

WxSmith tutorial: Using wxPanel resources

From Code::Blocks

Contents

- 1 Using wxPanel resources
 - 1.1 Creating our playground
 - 1.2 Adding the wxPanel using a "Custom" item
 - 1.3 Adding custom panel through standard wxPanel

Using wxPanel resources

[Subsequent tutorials do not depend upon this one. If you find it confusing or not relevant to your needs, you can skip to the next without loss of continuity.]

In some cases, one resource must be split into a few smaller parts. Such a split gives a few advantages: it helps with multiple people working on the same project, it may give you a cleaner view over really complex resources and it may help to divide source code for logical parts in case of several functionalities in one window. In this tutorial I'll show you how to create such resources.

Creating our playground

wxPanel resources can not live as independent items - they must be placed inside a frame or dialog. We can use the main resource created inside the wizard. Assuming that you've read the previous tutorials, creating a simple window shouldn't be a big problem for you :).

Let's start with something like this:

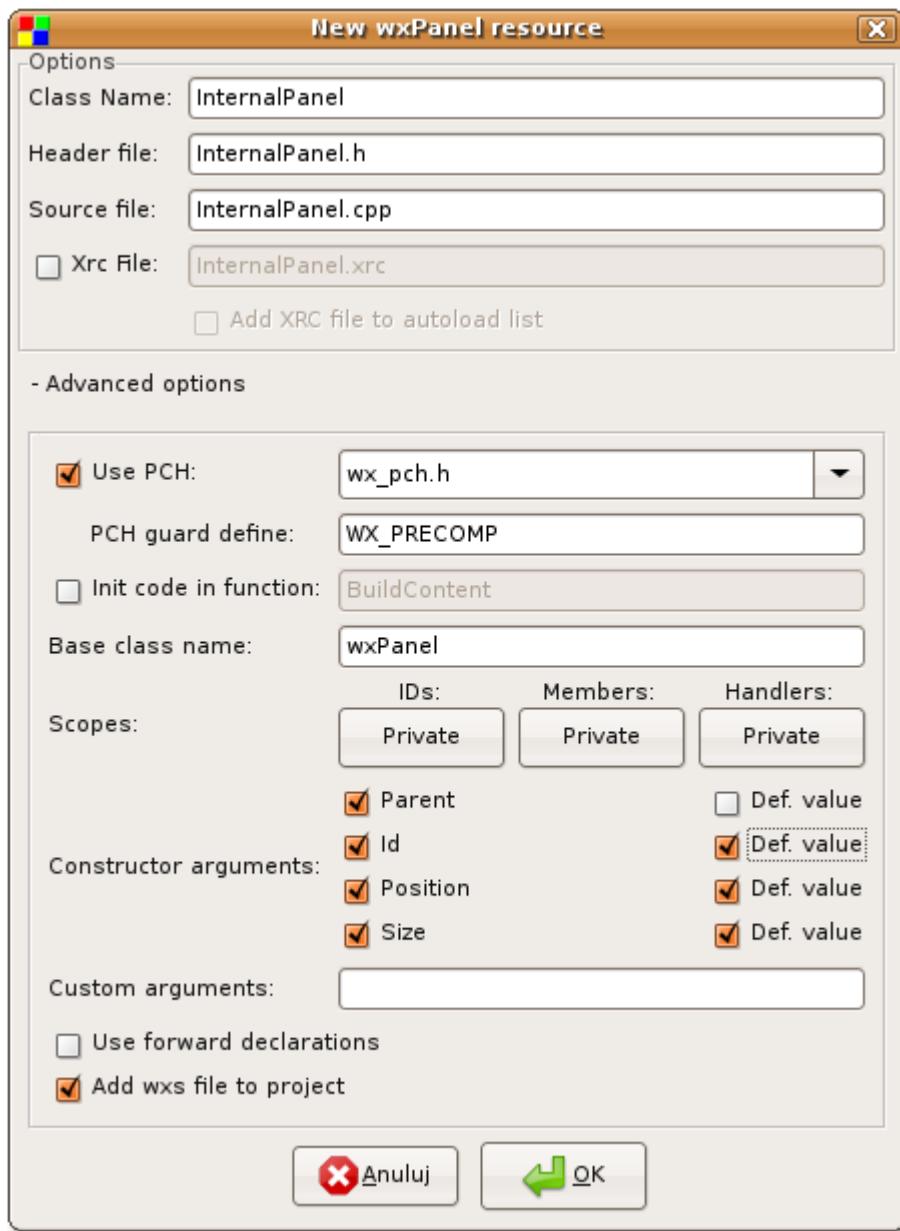


Adding the wxPanel using a "Custom" item

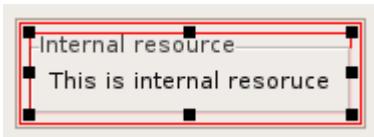
In this tutorial I'll show you two methods of embedding an external wxPanel inside another resource. The first one uses a "Custom" item which can be used to add any kind of resource not known to wxSmith.

So let's create the wxPanel resource by using the wxSmith item on the Code::Blocks main menu. Note that using this embedding method will affect the initial

configuration of the resource. We want the id, size and position of our panel to be controlled by the parent resource so make sure that we add them into the panel's constructor:



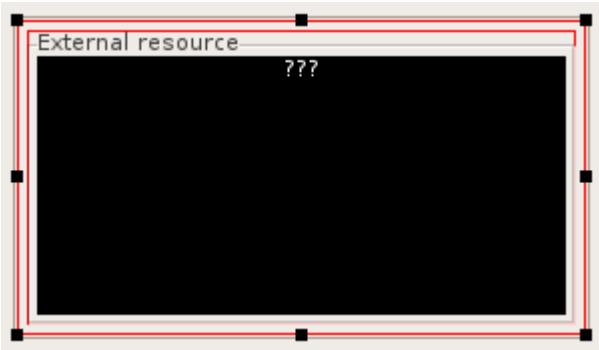
Now let's add some content into the panel:



The final step is to add a "Custom" item into the main window and configure it properly. The custom icon is the last one in the "Standard palette" identified by this icon:



A new item will be shown as a black square with three question marks on the top. Let's resize it a little bit:



Now we need to adjust the custom item's properties.

The first thing we will adjust is the "Creating code" property. As the name says, here we will be able to adjust the way wxSmith adds the code responsible for creating this item. The default value is:

```
$(THIS) = new $(CLASS)($(PARENT),$(ID),$(POS),$(SIZE),$(STYLE),wxDefaultValidator,$(NAME));
```

You may find that there are some macros used. They are here to help you map other properties of this item into the creating code:

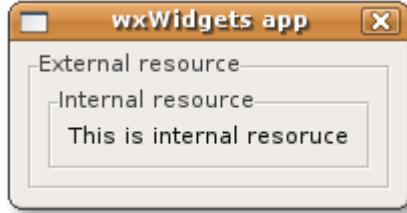
- \$(THIS) is replaced by the name of the variable for this item
- \$(CLASS) is replaced by the name of the item's class
- \$(PARENT) is replaced by the name of the parent item (it's granted that it will be a pointer to a class derived from wxWindow)
- \$(ID) is replaced by the value of the ID property
- \$(POS) is replaced by the value of the position property (it may be adjusted by using drag boxes)
- \$(SIZE) is replaced by the value of the size property (it may be adjusted by using drag boxes)
- \$(STYLE) is replaced by the style property - since custom items doesn't have a predefined set of styles, it's treated as a normal string
- \$(NAME) is replaced by the generated name (which is equivalent to the string representation of the item's identifier)

The default code template creates a standard item which uses the default set of properties. Let's replace it by a value corresponding to our panel's constructor:

```
$(THIS) = new $(CLASS)($(PARENT),$(ID),$(POS),$(SIZE));
```

Now we have to give the name of the header file with our external resource in the "Include file" property. I've created the wxPanel with the name "InternalPanel" so the header will be "InternalPanel.h". Also let's check the "Use "" for include..." since we're including a local file.

The last property to adjust is the "Class name" - we need to put the name of our panel here; in my case it's "InternalPanel". When we do this, our embedded resource is ready. While the resource view and the show preview continue to show the black square with three question marks, if you rebuild/run the project you will see the embedded resource;



Adding custom panel through standard wxPanel

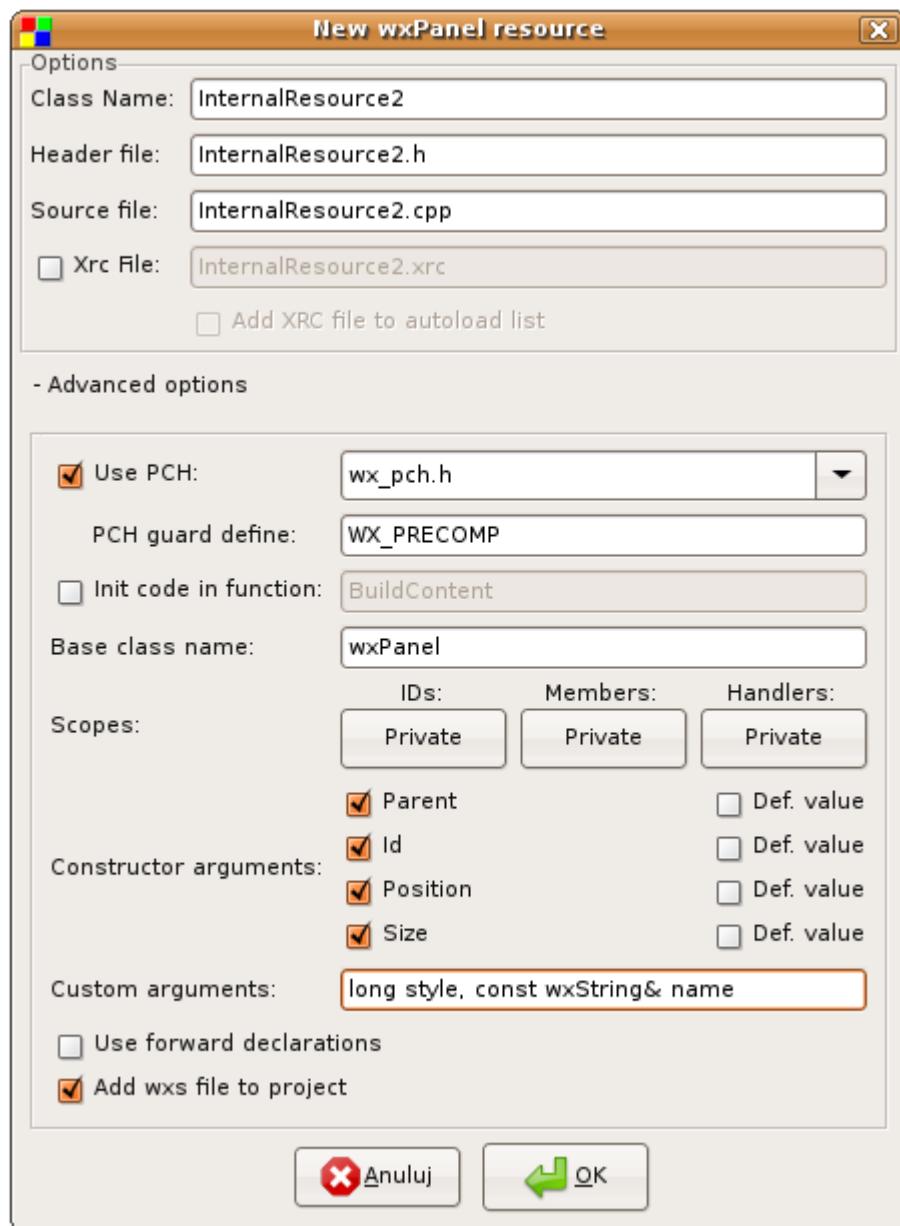
The second way in which an external resource may be used is through a normal wxPanel. If you look closer, you'll find that most of the items have a property called "Class name" - this is the name of the class which will be used as the item's type. By default it contains the name of the original class in wxWidgets. Changing it will notify wxSmith that a different class will be used instead.

So to put our own panel here we can add a "normal" panel into the main resource and change the class name to the name of our internal resource. So far it's easy but there's one requirement to this technique - our internal resource must have exactly the same constructor arguments as in the case of the "original" class.

So let's take a look at the wxWidgets documentation. Here's the declaration of wxPanel's constructor:

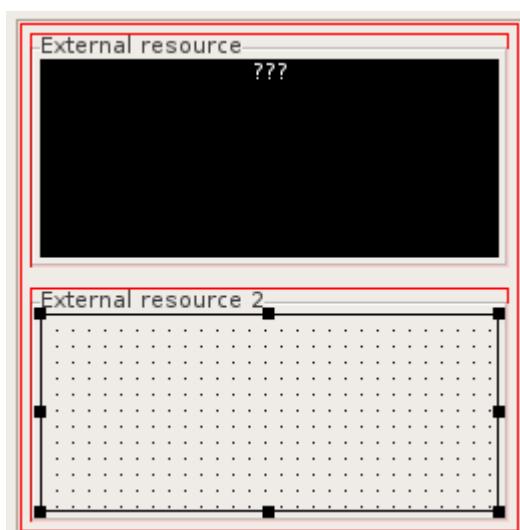
```
wxPanel( wxWindow* parent, wxWindowID id = wxID_ANY,
         const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize,
         long style = wxTAB_TRAVERSAL, const wxString& name = "panel")
```

Now let's create our own internal resource. We will have to adjust constructor arguments as in the case of the previous method:



Note that I've added custom arguments and turned off all default values (it's required since we cannot easily add default values of our custom args).

Now that our panel is ready we can add it to the main resource - let's add a "normal" panel first:

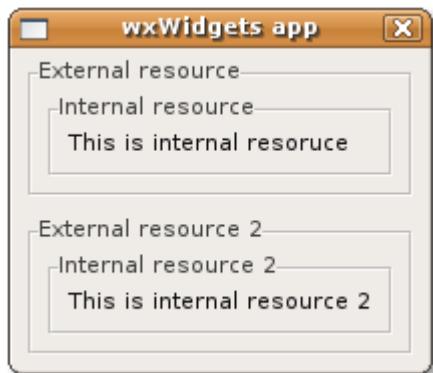


and change "Class name" property to the name of our resource (in my case it's InternalResource2).

We still need to add #include "InternalResource2.h" into the sources to make our project compile. In the previous solution it was done automatically through properties. Now we don't have such a system so let's add it manually (remember to put our include outside wxSmith's code section, otherwise any change in the resource will remove our change). Note that we need to add it to the header file of the main resource:

```
/* Headers(Tutorial5Dialog)
#include <wx/sizer.h>
#include <wx/panel.h>
#include "InternalPanel.h"
#include <wx/dialog.h>
*/
#include "InternalResource2.h"
```

Now we can run our application:



I've presented two easy methods of creating complex windows from smaller resources. Perhaps you could find other ways to do it (if you do so, you can extend this tutorial :)). On wxSmith's wishlist there's also a nice feature request to allow adding external panels in a more natural way (just by a few clicks) which will probably make other methods obsolete. But I hope that I gave you enough information to build nice compound resources so far :)

[Previous](#) | [Index](#) | [Next](#)