Using Custom Widgets with Qt Designer

Qt Designer can display custom widgets through its extensible plugin mechanism, allowing the range of designable widgets to be extended by the user and third parties. Alternatively, it is possible to use existing widgets as placeholders for widget classes that provide similar APIs.

Handling Custom Widgets

Although *Qt Designer* supports all of the standard Qt widgets, some specialized widgets may not be available as standard for a number of reasons:

- Custom widgets may not be available at the time the user interface is being designed.
- Custom widgets may be platform-specific, and designers may be developing the user interface on a different platform to end users.
- The source code for a custom widget is not available, or the user interface designers are unable to use the widget for non-technical reasons.

In the above situations, it is still possible to design forms with the aim of using custom widgets in the application. To achieve this, we can use the widget promotion feature of *Qt Designer*.

In all other cases, where the source code to the custom widgets is available, we can adapt the custom widget for use with *Qt Designer*.

Promoting Widgets

If some forms must be designed, but certain custom widgets are unavailble to the designer, we can substitute similar widgets to represent the missing widgets. For example, we might represent instances of a custom push button class, `MyPushButton`, with instances of QPushButton and promote these to `MyPushButton` so that uic generates suitable code for this missing class.

When choosing a widget to use as a placeholder, it is useful to compare the API of the missing widget with those of standard Qt widgets. For specialized widgets that subclass standard classes, the obvious choice of placeholder is the base class of the custom widget; for example, QSlider might be used for specialized QSlider subclasses.

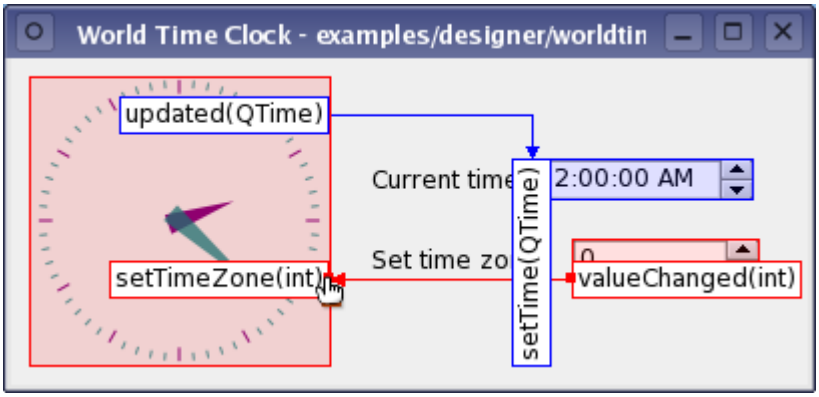
For specialized widgets that do not share a common API with standard Qt widgets, it is worth considering adapting a custom widget for use in *Qt Designer*. If this is not possible then QWidget is the obvious choice for a placeholder widget since it is the lowest common denominator for all widgets.

To add a placeholder, select an object of a suitable base class and choose **Promote to ...** from the form's context menu. After entering the class name and header file in the lower part of the dialog, choose **Add**. The placeholder class will now appear along with the base class in the upper list. Click the **Promote** button to accept this choice.

Now, when the form's context menu is opened over objects of the base class, the placeholder class will appear in the **Promote to** submenu, allowing for convenient promotion of objects to that class.

A promoted widget can be reverted to its base class by choosing **Demote to** from the form's context menu.

User Defined Custom Widgets



Custom widgets can be adapted for use with *Qt Designer*, giving designers the opportunity to configure the user interface using the actual widgets that will be used in an application rather than placeholder widgets. The process of creating a custom widget plugin is described in the Creating Custom Widgets for Qt Designer chapter of this manual.

To use a plugin created in this way, it is necessary to ensure that the plugin is located on a path that *Qt Designer* searches for plugins. Generally, plugins stored in `$QTDIR/plugins/designer` will be loaded when *Qt Designer* starts. Further information on building and installing plugins can be found here. You can also refer to the Plugins HOWTO document for information about creating plugins.

‹ Customizing Qt Designer Forms

Creating Custom Widgets for Qt Designer ›