6th May 2009

# Using pop-up windows

I was quite bussy last week and haven't posted anything, but here I am back with new topic.

We'll be looking at pop-up windows today (windows that are not visible throughout the applications life cycle and are usually shown in response to some user's action - preference/settings dialog being most usually implemented like this). I'll describe different methods of creating pop-up windows, what being "modal" and "transient" means and how creating user interfaces with glade fits in. At the end, you'll also find sample applications that demonstrate some of the principles found in this post.

## Important information

When creating new pop-up window, you may be tempted to create window like this:

```
window = gtk_window_new( GTK_WINDOW_POPUP );
```

or, if you're using Python:

```
window = gtk.Window( gtk.WINDOW_POPUP )
```

Just don't. Creating this kind of windows is reserved for tooltips, menus and other specialized occasions. What you'll want to do is to set window type hint with gtk_window_set_type_hint function (set_type_hint method).

You've been warned.

## Creating pop-up window

As I already wrote a couple of times before, there are two quite distinct ways of handling pop-up windows (I made up those names, so don't be shocked if someone asks you: "Where the heck did you learn that!?";):

1. *create-destroy* approach, where pop-up window is created just before it's shown and destroyed after not being needed anymore
2. *show-hide* approach, where pop-up window is created only once, usually at application startup or just before the first show

There are pros and cons to both of those methods and when combined, we could say that:

- create-destroy method is more memory efficient, since window is only held in memory when needed, but might be a little slower compared to show-hide method, because window needs to be recreated every time we want to show it
- show-hide method uses more memory, since window is loaded throughout the application's life cycle, but window is generally faster to show up

Which method is better? As usual, there is no single truth. On embedded systems, where memory available is scarce, you'll probably opt for the first method, while on desktop systems, second method might be preferred since it's a bit faster. Second method also makes it easier to make sure only one copy of pop-up is shown (see cb_show_prop callback function in second example code for more info).

What about glade? You can use both methods with glade, although the second method might feel a bit more natural, since interfaces tend to be created at application startup from a single glade UI file.

## Making sure user is pleased AND plays by the rules

This is where modality and making window transient comes into play.

Making window modal means restricting user's interaction with other open windows. Windows can be made application modal (when window is opened, interacting with other windows in the same application is not possible) or system modal (interaction with other windows is not possible, regardless of application window belongs to), although GTK+ only supports making windows application modal (which is a "Good Thing™" IMHO).

Since any kind of restriction usually makes people unhappy, try to make windows modal only if absolutely necessary. One good reason for making your window modal is possibility of data loss if user interacts with other windows.

Making pop-up window transient for the main window means allowing window managers to do some magic with our window like keeping it on top of and centering it over the main window. But this is only a hint, hence the "allow" in the first sentence; window manager might do nothing or eat your pet (but in general, window managers are quite well mannered and do what you expect;).

Sample code

In order to demonstrate some of the principles described above, I wrote two sample applications (one in C and another one in Python). Feel free to dissect them and use any newly acquired knowledge in your applications. Happy coding.

Bye.

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

# Sample PyGTK application that demonstrates create-destroy method of
# creating pop-up windows.

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

class Sample( gtk.Window ):
    def __init__( self ):
        gtk.Window.__init__( self )
        self.connect( "destroy", lambda *w: gtk.main_quit() )

        button = gtk.Button( "Properties" )
        button.connect( "clicked", self.cb_show_prop )
        self.add( button )

    def cb_show_prop( self, button ):
        popup = gtk.Window()
        popup.set_title( "Properties" )

        popup.add( gtk.Label( "No properties here, but this popup still\n"
```

```
                                    "represents a threat to data integrity.;)" ) )

        popup.set_modal( True )
        popup.set_transient_for( self )
        popup.set_type_hint( gtk.gdk.WINDOW_TYPE_HINT_DIALOG )

        popup.connect( "destroy", lambda *w: gtk.main_quit() )

        popup.show_all()

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

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

G_MODULE_EXPORT void
cb_show_prop( GtkButton *button,
              GtkWidget *popup )
{
    gtk_window_present( GTK_WINDOW( popup ) );
}

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

    gtk_init( &argc, &argv );

    builder = gtk_builder_new();
    gtk_builder_add_from_file( builder, "popup.builder", NULL );

    window = GTK_WIDGET( gtk_builder_get_object( builder, "main_w" ) );
    popup  = GTK_WIDGET( gtk_builder_get_object( builder, "popup_w" ) );

    gtk_builder_connect_signals( builder, popup );
    g_object_unref( G_OBJECT( builder ) );

    gtk_widget_show_all( window );
    gtk_main();

    return( 0 );
}
```

```
<?xml version="1.0"?>
<interface>
  <!-- interface-requires gtk+ 2.12 -->
```

```
<!-- interface-naming-policy project-wide -->
<object class="GtkWindow" id="main_w">
  <signal name="destroy" handler="gtk_main_quit"/>
  <child>
    <object class="GtkButton" id="button1">
      <property name="label" translatable="yes">gtk-preferences</property>
      <property name="visible">True</property>
      <property name="can_focus">True</property>
      <property name="receives_default">True</property>
      <property name="use_stock">True</property>
      <signal name="clicked" handler="cb_show_prop"/>
    </object>
  </child>
</object>
<object class="GtkWindow" id="popup_w">
  <property name="title" translatable="yes">Properties</property>
  <signal name="delete_event" handler="gtk_widget_hide_on_delete"/>
  <child>
    <object class="GtkLabel" id="label1">
      <property name="visible">True</property>
      <property name="label" translatable="yes">This pop-up is safe.</property>
    </object>
  </child>
</object>
</interface>
```

Posted 6th May 2009 by Tadej Borovšak

Labels: GTK+, Tips

6 View comments

**invierta en franquicias** 22 February, 2010 06:18

Thanks for all information. I'm very interesting in your blog please send any update

Reply

**Deciimal** 18 December, 2010 17:37

Errrr...

(popup:31303): Gtk-WARNING **: Could not find signal handler 'cb_show_prop

Reply

**Inversiones en oro** 06 June, 2011 22:07

Hello, i think that you did a great work, i really like the blog and i think that is the best that i have read.

Reply

**Anusha Iyer** 23 June, 2011 14:54

Thank you so much for this post, Tadeboro. This was exactly what I was looking for in the wide wide web. Your blog has been very helpful for a beginner like me!

Keep up the good work :-). All the best!!

Reply

**Anonymous** 01 April, 2012 16:34

@Deciimal: You have to compile with -export-dynamic.

I.e.: $ gcc -Wall -Wextra -pedantic -std=c89 -o p pop.c `pkg-config --cflags gtk+-2.0` `pkg-config --libs gtk+-2.0` -export-dynamic

Reply

**Anonymous** 03 November, 2013 19:12

Ah, it's so easy when you know how ;-) thank-you very much.

Reply

Enter your comment...

Comment as: Google Accou ▼

Publish Preview