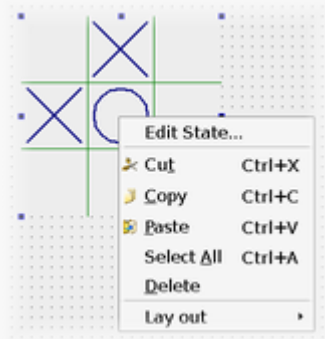
创建自定义小组件扩展

一旦你有了*Qt Designer*的自定义小部件插件，你就可以在*Qt Designer*的工作空间中使用自定义小部件扩展为它提供预期的行为和功能。

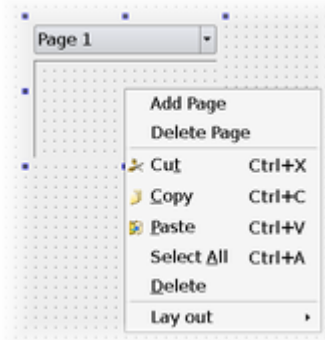
扩展类型

*Qt 设计器*中有几种类型的扩展。您可以在相同的模式中使用所有这些扩展，只需替换相应的扩展基类。

在实现自定义多页容器时，QDesigner 容器扩展是必需的。

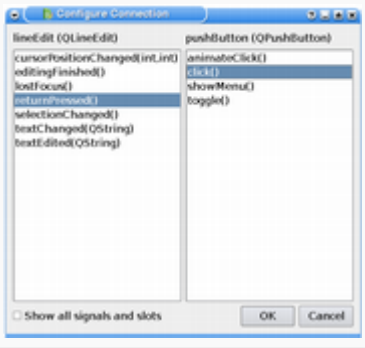


QDesignerTaskMenuExtension
QD设计者任务菜单扩展对于自定义小部件很有用。它提供了一个扩展，允许您将自定义菜单项添加到*Qt Designer*的任务菜单中。
任务菜单扩展示例阐释了如何使用此类。

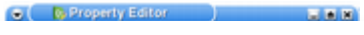


QDesignerContainerExtension
在实现自定义多页容器时，QDesigner 容器扩展是必需的。它提供了一个扩展，允许您在*Qt设计器*中添加和删除多页容器插件的页面。
容器扩展示例进一步说明了如何使用此类。

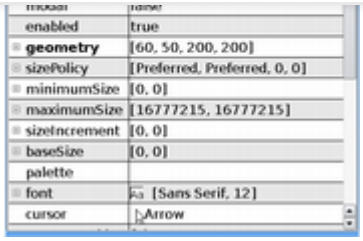
> **注意：** 由于某些小部件（例如 QTabWidget）的实现方式，无法为其添加自定义的每页属性。



QDesignerMemberSheetExtension
QD设计会员表扩展类允许您操作连接信号和插槽时显示的小部件的成员函数。



QDesignerPropertySheetExtension



式。

*Qt 设计器*使用 Q设计者属性表扩展和 Q设计者成员表扩展类来馈送其属性、信号和槽编辑器。每当在其工作区中选择小部件时，*Qt设计器*将查询小部件的属性表扩展;同样，每当请求两个小部件之间的连接时，*Qt Designer* 将查询小部件的成员工作表扩展。

> **警告：** 所有构件都具有默认属性和成员工作表。如果实现自定义属性表或成员工作表扩展，则自定义扩展将覆盖默认工作表。

创建扩展

若要创建扩展，必须同时继承 QObject 和相应的基类，并重新实现其函数。由于我们正在实现一个接口，我们必须确保使用扩展类定义中的Q_INTERFACES（）宏使元对象系统知道它。例如：

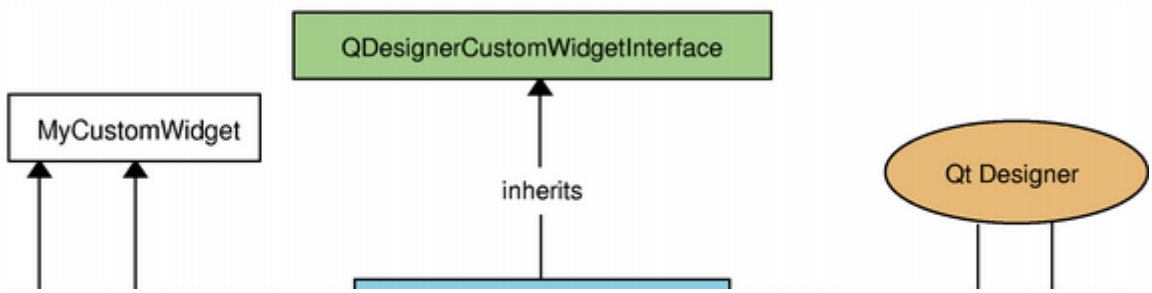
```
class MyExtension: public QObject,
                   public QdesignerContainerExtension
{
    Q_OBJECT
    Q_INTERFACE(QDesignerContainerExtension)

    ...
}
```

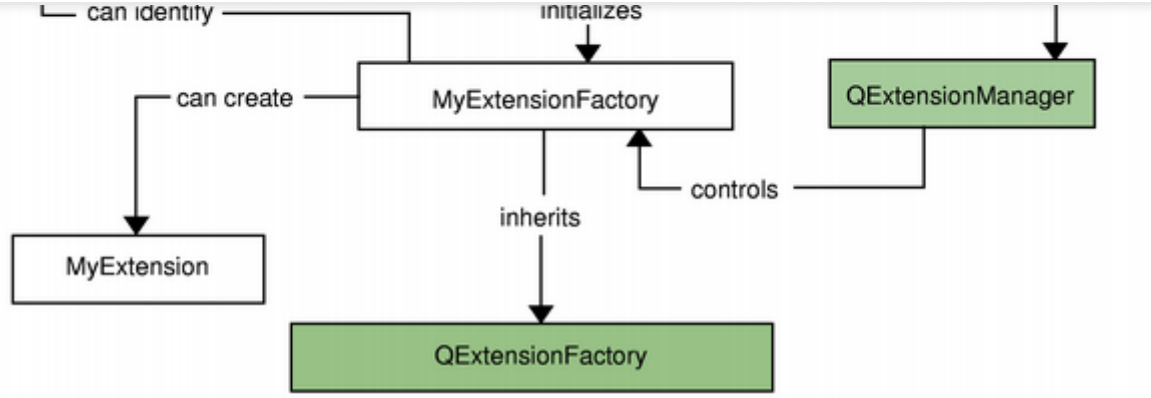
这使 *Qt 设计器*能够使用 qobject_cast（） 函数仅使用 QObject 指针查询支持的接口。

向 Qt 设计器公开扩展

在 *Qt 设计器*中，在需要扩展之前不会创建扩展。因此，在实现扩展时，必须对 QExtensionFactory 进行子类，以创建能够创建扩展实例的类。此外，您必须向*Qt设计师*的扩展管理器注册您的工厂;扩展管理器处理扩展的构造。

当请求扩展时，*Qt 设计器*的扩展管理器将运行其注册工厂，为每个工厂调用 QExtensionFactory：：createExtension（），直到找到一个能够为所选小部件创建请求的扩展。然后，此工厂将创建扩展的实例。





Creating an Extension Factory

The QExtensionFactory class provides a standard extension factory, but it can also be used as an interface for custom extension factories.

The purpose is to reimplement the QExtensionFactory::createExtension() function, making it able to create your extension, such as a MultiPageWidget container extension.

You can either create a new QExtensionFactory and reimplement the QExtensionFactory::createExtension() function:

```
QObject *ANewExtensionFactory::createExtension(QObject *object,
        const QString &iid, QObject *parent) const
{
    if (iid != Q_TYPEID(QDesignerContainerExtension))
        return 0;

    if (MyCustomWidget *widget = qobject_cast<MyCustomWidget*>
            (object))
        return new MyContainerExtension(widget, parent);

    return 0;
}
```

or you can use an existing factory, expanding the QExtensionFactory::createExtension() function to enable the factory to create your custom extension as well:

```
QObject *AGeneralExtensionFactory::createExtension(QObject *object,
        const QString &iid, QObject *parent) const
{
    MyCustomWidget *widget = qobject_cast<MyCustomWidget*>(object);

    if (widget && (iid == Q_TYPEID(QDesignerTaskMenuExtension))) {
         return new MyTaskMenuExtension(widget, parent);

    } else if (widget && (iid == Q_TYPEID(QDesignerContainerExtension))) {
        return new MyContainerExtension(widget, parent);

    } else {
```

Accessing Qt Designer's Extension Manager

When implementing a custom widget plugin, you must subclass the QDesignerCustomWidgetInterface to expose your plugin to *Qt Designer*. This is covered in more detail in the Creating Custom Widgets for Qt Designer section. The registration of an extension factory is typically made in the QDesignerCustomWidgetInterface::initialize() function:

```
void MyPlugin::initialize(QDesignerFormEditorInterface *formEditor)
{
    if (initialized)
        return;

    QExtensionManager *manager = formEditor->extensionManager();
    Q_ASSERT(manager != 0);

    manager->registerExtensions(new MyExtensionFactory(manager),
                                Q_TYPEID(QDesignerTaskMenuExtension));

    initialized = true;
}
```

The parameter in the QDesignerCustomWidgetInterface::initialize() function is a pointer to *Qt Designer*'s current QDesignerFormEditorInterface object. You must use the QDesignerFormEditorInterface::extensionManager() function to retrieve an interface to *Qt Designer*'s extension manager. Then you use the QExtensionManager::registerExtensions() function to register your custom extension factory.`formEditor`

Related Examples

For more information on creating custom widget extensions in *Qt Designer*, refer to the Task Menu Extension and Container Extension examples.

‹ Creating Custom Widgets for Qt Designer

Qt Designer's UI File Format ›

Qt DOCUMENTATION

The Qt Company

Contact Us

Company

About Us
Investors
Newsroom
Careers
Office Locations

Licensing

Terms & Conditions
Open Source
FAQ

Support

Support Services
Professional Services
Partners
Training

For Customers

Support Center
Downloads
Qt Login
Contact Us
Customer Success

Community

Contribute to Qt
Forum
Wiki
Downloads
Marketplace

Feedback Sign In