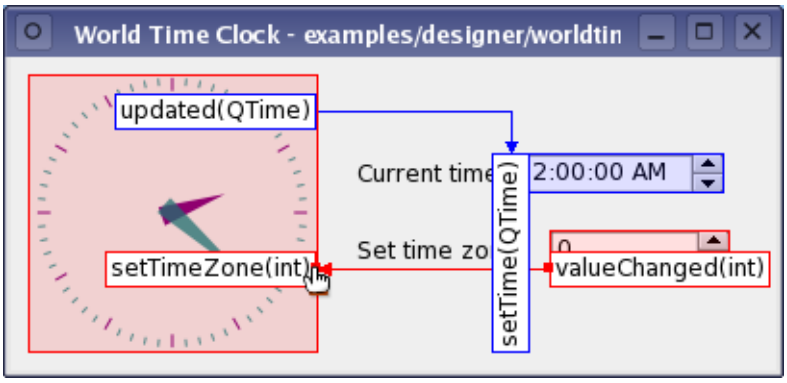
搜索

为 Qt 设计器创建自定义小部件

*Qt Designer*基于插件的架构允许编辑用户定义的和第三方自定义的小部件，就像使用标准Qt小部件一样。*Qt Designer*可以使用自定义控件的所有功能，包括控件属性，信号和插槽。由于*Qt Designer*在表单设计过程中使用真实的小部件，因此自定义小部件的外观将与预览时相同。



QtDesigner模块使您能够在*Qt设计器*中创建自定义小部件。

开始

要将自定义小部件与*Qt Designer*集成，您需要为小部件提供合适的描述和适当的文件。`.pro`

提供接口说明

要告知*Qt Designer*您想要提供的小部件类型，请创建QDesignerCustomWidgetInterface的子类，用于描述您的小部件公开的各种属性。其中大多数是由基类中的纯虚拟函数提供的，因为只有插件的作者才能提供此信息。

功能	返回值的说明
`name()`	提供小组件的类的名称。
`group()`	*Qt Designer*的小部件框中小部件所属的组。
`toolTip()`	帮助用户在*Qt Designer*中识别小部件的简短描述。
`whatsThis()`	为*Qt Designer*用户提供的对小部件的较长描述。
`includeFile()`	必须包含在使用此小组件的应用程序中的头文件。此信息存储在 UI 文件中，并将用于在它为包含自定义小部件的表单生成的代码中创建合适的语句。`uic#includes`
`icon()`	可用于在*Qt Designer*的小部件框中表示小部件的图标。
`isContainer()`	如果小部件将用于保存子小部件，则为 true；否则为假。

	注意： createWidget（） 是一个工厂函数，只负责创建 widget。在 load（） 返回之前，自定义小部件的属性将不可用。
domXml()	小部件属性的说明，例如其对象名称、大小提示和其他标准QWidget属性。
codeTemplate()	此功能保留供*Qt Designer*将来使用。

还可以重新实现另外两个虚拟函数：

initialize()	为自定义小部件设置扩展和其他功能。自定义容器扩展（请参阅QDesignerContainerExtension）和任务菜单扩展（请参阅QDesignerTaskMenuExtension）应在此函数中设置。
isInitialized()	如果小部件已初始化，则返回 true;否则返回 false。重新实现通常会检查函数是否已被调用并返回此测试的结果。initialize()

关于函数的说明domXml()

该函数返回一个UI文件片段，*Qt Designer*的小部件工厂使用该片段创建自定义小部件及其适用的属性。domXml()

从Qt 4.4开始，*Qt Designer*的小部件框允许一个完整的UI文件来描述**一个**自定义小部件。可以使用标记加载 UI 文件。指定 <ui> 标记允许添加包含自定义小部件附加信息的 <customwidget> 元素。如果不需要其他信息，则标签就足够了<ui><widget>

如果自定义小部件没有提供合理的大小提示，则需要在子类中的函数返回的字符串中指定默认几何。例如，由自定义控件插件提供的示例，按以下方式定义默认控件几何：domXml()AnalogClockPlugin

```
    ...
R"(
    <property name="geometry">
      <rect>
        <x>0</x>
        <y>0</y>
        <width>100</width>
        <height>100</height>
      </rect>
    </property>
")
    ...
```

该函数的另一个功能是，如果它返回一个空字符串，则该小部件将不会安装在*Qt Designer*的小部件框中。但是，它仍然可以由表单中的其他小部件使用。此功能用于隐藏不应由用户显式创建但其他小部件所需的小部件。domXml()

完整的自定义小部件规范如下所示：

```
<ui language="c++"> displayname="MyWidget">
    <widget class="widgets::MyWidget" name="mywidget"/>
    <customwidgets>
        <customwidget>
            <class>widgets::MyWidget </class>
            <addpagemethod>addPage </addpagemethod>
```

```
                <tooltip name="text">Explanatory text to be shown in Property Editor</tooltip>
            </propertyspecifications>
        </customwidget>
    </customwidgets>
</ui>
```

标签的属性：<ui>

属性	存在	值	评论
language	自选	“C++”、“占碑”	此属性指定自定义小组件的用途语言。它主要是为了防止C++插件出现在Qt Jambi中。
displayname	自选	类名	属性的值显示在“小组件”框中，可用于剥离命名空间。

Thetag告诉*Qt Designer*和uic应该使用哪种方法来将页面添加到容器小部件。这适用于需要调用特定方法来添加子项而不

该元素可以包含属性元信息的列表。<propertyspecifications>

标记可用于指定将鼠标悬停在属性上时要在属性编辑器中显示的工具提示。属性名称在属性中给出，元素文本是工具提示。此功能是在Qt 5.6中添加的。<tooltip>name

对于字符串类型的属性，可以使用标记。此标记具有以下属性：<stringpropertyspecification>

属性	存在	值	评论
name	必填	属性的名称	
type	必填	见下表	属性的值确定属性编辑器将如何处理它们。
notr	自选	“真”、“假”	如果属性为“true”，则不打算转换该值。

字符串属性的属性值：type

价值	类型
"richtext"	富文本。
"multiline"	多行纯文本。
"singleline"	单行纯文本。
"stylesheet"	CSS 样式的工作表。
"objectname"	对象名称（一组受限制的有效字符）。
"url"	网址、文件名。

插件要求

为了使插件在所有平台上都能正常工作，您需要确保它们导出*Qt Designer*所需的符号。

首先，必须导出插件类才能由*Qt Designer*加载插件。使用Q_PLUGIN_METADATA（） 宏执行此操作。此外，

创建行为良好的小部件

一些自定义小部件具有特殊的用户界面功能，可能会使它们的行为与*Qt Designer*中的许多标准小部件不同。具体来说，如果自定义控件由于调用QWidget：：grabKeyboard（）而抓取键盘，则*Qt Designer*的操作将受到影响。

要在*Qt* Designer中为自定义控件提供特殊行为，请提供initialize（）函数的实现，以配置*Qt Designer*特定行为的控件构造过程。这个函数将在调用createWidget（）之前第一次调用，并且也许可以设置一个内部标志，稍后当*Qt Designer*调用插件的createWidget（）函数时可以测试该标志。

构建和安装插件

一个简单的插件

自定义控件插件示例演示了一个简单的*Qt设计器*插件。

插件的文件必须指定自定义小部件和插件界面的标头和源。通常，此文件只需指定插件的项目将构建为库，但具有针对*Qt Designer*的特定插件支持。这是通过以下声明完成的：`.pro`

```
QT          += widgets uiplugin
CONFIG      += plugin
TEMPLATE    = lib
```

变量包含关键字。它表示该插件仅使用抽象接口QDesignerCustomWidgetInterface和QDesignerCustomWidgetCollectionInterface，并且没有链接到*Qt Designer*库。当访问*Qt Designer*的其他有链接的界面时，应改用;这确保了插件动态链接到*Qt Designer*库，并具有对它们的运行时依赖性。`QTuiplugindesigner`

如果插件是在与*Qt Designer*不兼容的模式下构建的，则不会加载和安装它们。有关插件的更多信息，请参阅插件HOWTO文档。

还需要确保插件与其他*Qt Designer*小部件插件一起安装：

```
target.path = $$[QT_INSTALL_PLUGINS]/designer
INSTALLS += target
```

变量是已安装的Qt插件位置的占位符。您可以在运行应用程序之前设置环境变量，将*Qt Designer*配置为在其他位置查找插件。`$[QT_INSTALL_PLUGINS]QT_PLUGIN_PATH`

> **注意：** *Qt Designer*将在提供的每个路径中查找子目录。`designer`

请参阅QCoreApplication：：libraryPaths（） 了解有关使用 Qt 应用程序自定义库和插件路径的更多信息。

拆分插件

在实际场景中，您不希望应用程序依赖于使用自定义小部件到*Qt Designer*头文件和库，如上面解释的简单方法所介绍的那样。

以下各节介绍如何解决此问题。

将小部件链接到应用程序中

```
INCLUDEPATH += $$PWD
HEADERS += $$PWD/analogclock.h
SOURCES += $$PWD/analogclock.cpp
```

然后，此文件将包含在插件和应用程序的文件中：`.pro`

```
include(customwidget.pri)
```

使用库共享微件

另一种方法是将小部件放入链接到*Qt Designer*插件和应用程序的库中。建议使用静态库，以避免在运行时查找库时出现问题。

有关共享库，请参阅创建共享库。

将插件与 QUiLoader 一起使用

将自定义小部件添加到QUiLoader的首选方法是对其进行子类化，重新实现QUiLoader：：createWidget（）。

但是，也可以使用*Qt Designer*自定义小部件插件（参见QUiLoader：:p luginPaths（） 和相关函数）。为了避免将Qt Designer库部署到目标设备上，这些插件不应链接到Qt Designer库（请参阅为*Qt*Designer创建自定义控件#构建和安装插件）。`QT = uiplugin`

相关示例

有关在Qt Designer中使用自定义控件的更多信息，请参阅自定义控件插件和世界时钟插件示例，以获取有关*在Qt Designer*中使用自定义控件的更多信息。此外，您可以使用QDesignerCustomWidgetCollectionInterface类将多个自定义小部件组合到一个库中。

‹ 在Qt Designer中使用自定义小部件

创建自定义小部件扩展 ›

编辑部
职业
办公地点

常见问题

支持

支持服务
专业服务
合作 伙伴
训练

对于客户

支持中心
下载
Qt登录
联系我们
客户成功案例

社区

为Qt做贡献
论坛
维基
下载
市场

反馈 登录