How To Add New Panels

BigWorld Technology 2.0. Released 2010.

Software designed and built in Australia by BigWorld.

Level 2, Wentworth Park Grandstand, Wattle St
Glebe NSW 2037, Australia
www.bigworldtech.com

Table of Contents

Chapter 1. Introduction

Three content creation tools, World Editor Model Editor and Particle Editor, use a custom tear-off panel system developed by BigWorld. This system allows for easy creating of new panels from C++ which can then be docked, teared off and resized by the user. For more information, please see the Content Tools Reference Guide's section *Panel System*.

Chapter 2. Sample Panel

We have included a barebones panel template example in the World Editor Visual Studio project that can be used either in World Editor, Model Editor or Particle Editor. It is implemented in the file `src/tools/worldeditor/gui/pages/page_my_panel.cpp` and its corresponding header file. The code is well documented, and includes comments about how to convert that template into an actual panel in the tool.

Chapter 3. Detailed Steps

It is always best to start by looking at one of the other panels, or from the self-explanatory sample panel template described above, but it is also important to know the detailed steps involved in the process.

First, you start from either a `CDialog` or a `CFormView`. A `CDialog` is simpler and won't show scrollbars if the tab is resized too small. `CDialog` classes are great if you make your dialog's controls to expand / shrink with the panel. A `CDialog` based example would be `PageOptionsHistogram`, in `src/tools/bigbang/page_options_histogram.?pp`. A `CFormView` creates scrollbars as needed, so it's great for when you want to keep your controls the same size and position inside the dialog, and show scrollbars if the tab is smaller than a certain size. An example of this would be `PageOptionsWeather`, in `src/tools/bigbang/page_options_weather.?pp`. It is probably possible to inherit from other MFC classes, but those are the two we use.

Whatever MFC base class you use, you also need to inherit from `GUITABS::Content`, which is the base class that allows the `GUITABS` Manager to handle tab/panel docking/floating of your dialog. `GUITABS::Content` is pure virtual, but there are some macros that were written afterwards that implement all its methods with reasonable default implementations. These macros are defined in `src/lib/guitabs/content.hpp`, and are well documented there. For example, `PageOptionsWeather` uses the `IMPLEMENT_BASIC_CONTENT` macro at the beginning of the class' declaration.

You also need to declare a factory for your dialog. Again, some handy macros found in `src/lib/guitabs/content_factory.hpp` will be sufficient in most cases. Note that there's a special macro for `CDialog` derived panels, called `IMPLEMENT_CDIALOG_CONTENT_FACTORY`.

Once you have done all this, you are ready to go. All it's left is to register the factory, and to create a the panel manually in case a previous `layout.xml` file doesn't exist. Both these things are done in the `PanelManager` class, which for World Editor is in `src/tools/bigbang/panel_manager.?pp`. In the method `PanelManager::initPanels`, the factories are registered. For example, the `PageOptionsWeather` factory is registered like this:

```
GUITABS::Manager::instance()->registerFactory(
       new PageOptionsWeatherFactory() );
```

You must also manually insert the panel inside `PanelManager::loadDefaultPanels` like this:

```
p = GUITABS::Manager::instance()->insertPanel(
       PageOptionsWeather::contentID, GUITABS::TAB, p );
```

Last, there's a map that we build manually to keep track of the panels currently supported in the tool. This is handy for when a new panel is added, the tool will detect that it's missing in the `layout.xml` file, and will reload the default panel layout which should bring up the new panel (because you add it there with `insertPanel()` as described above). This is done in the `PanelManager` constructor, so simply add your panel there. Particle Editor has its own way of checking for missing dialogs as well, in `PanelManager::finishLoad()`.

Once you do all this, the tool will take care of all the docking / floating of your panel, plus it will save your last panel layout to disk or load the default layout when it needs to (i.e. the user requests it by clicking on the appropriate menu item).