

Basics of Qt

Andreas Jakl

Senior Technical Consultant
Forum Nokia

20 September, 2010
v3.0.0

Contents

- Signals and Slots
- Widgets, Layouts and Styles
- Meta-Objects and Memory Management

A man and a woman are sitting at a table in a cafe, playing a slot machine game on a smartphone. The man is on the left, looking at the phone with a focused expression. The woman is on the right, smiling and shielding her eyes with her hands, as if she's excited or surprised by the game. The table has a white mug, a glass of water, and some snacks. The background wall is covered with many small framed portraits of famous people.

Signals and Slots

Callbacks?

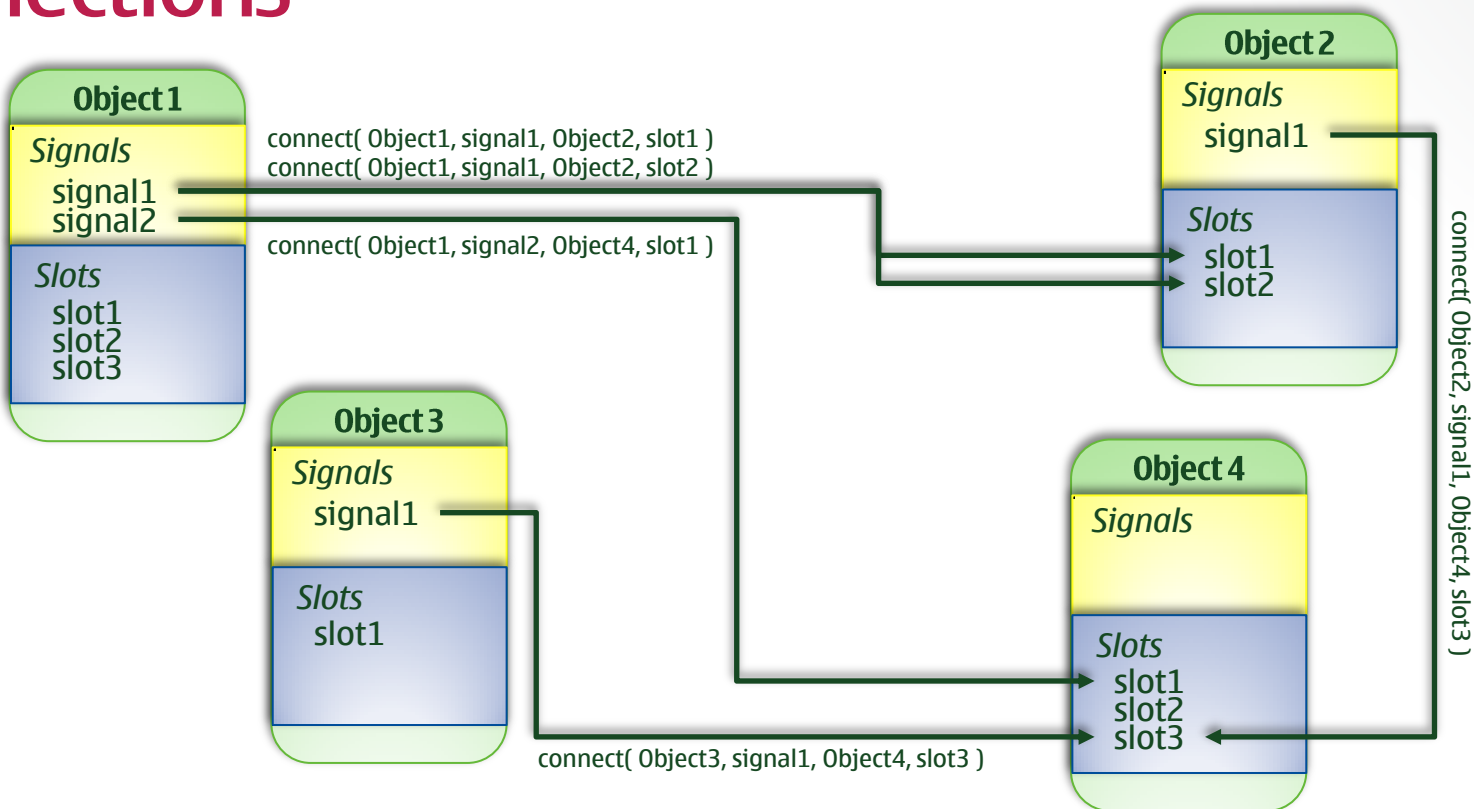
- **Traditional callbacks**
 - Callback is pointer to a function
 - Called when appropriate (event notification, ...)
- **Problems**
 - **Not type-safe:** does the caller use the correct arguments?
 - **Less generic:** callback strongly coupled to processing function. Processing function must know which callback to call.

Signals & Slots

- **Signal**
 - **Emitted when a particular event occurs** (e.g., `clicked()`)
 - Qt widgets: predefined signals
 - Also create your own signals
- **Slot**
 - **Function called in response to a signal**
 - Qt widgets: predefined slots (e.g., `quit()`)
 - Also create your own slots
- **Connection signals ↔ slots established by developer, handled by Qt framework**

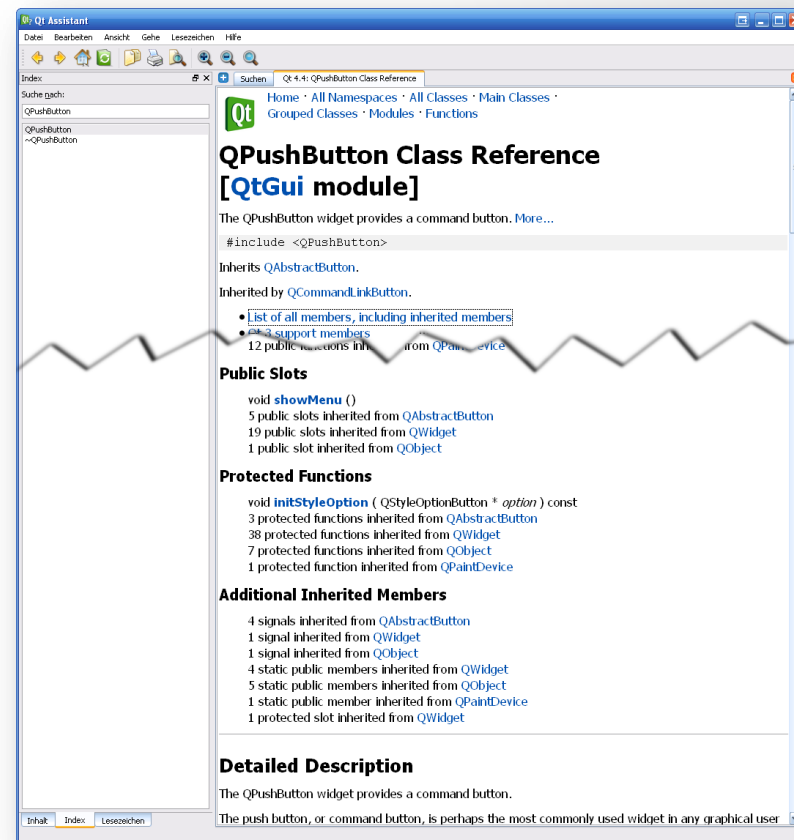


Connections

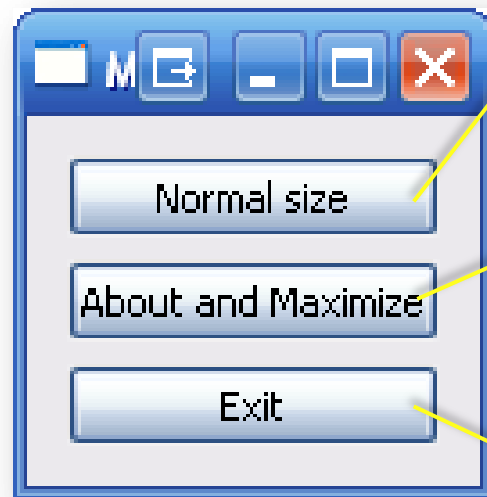


Find Signals and Slots

- Look up signals and slots of Qt classes in the documentation



Example: Multiple Signal-Slot Connections

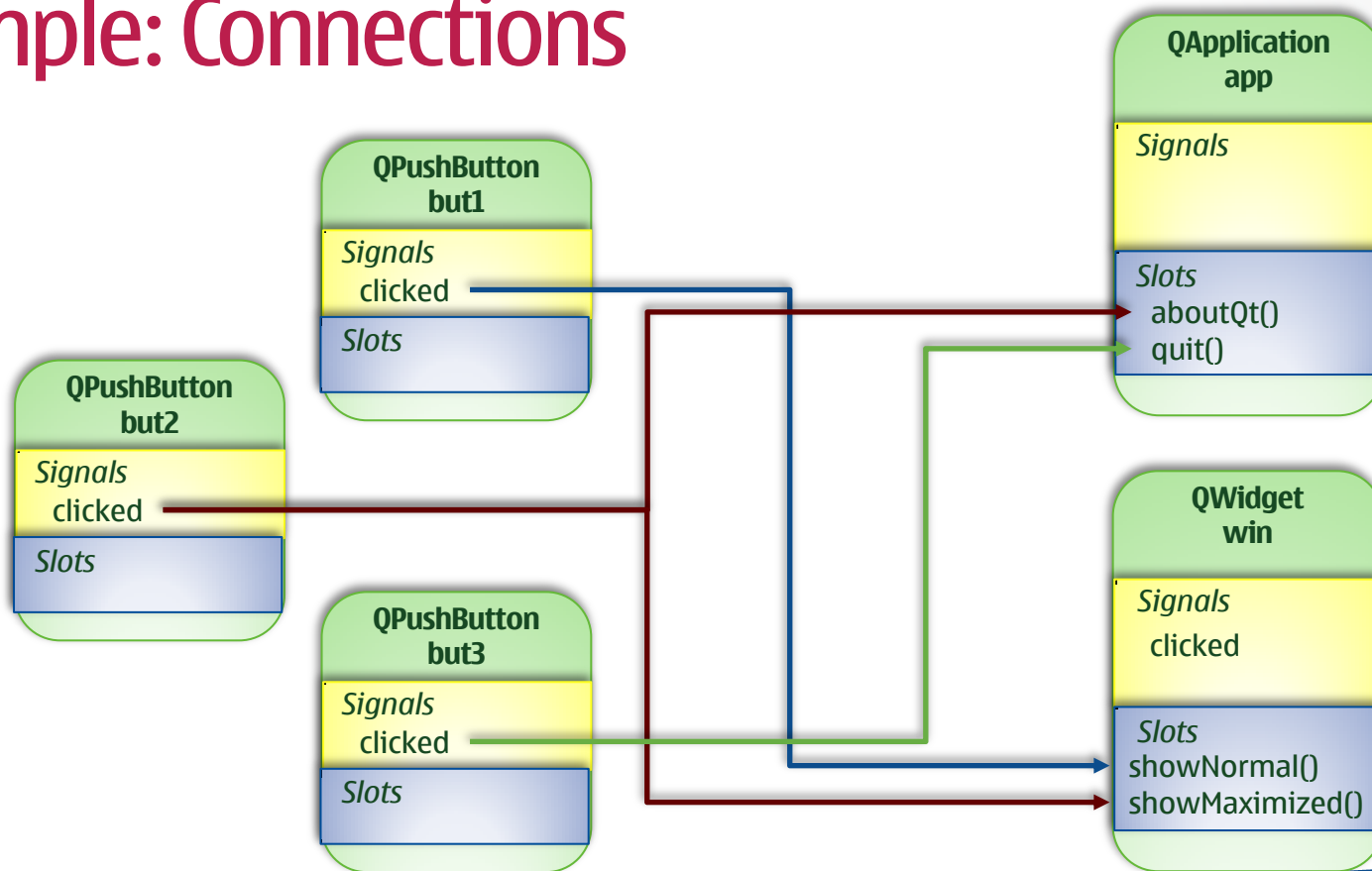


Restore window to
normal size

Show an about dialog
and then maximize the
window

Exit the application

Example: Connections





```
#include <QApplication>
#include <QPushButton>
#include <QVBoxLayout>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QWidget* win = new QWidget();
    QVBoxLayout* layout = new QVBoxLayout(win);

    QPushButton* but1 = new QPushButton("Normal size");
    but1->resize(150, 30);
    layout->addWidget(but1);

    QPushButton* but2 = new QPushButton("About and Maximize");
    but2->resize(150, 30);
    layout->addWidget(but2);

    QPushButton* but3 = new QPushButton("Exit");
    but3->resize(150, 30);
    layout->addWidget(but3);

    QObject::connect(but1, SIGNAL(clicked()), win, SLOT(showNormal()));
    QObject::connect(but2, SIGNAL(clicked()), &app, SLOT(aboutQt()));
    QObject::connect(but2, SIGNAL(clicked()), win, SLOT(showMaximized()));
    QObject::connect(but3, SIGNAL(clicked()), &app, SLOT(quit()));

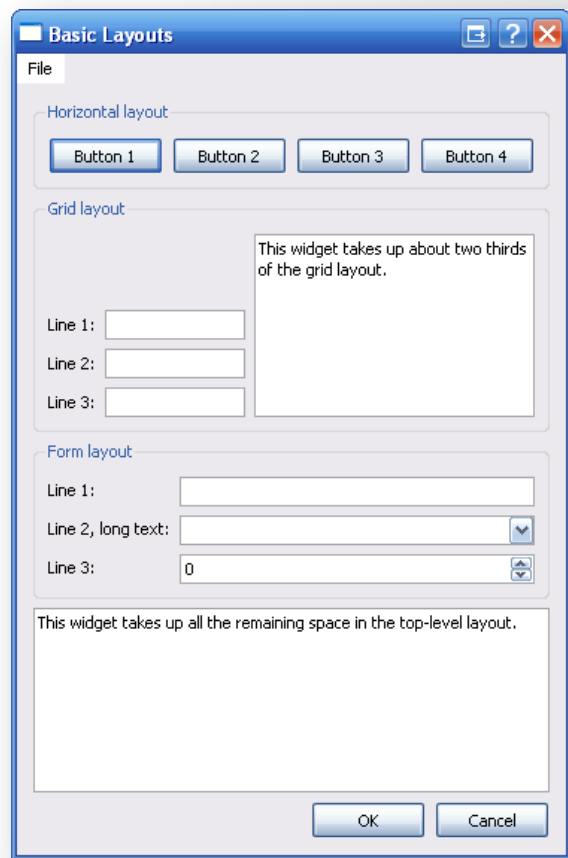
    win->show();

    return app.exec();
}
```

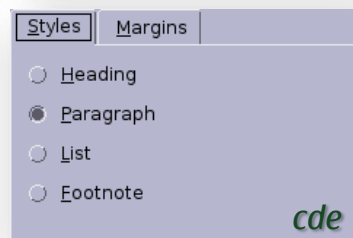
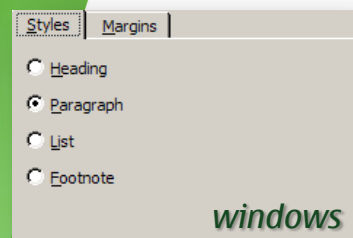
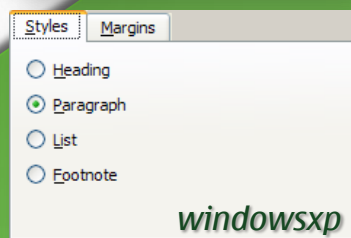
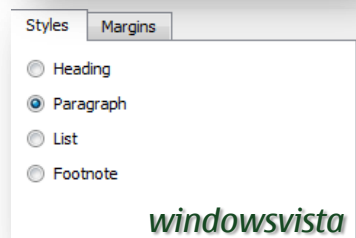
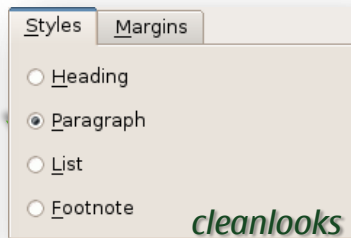
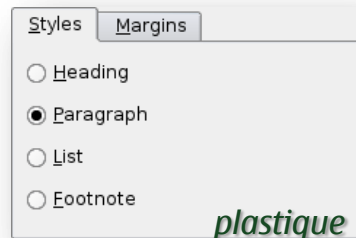
**aboutQt() and
showMaximized()
executed one after
another**

Layouts

- ! **Most common layouts**
 - **QHBoxLayout:** horizontal, from left to right (right to left for some cultures)
 - **QVBoxLayout:** vertical, top to bottom
 - **QGridLayout:** grid
 - **QFormLayout:** manages input widgets and associated labels
 - **QStackedLayout:** widgets on top of each other, switch index of currently visible widget
- **Layouts can be nested**
 - Add new layout to existing layout like a widget:
`l->addLayout(l2)`



Styles



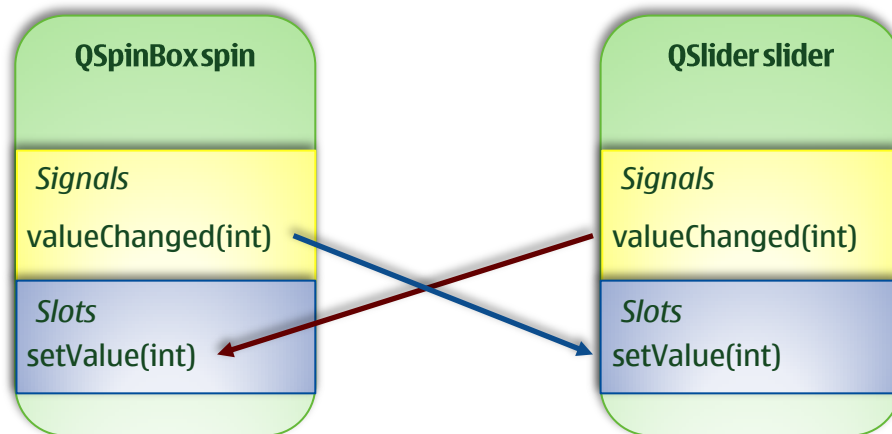
Style can be set using
`-style <name>` command-line option.

Windows XP/Vista/7 and Macintosh styles are only available on native platforms (relies on platform's theme engines)

It's possible to create own styles.

Example 2

- **Synchronize two widgets**
 - Changing the value in one automatically changes the other



```
#include <QApplication>
#include <QHBoxLayout>
#include <QSpinBox>
#include <QSlider>

int main(int argc, char *argv[]) {
    QApplication app(argc, argv);
    QWidget* win = new QWidget();
    win->setWindowTitle("Synchronized Widgets");

    QHBoxLayout* layout = new QHBoxLayout(win);

    QSpinBox* spin = new QSpinBox();
    spin->setMinimum(0);
    spin->setMaximum(100);
    layout->addWidget(spin);

    QSlider* slider = new QSlider(Qt::Horizontal);
    slider->setMinimum(0);
    slider->setMaximum(100);
    layout->addWidget(slider);

    QObject::connect( spin, SIGNAL( valueChanged(int) ),
                      slider, SLOT( setValue(int) ) );
    QObject::connect( slider, SIGNAL( valueChanged(int) ),
                      spin, SLOT( setValue(int) ) );

    spin->setValue(50);

    win->show();
    return app.exec();
}
```



Signal Parameters



Transmit additional information

```
QObject::connect( spin, SIGNAL( valueChanged(int) ),  
                  slider, SLOT( setValue(int) ) );
```

- Specify type of argument(s)
- Signal has to be compatible to the slot
(same parameter type(s))

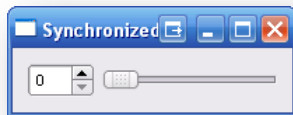
Signal Parameters

- **Types not automatically casted by Qt**
 - Signals & slots with exactly the specified parameters have to exist
 - **Otherwise:** no compilation error / warning
 - **But:** warning at runtime when executing connect():

```
QObject::connect: Incompatible sender/receiver arguments  
QSpinBox::valueChanged(QString) --> QSlider::setValue(int)
```

- → **Signals & slots are type safe**
 - No connection if either sender or receiver doesn't exist
 - Or if signatures of signal and slot do not match
- **connect() returns boolean value indicating success**

Signal & Slot Processing



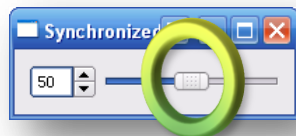
`setValue(50)` is called in the source code



QSpinBox emits `valueChanged(int)` signal with `int` argument of 50

signal → slot

Argument is passed to QSlider's `setValue(int)` slot, sets slider value to 50



QSlider emits `valueChanged(int)` signal with `int` argument of 50

signal → slot

Argument is passed to QSpinBox's `setValue(int)` slot, but value is already set to 50.
→ doesn't emit any further signal to prevent infinite recursion.

Signals & Slots

- **Type safe**
 - Signal signature must match signature of receiving slot
 - (Slot might also have shorter signature and ignore rest of the arguments)
- **Loosely coupled**
 - Emitter doesn't know or care which slots receive signal
 - Information encapsulation
- **Integrated, independent components**
 - Slots are normal C++ member functions
 - Don't know if signals are connected to them

Manual Signal & Slots

counter.h

```
#ifndef COUNTER_H
#define COUNTER_H

#include <QObject>

class Counter : public QObject
{
    Q_OBJECT

public:
    Counter() { m_value = 0; }
    int value() const { return m_value; }

public slots:
    void setValue(int value);

signals:
    void valueChanged(int newValue);

private:
    int m_value;
};

#endif // COUNTER_H
```

counter.cpp

```
#include "counter.h"

void Counter::setValue(int value)
{
    if (value != m_value)
    {
        m_value = value;
        emit valueChanged(value);
    }
}
```

Own Class that represents
behaviour of Example 2

main.cpp

```
#include <QtGui/QApplication>
#include <QMessageBox>
#include "counter.h"

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    Counter a, b;
    QObject::connect(&a, SIGNAL(valueChanged(int)),
                    &b, SLOT(setValue(int)));

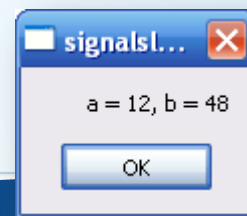
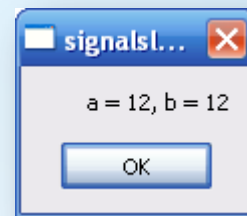
    a.setValue(12);      // a.value() == 12, b.value() == 12

    QMessageBox msgBox;
    msgBox.setText("a = " + QString::number(a.value()) + ", b = " + QString::number(b.value()));
    msgBox.exec();

    b.setValue(48);      // a.value() == 12, b.value() == 48

    msgBox.setText("a = " + QString::number(a.value()) + ", b = " + QString::number(b.value()));
    msgBox.exec();

    app.quit();
    return 1;
}
```



A blue-tinted photograph of a group of people sitting together, possibly in a meeting or workshop. The focus is on the legs and hands of the individuals. One person in the foreground is wearing jeans and a black Converse shoe. Another person's hand is visible, gesturing. The text 'Meta Objects' is overlaid on the left side of the image.

Meta Objects

Qt Meta-Object System

- **C++ extended** with meta-object mechanism:
Introspection
 - Obtain **meta-information about QObject subclasses** at runtime
 - **Used for:** getting list of signals & slots, properties and text translation



QObject

- **Meta information** not supported by standard C++
 - QObject class
 - Base class for objects that use meta-object system
 - Q_OBJECT macro
 - Enables meta-object features
 - Without ;
 - “moc” tool
 - Parses Q_OBJECT macro
 - Extends source code with additional functions (extra files)
 - Removes the signals, slots and emit keywords so compiler sees standard C++
- **Advantage:** works with any C++ compiler

```
#include <QObject>
class Counter : public QObject
{
    Q_OBJECT
    [...]
};
```

Meta-Object Features

- **Features provided by meta-object code:**
 - **Signals and slots**
 - **metaObject()** – returns associated meta-object for the class
 - **QMetaObject::className()** – returns class name as a string, without requiring native run-time type information (RTTI) support through the C++ compiler
 - **inherits()** – returns whether this instance inherits the specified QObject class
 - **tr()** – translate strings
 - **setProperty()** and **property()** – dynamically set and get properties by name

Casting

- **Meta-object information can be used for casting:**

```
if (widget->inherits("QAbstractButton")) {  
    QAbstractButton *button = static_cast<QAbstractButton*>(widget);  
    button->toggle();  
}
```

- **Similar, but less error-prone:**

```
if (QAbstractButton *button = qobject_cast<QAbstractButton*>(widget))  
    button->toggle();
```

More on... Signals

- **Emitted signal**
 - Slot usually **executed immediately** (like normal function call)
 - Code after emit keyword executed after all slots returned
 - **Queued connections:** slots executed later
- **1 signal → n slots**
 - Slots executed **one after another** (arbitrary order)
- **Implementation**
 - **Automatically generated** by moc
 - **Define in .h, do not implement** in .cpp file
 - Can never have return values (→ use void)
 - **Good style:** arguments should not use special types (reusability)



```
#include <QObject>
class Counter : public QObject {
    Q_OBJECT
    [...]
signals:
    void valueChanged(int newValue);
    [...]
};
```

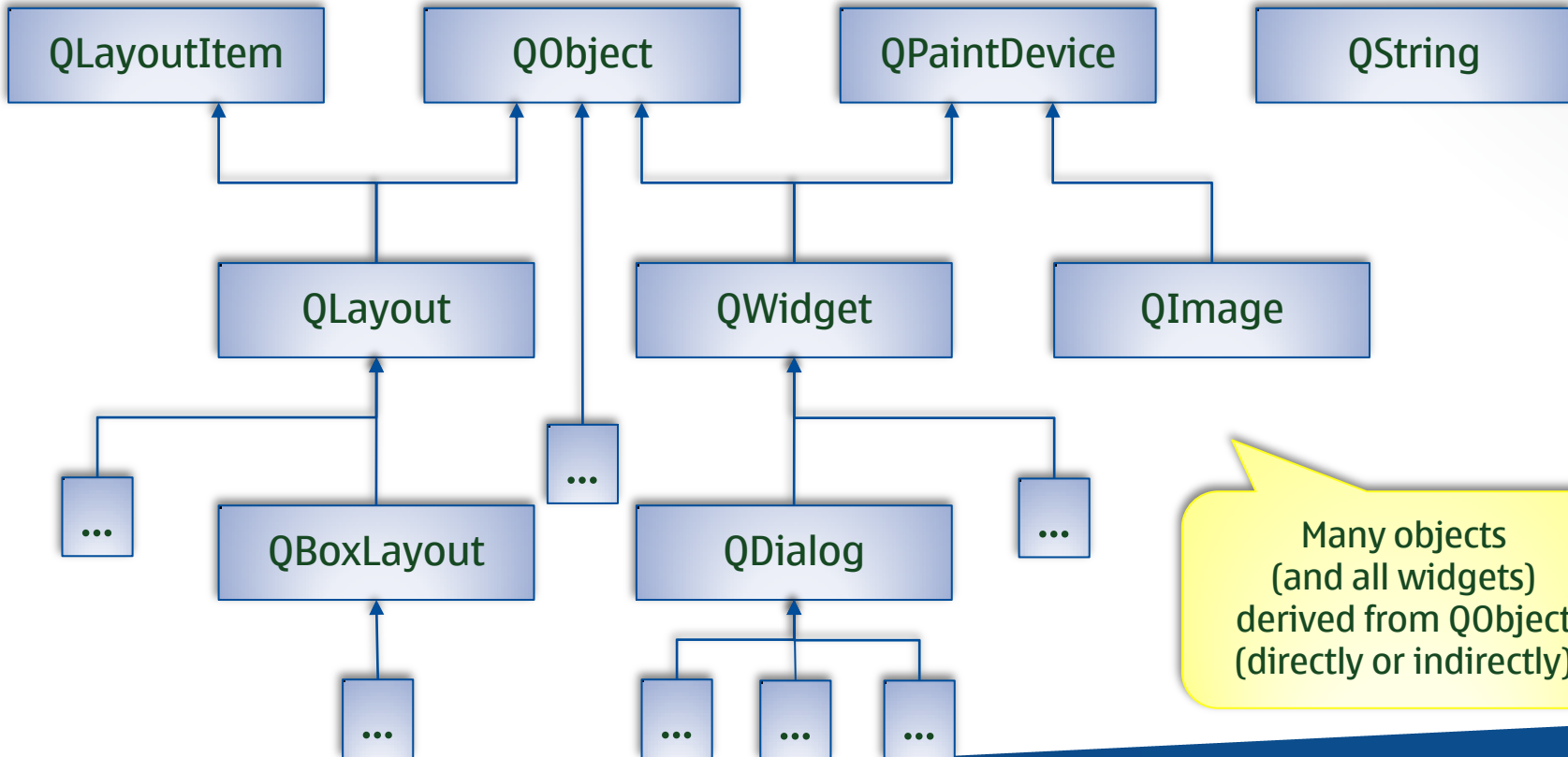
More on... Slots

- **C++ function**
 - Can be called directly/normally
 - If connected to and invoked by signal: works regardless of access level. Arbitrary class can invoke private slot of an unrelated class instance.
 - Slots can be virtual, but not static
- **Overhead compared to direct call**
 - **Does not matter in practice**
 - Around 10x slower than direct, non-virtual call
 - Sounds a lot, but isn't compared to for example new/delete operations
 - i586-500: emit per second
2,000,000 signals → 1 slot.
1,200,000 signals → 2 slots.

```
#include <QObject>
class Counter : public QObject {
    Q_OBJECT
    [...]
public slots:
    void setValue(int value);
    [...]
};
```



Qt Class Hierarchy

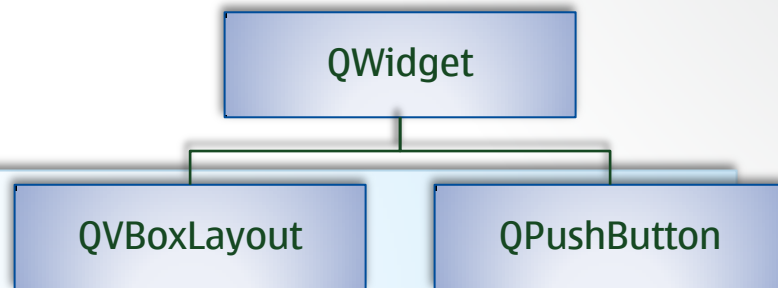


Memory Management

- **Parent-child hierarchy implemented in QObject**
 - **Initialization with pointer to parent QObject** → parent adds new object to list of its children
 - **Delete parent:** automatically deletes all its children (recursively)
 - **Delete child:** automatically removed from parent's child list
 - → Some memory management handled by Qt, only **delete objects created with new without parent**
- **Widgets**
 - **Parent has additional meaning:** child widgets shown within parent's area

Example: Parent-Child

```
QWidget* win = new QWidget();  
QVBoxLayout* layout = new QVBoxLayout(win);  
QPushButton* but = new QPushButton("Label");  
layout->addWidget(but);  
win->show();
```



- Layout is a child of parent widget
- Push button added to layout, but widget takes ownership

```
QWidget* win = new QWidget();  
QVBoxLayout* layout = new QVBoxLayout(win);  
QPushButton* but = new QPushButton("Label", win);  
win->show();
```

- **Directly adding push button to the parent widget:**
 - Also displays push button, but not managed by layout manager

Example: Parent-Child II

- **Helpful:** print parent-child relationship

```
QWidget* win = new QWidget();  
QVBoxLayout* layout = new QVBoxLayout(win);  
QPushButton* but = new QPushButton("Label");  
layout->addWidget(but);  
win->dumpObjectTree();
```



Console

```
QWidget::  
    QVBoxLayout::  
        QPushButton::
```

Creating Objects

- **Objects inheriting from `QObject` are allocated on the heap using `new`**
 - If a parent object is assigned, it takes ownership of the newly created object – and eventually calls `delete`

```
QLabel *label = new QLabel("Some Text", parent);
```

- **Objects not inheriting `QObject` are allocated on the stack, not the heap**

```
QStringList list;  
QColor color;
```

- **Exceptions**
 - `QFile` and `QApplication` (inheriting `QObject`) are usually allocated on the stack
 - Modal dialogs are often allocated on the stack, too

Thank You.

