25th April 2009

GtkDialog tutorial - part 1

Hello everybody and welcome back!

In next two posts we'll be looking at GtkDialog widget. Today's part will be dealing with manual dialog creation, which will allow us to get a good grip at GtkDialog's internal workings. In second part, we'll try to use our newly acquired knowledge to recreate sample application using glade interface builder.

Contents:

Manual creation

A good place to start learning about GtkDialog is description section of the API reference [http://library.gnome.org/devel/gtk/stable/GtkDialog.html#GtkDialog.description] . It'll give you a nice and concise overview of the dialog and make it easier to follow this post.

Done reading API docs? OK, let's move on. Sample application that we'll develop for this tutorial will consist of main window with one "About" button, about dialog that will be shown when we click "About" button and quit conformation dialog, which will ask us if we really want to quit. Now let's start coding.

First thing that we need to create is our main window and connect "delete-event" and "destroy" signals; add "About" stock button to window and connect "clicked" signal. Try writing it yourself before looking at my sample code, since we're trying to learn here (remember, "Repetitio Est Mater Studiorum").

```
int
main( int    argc,
      char **argv )
{
    GtkWidget *window;
    GtkWidget *button;

    gtk_init( &argc, &argv );

    /* Create main window */
    window = gtk_window_new( GTK_WINDOW_TOPLEVEL );
    gtk_container_set_border_width( GTK_CONTAINER( window ), 10 );

    /* Connect delete event signal - this is where we'll show our
     * quit conformation dialog. */
    g_signal_connect( G_OBJECT( window ), "delete-event",
                      G_CALLBACK( cb_delete_event ), NULL );

    /* Connect destroy signal, which will be emitted if we return
     * FALSE from our cb_delete_event function. */
    g_signal_connect( G_OBJECT( window ), "destroy",
                      G_CALLBACK( gtk_main_quit ), NULL );
```

```
    /* Create button ... */
    button = gtk_button_new_from_stock( GTK_STOCK_ABOUT );

    /* ... and connect clicked signal to handler that will create
     * and show about dialog. */
    g_signal_connect( G_OBJECT( button ), "clicked",
                      G_CALLBACK( cb_show_about ), window );
    gtk_container_add( GTK_CONTAINER( window ), button );

    /* Show our main window and start main loop. */
    gtk_widget_show_all( window );
    gtk_main();

    return( 0 );
}
```

```
class Sample( gtk.Window ):

    # More code here

    def __init__( self ):
        # Create main window
        gtk.Window.__init__( self )
        self.set_border_width( 10 )

        # Connect delete event signal - this is where we'll show our
        # quit conformation dialog.
        self.connect( 'delete-event', self.cb_delete_event )

        # Connect destroy signal, which will be emitted if we return
        # False from our self.cb_delete_event method. */
        self.connect( 'destroy', lambda *w: gtk.main_quit() )

        # Create button ...
        button = gtk.Button( None, gtk.STOCK_ABOUT )

        # ... and connect clicked signal to handler that will create
        # and show about dialog.
        button.connect( 'clicked', self.cb_show_about )
        self.add( button )

        # Add "placeholders" for dialogs
        self.about_dialog = None
        self.quit_dialog  = None

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

Now, we'll write delete event handler and create a conformation dialog. This dialog will be very simple, with label saying "Are you sure you want to quit?" and two buttons, "Yes" and "No". I'll show you how to

create dialog step-by-step and by using convenience function.

First, we need to create new dialog with gtk_dialog_new (gtk.Dialog), make it modal and transient for main window and set it's title to "Conformation". Now we need to add two buttons to the dialog with gtk_dialog_add_button function (add_button method). If you're looking at the API docs for this function now (and if you don't, you should from now on;), you see that the last parameter is response id. This value will be returned from gtk_dialog_run function (run method) and will enable us to determine which button has been pressed. We'll assign 1 to "Yes" button and 2 to "No" button. Last thing we need to do is to add label to content area of the widget. And this is where things get a little tricky, since some incompatible changes have been made between 2.12 and 2.14 version of gtk. If we're using gtk+-2.12 or less, we need to access content area directly like a struct member in C and like a attribute in Python, and if our gtk+ version is 2.14 or more, we need to use accessor function gtk_dialog_get_content_area (get_content_area method). See code for more details.

To finish our callback function, we'll write an if statement which will return false if user clicked "Yes" and true otherwise.

```
static gboolean
cb_delete_event( GtkWidget *window,
                 GdkEvent  *event,
                 gpointer   data )
{
    /* We'll create dialog only the first time we enter this callback.
     * After first time, we'll just show it. */
    static GtkWidget *dialog = NULL;
    gint              response;

    if( ! dialog )
    {
        GtkWidget *label;
        GtkWidget *box;
#if 1
        /* Create dialog */
        dialog = gtk_dialog_new();

        /* Set it modal and transient for main window. */
        gtk_window_set_modal( GTK_WINDOW( dialog ), TRUE );
        gtk_window_set_transient_for( GTK_WINDOW( dialog ),
                                      GTK_WINDOW( window ) );

        /* Set title */
        gtk_window_set_title( GTK_WINDOW( dialog ), "Conformation" );

        /* Add buttons. */
        gtk_dialog_add_button( GTK_DIALOG( dialog ), GTK_STOCK_YES, 1 );
        gtk_dialog_add_button( GTK_DIALOG( dialog ), GTK_STOCK_NO,  2 );

#else

        /* If we use convenience API function gtk_dialog_new_with_buttons,
         * last six function calls can be written as: */
```

```
        dialog = gtk_dialog_new_with_buttons( "Conformation",
                                              GTK_WINDOW( window ),
                                              GTK_DIALOG_MODAL,
                                              GTK_STOCK_YES, 1,
                                              GTK_STOCK_NO,  2,
                                              NULL );
#endif

        /* Create label */
        label = gtk_label_new( "Are you sure you want to quit?" );

        /* Pack label, taking API change in account. */
#if GTK_MINOR_VERSION < 14
        box = GTK_DIALOG( dialog )->vbox;
#else
        box = gtk_dialog_get_content_area( GTK_DIALOG( dialog ) );
#endif
        gtk_box_pack_start( GTK_BOX( box ), label, TRUE, TRUE, 0 );

        /* Show dialog */
        gtk_widget_show_all( dialog );
    }

    /* Run dialog */
    response = gtk_dialog_run( GTK_DIALOG( dialog ) );
    gtk_widget_hide( dialog );

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

```
    def cb_delete_event( self, window, event ):
        # If dialog does not exists, create it
        if self.quit_dialog == None:
            # Create dialog
            self.quit_dialog = gtk.Dialog()

            # Set it modal and transient for main window.
            self.quit_dialog.set_modal( True )
            self.quit_dialog.set_transient_for( self )

            # Set title
            self.quit_dialog.set_title( 'Conformation' )

            # Add buttons.
            self.quit_dialog.add_button( gtk.STOCK_YES, 1 )
            self.quit_dialog.add_button( gtk.STOCK_NO,  2 )

            # Using non-null parameter list when creating dialog,
            # the last six calls can be written as:
            # self.quit_dialog = gtk.Dialog( 'Conformation', self,
            #                                gtk.DIALOG_MODAL,
```

```
            #                                   ( gtk.STOCK_YES, 1,
            #                                     gtk.STOCK_NO,  2 ) )

            # Create label
            label = gtk.Label( 'Are you sure you want to quit?' )

            # Pack label, taking API change in account
            if gtk.pygtk_version[1] < 14:
                self.quit_dialog.vbox.pack_start( label )
            else:
                self.quit_dialog.get_content_area().pack_start( label )

            # Show dialog
            self.quit_dialog.show_all()

        # Run dialog
        response = self.quit_dialog.run()
        self.quit_dialog.hide()

        return response != 1
```

Now we'll create about dialog and callback function for about button. This function will be quite similar to the delete event handler, but instead of plain GtkDialog, we'll use GtkAboutDialog, which will save us some time we would use packing different widgets inside content area. Again, I wrote code that shows you how to manually set all the options and how to use convenience function. And this is the code:

```
static void
cb_show_about( GtkButton *button,
               GtkWidget *window )
{
#if 1
    /* This is our about dialog, declared static to avoid recreating it
     * each time we click about button. */
    static GtkWidget *dialog = NULL;

    if( ! dialog )
    {
        /* This is just a convenience to avoid castings. */
        GtkAboutDialog *about;
        const gchar    *auth[] = { "Tadej Borovšak <tadeboro@gmail.com>",
                                   NULL };

        /* Create dialog */
        dialog = gtk_about_dialog_new();
        gtk_window_set_transient_for( GTK_WINDOW( dialog ),
                                      GTK_WINDOW( window ) );
        about = GTK_ABOUT_DIALOG( dialog );

        /* Set it's properties */
        gtk_about_dialog_set_program_name( about, "Sample Dialog App" );
        gtk_about_dialog_set_version( about, "0.1" );
        gtk_about_dialog_set_copyright( about,
```

```
                                        "Copyright 2009 © Tadej Borovšak" );
        gtk_about_dialog_set_website( about, "http://tadeboro.blogspot.com" );
        gtk_about_dialog_set_authors( about, auth );

        /* Show dialog */
        gtk_widget_show_all( dialog );
    }

    /* Run dialog and hide it after it returns. */
    gtk_dialog_run( GTK_DIALOG( dialog ) );
    gtk_widget_hide( dialog );

#else

    /* This is how creating about dialog using gtk_show_about_dialog
     * convenience function looks like. */
    const gchar *auth[] = { "Tadej Borovšak <tadeboro@gmail.com>", NULL };

    gtk_show_about_dialog( GTK_WINDOW( window ),
                           "program-name", "Sample Dialog App",
                           "version", "0.1",
                           "copyright", "Copyright 2009 © Tadej Borovšak",
                           "website", "http://tadeboeo.blogspot.com",
                           "authors", auth,
                           NULL );
#endif
}
```

```
    def cb_show_about( self, button ):
        if self.about_dialog == None:
            # Create about dialog
            self.about_dialog = gtk.AboutDialog()
            self.about_dialog.set_transient_for( self )

            # Set dialog's properties
            self.about_dialog.set_program_name( "Sample Dialog App" )
            self.about_dialog.set_version( "0.1" )
            self.about_dialog.set_copyright( "Copyright 2009 © Tadej Borovšak" )
            self.about_dialog.set_website( "http://tadeboro.blogspot.com" )
            self.about_dialog.set_authors(
                                [ "Tadej Borovšak <tadeboro@gmail.com>" ] )

            # Show dialog
            self.about_dialog.show_all()

        # Run dialog
        self.about_dialog.run()
        self.about_dialog.hide()

        # Convenience method for creating about dialog is wrapped,
        # but unfortunatelly I don't know how to use it.
```

```
        # API reference for PyGTK says nothing about gtk.show_about_dialog
        # Sorry for the inconveniance, Python coders.
```

And that would be everything for today. Join me next time when we'll recreate this application using glade. All I need to do now is to say "Bye" and paste the complete code. See ya;)

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

static void
cb_show_about( GtkButton *button,
               GtkWidget *window )
{
#if 1
    /* This is our about dialog, declared static to avoid recreating it
     * each time we click about button. */
    static GtkWidget *dialog = NULL;

    if( ! dialog )
    {
        /* This is just a convenience to avoid castings. */
        GtkAboutDialog *about;
        const gchar    *auth[] = { "Tadej Borovšak <tadeboro@gmail.com>",
                                   NULL };

        /* Create dialog */
        dialog = gtk_about_dialog_new();
        gtk_window_set_transient_for( GTK_WINDOW( dialog ),
                                      GTK_WINDOW( window ) );
        about = GTK_ABOUT_DIALOG( dialog );

        /* Set it's properties */
        gtk_about_dialog_set_program_name( about, "Sample Dialog App" );
        gtk_about_dialog_set_version( about, "0.1" );
        gtk_about_dialog_set_copyright( about,
                                        "Copyright 2009 © Tadej Borovšak" );
        gtk_about_dialog_set_website( about, "http://tadeboro.blogspot.com" );
        gtk_about_dialog_set_authors( about, auth );

        /* Show dialog */
        gtk_widget_show_all( dialog );
    }

    /* Run dialog and hide it after it returns. */
    gtk_dialog_run( GTK_DIALOG( dialog ) );
    gtk_widget_hide( dialog );

#else

    /* This is how creating about dialog using gtk_show_about_dialog
     * convenience function looks like. */
    const gchar *auth[] = { "Tadej Borovšak <tadeboro@gmail.com>", NULL };
```

```
    gtk_show_about_dialog( GTK_WINDOW( window ),
                           "program-name", "Sample Dialog App",
                           "version", "0.1",
                           "copyright", "Copyright 2009 © Tadej Borovšak",
                           "website", "http://tadeboeo.blogspot.com",
                           "authors", auth,
                           NULL );
#endif
}

static gboolean
cb_delete_event( GtkWidget *window,
                 GdkEvent  *event,
                 gpointer   data )
{
    /* We'll create dialog only the first time we enter this callback.
     * After first time, we'll just show it. */
    static GtkWidget *dialog = NULL;
    gint              response;

    if( ! dialog )
    {
        GtkWidget *label;
        GtkWidget *box;
#if 1
        /* Create dialog */
        dialog = gtk_dialog_new();

        /* Set it modal and transient for main window. */
        gtk_window_set_modal( GTK_WINDOW( dialog ), TRUE );
        gtk_window_set_transient_for( GTK_WINDOW( dialog ),
                                      GTK_WINDOW( window ) );

        /* Set title */
        gtk_window_set_title( GTK_WINDOW( dialog ), "Conformation" );

        /* Add buttons. */
        gtk_dialog_add_button( GTK_DIALOG( dialog ), GTK_STOCK_YES, 1 );
        gtk_dialog_add_button( GTK_DIALOG( dialog ), GTK_STOCK_NO,  2 );
#endif
        /* If we use convenience API function gtk_dialog_new_with_buttons,
         * last six function calls can be written as: */
#if 0
        dialog = gtk_dialog_new_with_buttons( "Conformation",
                                              GTK_WINDOW( window ),
                                              GTK_DIALOG_MODAL,
                                              GTK_STOCK_YES, 1,
                                              GTK_STOCK_NO,  2,
                                              NULL );
#endif
```

```
        /* Create label */
        label = gtk_label_new( "Are you sure you want to quit?" );

        /* Pack label, taking API change in account. */
#if GTK_MINOR_VERSION < 14
        box = GTK_DIALOG( dialog )->vbox;
#else
        box = gtk_dialog_get_content_area( GTK_DIALOG( dialog ) );
#endif
        gtk_box_pack_start( GTK_BOX( box ), label, TRUE, TRUE, 0 );

        /* Show dialog */
        gtk_widget_show_all( dialog );
    }

    /* Run dialog */
    response = gtk_dialog_run( GTK_DIALOG( dialog ) );
    gtk_widget_hide( dialog );

    return( 1 != response );
}

int
main( int    argc,
      char **argv )
{
    GtkWidget *window;
    GtkWidget *button;

    gtk_init( &argc, &argv );

    /* Create main window */
    window = gtk_window_new( GTK_WINDOW_TOPLEVEL );
    gtk_container_set_border_width( GTK_CONTAINER( window ), 10 );

    /* Connect delete event signal - this is where we'll show our
     * quit conformation dialog. */
    g_signal_connect( G_OBJECT( window ), "delete-event",
                      G_CALLBACK( cb_delete_event ), NULL );

    /* Connect destroy signal, which will be emitted if we return
     * FALSE from our cb_delete_event function. */
    g_signal_connect( G_OBJECT( window ), "destroy",
                      G_CALLBACK( gtk_main_quit ), NULL );

    /* Create button ... */
    button = gtk_button_new_from_stock( GTK_STOCK_ABOUT );

    /* ... and connect clicked signal to handler that will create
     * and show about dialog. */
    g_signal_connect( G_OBJECT( button ), "clicked",
```

```
                        G_CALLBACK( cb_show_about ), window );
    gtk_container_add( GTK_CONTAINER( window ), button );

    /* Show our main window and start main loop. */
    gtk_widget_show_all( window );
    gtk_main();

    return( 0 );
}
```

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

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

class Sample( gtk.Window ):
    def cb_delete_event( self, window, event ):
        # If dialog does not exists, create it
        if self.quit_dialog == None:
            # Create dialog
            self.quit_dialog = gtk.Dialog()

            # Set it modal and transient for main window.
            self.quit_dialog.set_modal( True )
            self.quit_dialog.set_transient_for( self )

            # Set title
            self.quit_dialog.set_title( 'Conformation' )

            # Add buttons.
            self.quit_dialog.add_button( gtk.STOCK_YES, 1 )
            self.quit_dialog.add_button( gtk.STOCK_NO,  2 )

            '''
            # Using non-null parameter list when creating dialog,
            # the last six calls can be written as:
            self.quit_dialog = gtk.Dialog( 'Conformation', self,
                                           gtk.DIALOG_MODAL,
                                           ( gtk.STOCK_YES, 1,
                                             gtk.STOCK_NO,  2 ) )
            '''

            # Create label
            label = gtk.Label( 'Are you sure you want to quit?' )

            # Pack label, taking API change in account
            if gtk.pygtk_version[1] < 14:
                self.quit_dialog.vbox.pack_start( label )
```

```
            else:
                self.quit_dialog.get_content_area().pack_start( label )

            # Show dialog
            self.quit_dialog.show_all()

        # Run dialog
        response = self.quit_dialog.run()
        self.quit_dialog.hide()

        return response != 1

    def cb_show_about( self, button ):
        if self.about_dialog == None:
            # Create about dialog
            self.about_dialog = gtk.AboutDialog()
            self.about_dialog.set_transient_for( self )

            # Set dialog's properties
            self.about_dialog.set_program_name( "Sample Dialog App" )
            self.about_dialog.set_version( "0.1" )
            self.about_dialog.set_copyright( "Copyright 2009 © Tadej Borovšak" )
            self.about_dialog.set_website( "http://tadeboro.blogspot.com" )
            self.about_dialog.set_authors(
                                [ "Tadej Borovšak <tadeboro@gmail.com>" ] )

            # Show dialog
            self.about_dialog.show_all()

        # Run dialog
        self.about_dialog.run()
        self.about_dialog.hide()

        # Convenience method for creating about dialog is wrapped,
        # but unfortunatelly I don't know how to use it.
        # API reference for PyGTK says nothing about gtk.show_about_dialog
        # Sorry for the inconveniance, Python coders.

    def __init__( self ):
        # Create main window
        gtk.Window.__init__( self )
        self.set_border_width( 10 )

        # Connect delete event signal - this is where we'll show our
        # quit conformation dialog.
        self.connect( 'delete-event', self.cb_delete_event )

        # Connect destroy signal, which will be emitted if we return
        # False from our self.cb_delete_event method. */
        self.connect( 'destroy', lambda *w: gtk.main_quit() )
```

```
        # Create button ...
        button = gtk.Button( None, gtk.STOCK_ABOUT )

        # ... and connect clicked signal to handler that will create
        # and show about dialog.
        button.connect( 'clicked', self.cb_show_about )
        self.add( button )

        # Add "placeholders" for dialogs
        self.about_dialog = None
        self.quit_dialog  = None

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

Posted 25th April 2009 by Tadej Borovšak

Labels: GTK+, GtkDialog, tutorial

2 View comments

Sasa Ostrouska 22 December, 2010 03:05

Zivijo Tadej, lep clanek, najlepsa hvala.

Reply

Jan 12 March, 2011 10:08

Thanks for this nice tutorial! This one and the gtkTreeView tut really saved my day. Glade and pyGtk rock...

Reply

Enter your comment...

Comment as: Google Accou ▼

Publish Preview