# Qobject

The [QObject](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html) class is the base class of all Qt objects.

[QObject](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html) is the heart of the Qt [Object Model](qthelp://org.qt-project.qtcore.540/qtcore/object.html). The central feature in this model is a very powerful mechanism for seamless object communication called [signals and slots](qthelp://org.qt-project.qtcore.540/qtcore/signalsandslots.html#signals-and-slots). You can connect a signal to a slot with [connect](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html#connect)() and destroy the connection with[disconnect](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html#disconnect)(). To avoid never ending notification loops you can temporarily block signals with [blockSignals](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html#blockSignals)(). The protected functions [connectNotify](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html#connectNotify)() and [disconnectNotify](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html#disconnectNotify)() make it possible to track connections.

QObjects organize themselves in [object trees](qthelp://org.qt-project.qtcore.540/qtcore/objecttrees.html). When you create a [QObject](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html) with another object as parent, the object will automatically add itself to the parent's [children](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html#children)() list. The parent takes ownership of the object; i.e., it will automatically delete its children in its destructor. You can look for an object by name and optionally type using [findChild](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html#findChild)() or [findChildren](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html#findChildren)().

{

QLabel\* label = new QLabel(QStringLiteral("label"), this);

QObject\* parent = label->parent();

if (parent == (QObject\*)this)

\_\_debugbreak();

so label is added to the children() list of this.

QGraphicsScene\* scene = new QGraphicsScene;

QObject\* parent = scene->parent();

if (parent == this->parent())

\_\_debugbreak();

so scene is added to the children() list of the parent of this.

}

Every object has an [objectName](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html#objectName-prop)() and its class name can be found via the corresponding [metaObject](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html#metaObject)() (see [QMetaObject::className](qthelp://org.qt-project.qtcore.540/qtcore/qmetaobject.html#className)()). You can determine whether the object's class inherits another class in the [QObject](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html) inheritance hierarchy by using the[inherits](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html#inherits)() function.

When an object is deleted, it emits a [destroyed](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html#destroyed)() signal. You can catch this signal to avoid dangling references to QObjects.

QObjects can receive events through [event](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html#event)() and filter the events of other objects. See [installEventFilter](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html#installEventFilter)() and [eventFilter](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html#eventFilter)() for details. A convenience handler, [childEvent](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html#childEvent)(), can be reimplemented to catch child events.

Last but not least, [QObject](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html) provides the basic timer support in Qt; see [QTimer](qthelp://org.qt-project.qtcore.540/qtcore/qtimer.html) for high-level support for timers.

Notice that the [Q\_OBJECT](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html#Q_OBJECT) macro is mandatory for any object that implements signals, slots or properties. You also need to run the [Meta Object Compiler](qthelp://org.qt-project.qtcore.540/qtdoc/moc.html#moc) on the source file. We strongly recommend the use of this macro in all subclasses of [QObject](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html)regardless of whether or not they actually use signals, slots and properties, since failure to do so may lead certain functions to exhibit strange behavior.

All Qt widgets inherit [QObject](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html). The convenience function [isWidgetType](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html#isWidgetType)() returns whether an object is actually a widget. It is much faster than [qobject\_cast](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html#qobject_cast)<[QWidget](qthelp://org.qt-project.qtcore.540/qtwidgets/qwidget.html) \*>(*obj*) or *obj*->[inherits](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html#inherits)("[QWidget](qthelp://org.qt-project.qtcore.540/qtwidgets/qwidget.html)").

Some [QObject](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html) functions, e.g. [children](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html#children)(), return a [QObjectList](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html#QObjectList-typedef). [QObjectList](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html#QObjectList-typedef) is a typedef for [QList](qthelp://org.qt-project.qtcore.540/qtcore/qlist.html)<[QObject](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html) \*>.

## Q\_OBJECT

The Q\_OBJECT macro must appear in the private section of a class definition that declares its own signals and slots or that uses other services provided by Qt's meta-object system.

**Note:**This macro requires the class to be a subclass of [QObject](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html). Use Q\_GADGET instead of Q\_OBJECT to enable the meta object system's support for enums in a class that is not a [QObject](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html) subclass. Q\_GADGET makes a class member,staticMetaObject, available. staticMetaObject is of type [QMetaObject](qthelp://org.qt-project.qtcore.540/qtcore/qmetaobject.html) and provides access to the enums declared with [Q\_ENUMS](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html#Q_ENUMS). Q\_GADGET is provided only for C++.

## Q\_PROPERTY(...)

This macro is used for declaring properties in classes that inherit [QObject](qthelp://org.qt-project.qtcore.540/qtcore/qobject.html). Properties behave like class data members, but they have additional features accessible through the [Meta-Object System](qthelp://org.qt-project.qtcore.540/qtcore/metaobjects.html#meta-object-system).

Q\_PROPERTY(type name

(READ getFunction [WRITE setFunction] |

MEMBER memberName [(READ getFunction | WRITE setFunction)])

[RESET resetFunction]

[NOTIFY notifySignal]

[REVISION int]

[DESIGNABLE bool]

[SCRIPTABLE bool]

[STORED bool]

[USER bool]

[CONSTANT]

[FINAL])

The property name and type and the READ function are required. The type can be any type supported by [QVariant](qthelp://org.qt-project.qtcore.540/qtcore/qvariant.html), or it can be a user-defined type. The other items are optional, but a WRITE function is common. The attributes default to true except USER, which defaults to false.

For example:

Q\_PROPERTY([QString](qthelp://org.qt-project.qtcore.540/qtcore/qstring.html) title READ title WRITE setTitle USER true)

# Qsettings

The [QSettings](qthelp://org.qt-project.qtcore.540/qtcore/qsettings.html) class provides persistent platform-independent application settings.

Users normally expect an application to remember its settings (window sizes and positions, options, etc.) across sessions. This information is often stored in the system registry on Windows, and in XML preferences files on Mac OS X. On Unix systems, in the absence of a standard, many applications (including the KDE applications) use INI text files.

[QSettings](qthelp://org.qt-project.qtcore.540/qtcore/qsettings.html) is an abstraction around these technologies, enabling you to save and restore application settings in a portable manner. It also supports [custom storage formats](qthelp://org.qt-project.qtcore.540/qtcore/qsettings.html#registerFormat).

[QSettings](qthelp://org.qt-project.qtcore.540/qtcore/qsettings.html)'s API is based on [QVariant](qthelp://org.qt-project.qtcore.540/qtcore/qvariant.html), allowing you to save most value-based types, such as [QString](qthelp://org.qt-project.qtcore.540/qtcore/qstring.html), [QRect](qthelp://org.qt-project.qtcore.540/qtcore/qrect.html), and [QImage](qthelp://org.qt-project.qtcore.540/qtgui/qimage.html), with the minimum of effort.

If all you need is a non-persistent memory-based structure, consider using [QMap](qthelp://org.qt-project.qtcore.540/qtcore/qmap.html)<[QString](qthelp://org.qt-project.qtcore.540/qtcore/qstring.html), [QVariant](qthelp://org.qt-project.qtcore.540/qtcore/qvariant.html)> instead.

# [QSignalMapper](qthelp://org.qt-project.qtcore.540/qtcore/qsignalmapper.html)

The [QSignalMapper](qthelp://org.qt-project.qtcore.540/qtcore/qsignalmapper.html) class bundles signals from identifiable senders.

This class collects a set of parameterless signals, and re-emits them with integer, string or widget parameters corresponding to the object that sent the signal.

The class supports the mapping of particular strings or integers with particular objects using [setMapping](qthelp://org.qt-project.qtcore.540/qtcore/qsignalmapper.html#setMapping)(). The objects' signals can then be connected to the [map](qthelp://org.qt-project.qtcore.540/qtcore/qsignalmapper.html#map)() slot which will emit the [mapped](qthelp://org.qt-project.qtcore.540/qtcore/qsignalmapper.html#mapped)() signal with the string or integer associated with the original signalling object. Mappings can be removed later using [removeMappings](qthelp://org.qt-project.qtcore.540/qtcore/qsignalmapper.html#removeMappings)().

Example: Suppose we want to create a custom widget that contains a group of buttons (like a tool palette). One approach is to connect each button's clicked() signal to its own custom slot; but in this example we want to connect all the buttons to a single slot and parameterize the slot by the button that was clicked.

Here's the definition of a simple custom widget that has a single signal, clicked(), which is emitted with the text of the button that was clicked:

class ButtonWidget : public [QWidget](qthelp://org.qt-project.qtcore.540/qtwidgets/qwidget.html)

{

Q\_OBJECT

public:

ButtonWidget([QStringList](qthelp://org.qt-project.qtcore.540/qtcore/qstringlist.html) texts, [QWidget](qthelp://org.qt-project.qtcore.540/qtwidgets/qwidget.html) \*parent = 0);

signals:

void clicked(const [QString](qthelp://org.qt-project.qtcore.540/qtcore/qstring.html) &text);

private:

[QSignalMapper](qthelp://org.qt-project.qtcore.540/qtcore/qsignalmapper.html#QSignalMapper) \*signalMapper;

};

The only function that we need to implement is the constructor:

ButtonWidget::ButtonWidget([QStringList](qthelp://org.qt-project.qtcore.540/qtcore/qstringlist.html) texts, [QWidget](qthelp://org.qt-project.qtcore.540/qtwidgets/qwidget.html) \*parent)

: [QWidget](qthelp://org.qt-project.qtcore.540/qtwidgets/qwidget.html)(parent)

{

signalMapper = new [QSignalMapper](qthelp://org.qt-project.qtcore.540/qtcore/qsignalmapper.html#QSignalMapper)(this);

[QGridLayout](qthelp://org.qt-project.qtcore.540/qtwidgets/qgridlayout.html) \*gridLayout = new [QGridLayout](qthelp://org.qt-project.qtcore.540/qtwidgets/qgridlayout.html);

for (int i = 0; i < texts.size(); ++i) {

[QPushButton](qthelp://org.qt-project.qtcore.540/qtwidgets/qpushbutton.html) \*button = new [QPushButton](qthelp://org.qt-project.qtcore.540/qtwidgets/qpushbutton.html)(texts[i]);

connect(button, SIGNAL(clicked()), signalMapper, SLOT(map()));

signalMapper->setMapping(button, texts[i]);

gridLayout->addWidget(button, i / 3, i % 3);

}

connect(signalMapper, SIGNAL(mapped([QString](qthelp://org.qt-project.qtcore.540/qtcore/qstring.html))),

this, SIGNAL(clicked([QString](qthelp://org.qt-project.qtcore.540/qtcore/qstring.html))));

setLayout(gridLayout);

}

A list of texts is passed to the constructor. A signal mapper is constructed and for each text in the list a [QPushButton](qthelp://org.qt-project.qtcore.540/qtwidgets/qpushbutton.html) is created. We connect each button's clicked() signal to the signal mapper's [map](qthelp://org.qt-project.qtcore.540/qtcore/qsignalmapper.html#map)() slot, and create a mapping in the signal mapper from each button to the button's text. Finally we connect the signal mapper's [mapped](qthelp://org.qt-project.qtcore.540/qtcore/qsignalmapper.html#mapped)() signal to the custom widget's clicked() signal. When the user clicks a button, the custom widget will emit a single clicked() signal whose argument is the text of the button the user clicked