video??

Elmar

Reply

Umpirsky 10 December, 2011 00:49

Can we somehow use constants http://developer.gnome.org/pygtk/stable/gtk-constants.html#gtk-response-type-constants instead 1, 2 integers? How can we use them from Glade?

Reply

ClouD and Tifa 17 January, 2013 13:52

Hi there. Can you show me how i make the dialog appear when i click on a menu item? I want the menu about to show me a dialog i created. I can't make it work :(

Reply

Anonymous 07 February, 2014 02:44

i cant add label/button in GtkDialog.... please tell me how to do that?

Reply

Anonymous 13 May, 2014 19:05

Thanks very much for putting this tutorial up. It's a very good introduction :)

Your work is appreciated.

Reply

Enter your comment...

Comment as: Google Accou ▼

Publish Preview