

Workshop: Entwickeln mit Qt 4



Torsten Rahn <torsten.rahn@credativ.de>
Daniel Molkentin <molkentin@kde.org>

Das Wichtigste zuerst!

- Haben Sie einen Laptop dabei?
 - Nein? Suchen Sie sich schnell einen netten Nachbarn :)
- Haben Sie Qt installiert?
 - Dann wird's aber höchste Eisenbahn!
- Ist es auch tatsächlich Qt 4?
 - Einfach mal `assistant` aufrufen
 - Klappt nicht? Prüfen Sie ob alle Entwicklungspakete von Qt installiert sind.
- Haben Sie einen geeigneten Editor?
 - **Kate** bietet alles, was wir brauchen

Qt laut Wikipedia

Qt (Toolkit)

Qt ist eine [Klassenbibliothek](#) für die plattformübergreifende Programmierung graphischer Benutzeroberflächen ([GUIs](#)) unter [C++](#). Qt wird besonders in den Bibliotheken des [K Desktop Environments](#) verwendet.

Qt wird von der norwegischen Firma [Trolltech](#) (früher *Quasar Technologies*) entwickelt, und ist für verschiedene Betriebssysteme bzw. Grafikplattformen wie [X11 \(Linux\)](#), [Mac OS X](#) und [Windows](#) erhältlich. Neben der Entwicklung von graphischen Benutzeroberflächen ([GUI](#)) bietet Qt umfangreiche Funktionen zur Internationalisierung sowie Datenbankfunktionen und XML-Unterstützung an.

Die Klassenbibliothek steht sowohl unter der [GNU General Public License](#) (GPL) als auch unter einer kommerziellen Lizenz, welche allerdings nur benötigt wird, falls mit der Umgebung Produkte entwickelt werden sollen, die nicht wiederum unter einer [freien Lizenz](#) stehen. (siehe [Duales Lizenzsystem](#))

Qt verwendet eine Erweiterung der Programmiersprache [C++](#). Es existieren auch Implementierungen für [Python](#), [Ruby](#), [C](#), [C#](#) und [Perl](#), die allerdings nicht von Trolltech betreut werden.

Qt	
Entwickler:	Trolltech
Aktuelle Version:	4.1.1 (21. Februar 2006)
Betriebssystem:	X11 (Unix-Derivate) , Mac OS X , Windows
Kategorie:	Klassenbibliothek
Lizenz:	Duales Lizenzsystem (Proprietär und GPL/QPL)
Deutschsprachig:	nein
Website:	Trolltech ↗

Plattformunabhängigkeit

Platforms 32bit			
	GCC	Vendor	3rd party
Microsoft Windows	gcc (MinGW)	MS Visual Studio/Visual C++	Intel icc
Linux	gcc	-	-
Apple Mac OS X	gcc	-	-
IBM AIX	gcc	IBM xLC	-
HP HP-UX	gcc	HP aCC	-
SGI IRIX	gcc	SGI MipsPRO	-
Sun Solaris	gcc	Sun CC	-
FreeBSD	gcc	-	Intel icc
Platforms 64bit			
	GCC	Vendor	3rd party
Microsoft Windows	-	MS Visual Studio/Visual C++	-
IBM AIX	gcc	IBM xLC	-
HP-UX	-	HP acc	-
SGI IRIX	gcc	SGI MipsPRO	-
Sun Solaris	gcc	Sun cc	-
Linux	gcc	-	Intel icc

...sowie Linux auf Embedded-Geräten mit Framebuffer (QtopiaCore)

Verbreitung von Qt

Freie Software

- Freie Qt-Software (z.B. Scribus, Earth 3D)
- KDE und KDE-Programme (-><http://www.kde-apps.org>)

Kommerzielle Software

- Google Earth, Adobe Photoalbum (nur Windows)
- Opera (nur Linux)
- Skype, Brockhaus Multimedia 2006, Eagle, Mainactor, ...
- Inhouse-Anwendungen

Bestandteile von Qt 4

Qt 4 ist modular aufgebaut:

QtCore	Core non-GUI classes used by other modules
QtGui	Graphical user interface components
QtNetwork	Classes for network programming
QtOpenGL	OpenGL support classes
QtSql	Classes for database integration using SQL
QtSvg	Classes for displaying the contents of SVG files
QtXml	Classes for handling XML
QtDesigner	Classes for extending <i>Qt Designer</i>
QtUiTools	Classes for handling <i>Qt Designer</i> forms in applications
QtAssistant	Support for online help
Qt3Support	Qt 3 compatibility classes
QtTest	Tool classes for unit testing

Der Kern von Qt 4: QtCore

Die Basisbibliothek heißt **QtCore**

- **QtCore** enthält nicht-grafische Klassen (z.B. String- und Container-Klassen, vgl. STL)

Beispiele:

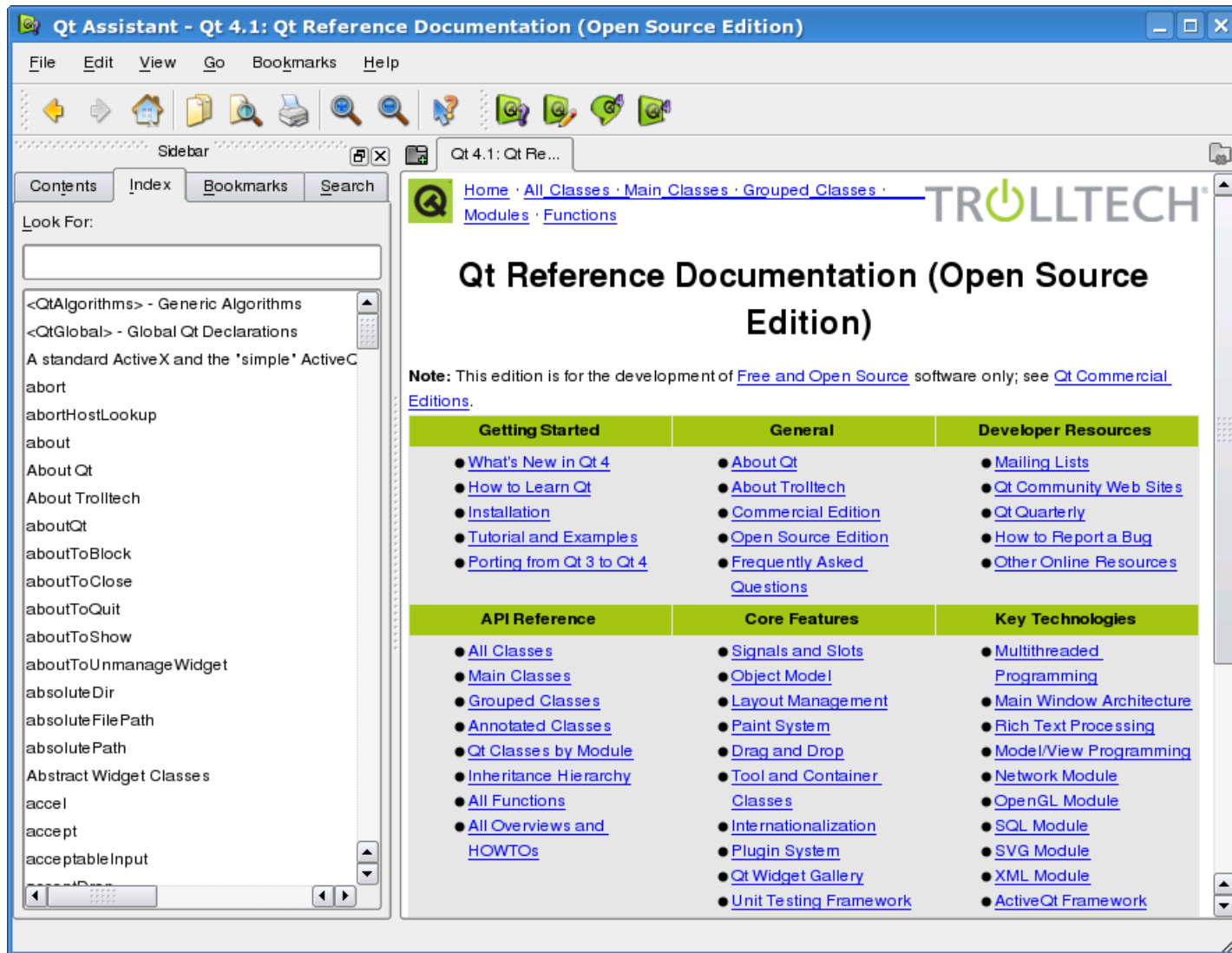
- **QString**: Zeichenketten kodiert in Unicode. Enthält Funktionen zur Stringmanipulation und zur Konvertierung von Codierungen
- **QObject**: Objektbaum, „Signals & Slots“, Introspektion, Objekteigenschaften
- 200 nützliche Klassen: QList, QSettings, QVariant, QFile, QThread

Qt grafisch: QtGui

QtGui enthält alle grafischen Klassen:

- Zentrale Klasse ist QWidget: Sie erweitert QObject. Alle anderen Widget-Klassen leiten sich von ihr ab
- **QPainter** malt auf **QPaintDevice**-Klassen: QWidget, QPixmap, QPrinter, QGLWidget, etc.
- Weitere Klassen: QPushButton, QTextEdit, QDialog, QIcon, QPainter, etc.

Für (fast) alle Fälle: Qt Assistant



Dokumentation auch per Browser: <http://doc.trolltech.com/>

Hallo Welt!

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

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

    QPushButton hello("Hallo Welt!");
    hello.resize(100,30);
    hello.show();

    return app.exec();
}
```

- Ein `#include` pro Klasse
- `QApplication` enthält die *Haupt-Ereignisschleife* die *Events* zwischen Programm und Betriebssystem vermittelt
- z.B. mit Hilfe des Editors **Kate** eingeben
- als **main.cpp** abspeichern
- Problem: Wie kompilieren?

Projektverwalter qmake

```
#####  
# Automatically generated by qmake  
#####  
  
TEMPLATE = app  
TARGET += .  
DEPENDPATH += .  
INCLUDEPATH += .  
  
# Input  
SOURCES += main.cpp
```

- Jede Plattform hat eigene Tools (*Compiler, Linker, make-Programm*)
- Qt benötigt zusätzlich Tools (*moc, uic*)

Lösung:

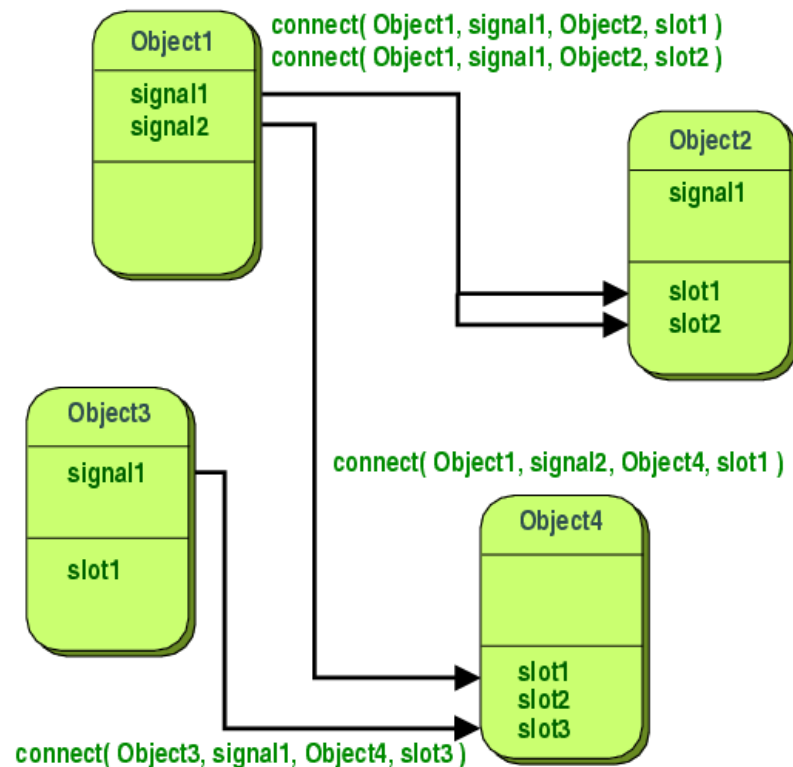
- `qmake -project` generiert Projektdatei (.pro)
- `qmake` erstellt aus ihr ein Makefile
- `make` erstellt mit der *Toolchain* ausführbare Dateien

Mehr zu Widgets

- QWidget besitzt zahlreiche Methoden, darunter:
 - ein Rechteck (`geometry()`, `setGeometry()`)
 - einen Font (`font()`, `setFont()`)
 - eine Standardfarben-Palette (`palette()`, `setPalette()`)
 - weitere Eigenschaften, die ihr Aussehen bestimmen
 - virtuelle Funktionen, die aufgerufen werden, wenn die Widgetfläche oder ein Teil davon gezeichnet werden soll (`paintEvent()`), bei Mausklicks (`mousePressEvent()`, `mouseReleaseEvent()`), bei Drücken der Tastatur (`keyPressEvent()`), usw.

Signale und Slots

- Jede Klasse, die von **QObject** erbt, kann eine Sammlung von *Signalen und Slots* besitzen
- Signale einer Klasse werden zur Laufzeit an Slots gebunden, ohne dass einer Klasse über die andere genaue Informationen benötigt!



Hallo Welt, und wech...

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

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

    QPushButton hello("Hallo Welt!");
    hello.resize(100,30);
    hello.show();
    QObject::connect(&hello,
        SIGNAL(clicked()),
        qApp, SLOT(quit()));

    return app.exec();
}
```

- `connect()` erstellt Verbindungen zwischen Signalen und Slots
- `qApp`: Zugriff auf `QApplication`-Objekt
- Jetzt tut unser Programm schon etwas. Klicken Sie mal auf den Button!
- Aber wie macht man Signale oder Slots selber?

Signale und Slots - Deklaration

```
#include <QObject>

class Counter : public QObject
{
    Q_OBJECT
public:
    Counter(QObject *parent = 0)
    { m_val = 0; }
    int value() const {return m_val};
public slots:
    void setValue(int);
signals:
    void valueChanged(int);
private:
    int m_val;
}
```

- Klasse *muss* von **QObject** erben
- **Q_OBJECT**-Macro
 - Hinweis an *moc*: erzeugt Meta-Informationen für die Klasse und Implementierung für Signale (eigene Datei)
 - Fügt **QMetaObject** ein (Präprozessor)
- **slots**: expandiert zu „“
- **signals**: „protected“
- Ergebnis: Standard C++

Signale und Slots - Implementierung

```
// in counter.cpp
#include "counter.h"

void Counter::setValue(int val)
{
    if (m_val != val) {
        m_val = val;
        emit valueChanged(val);
    }
}

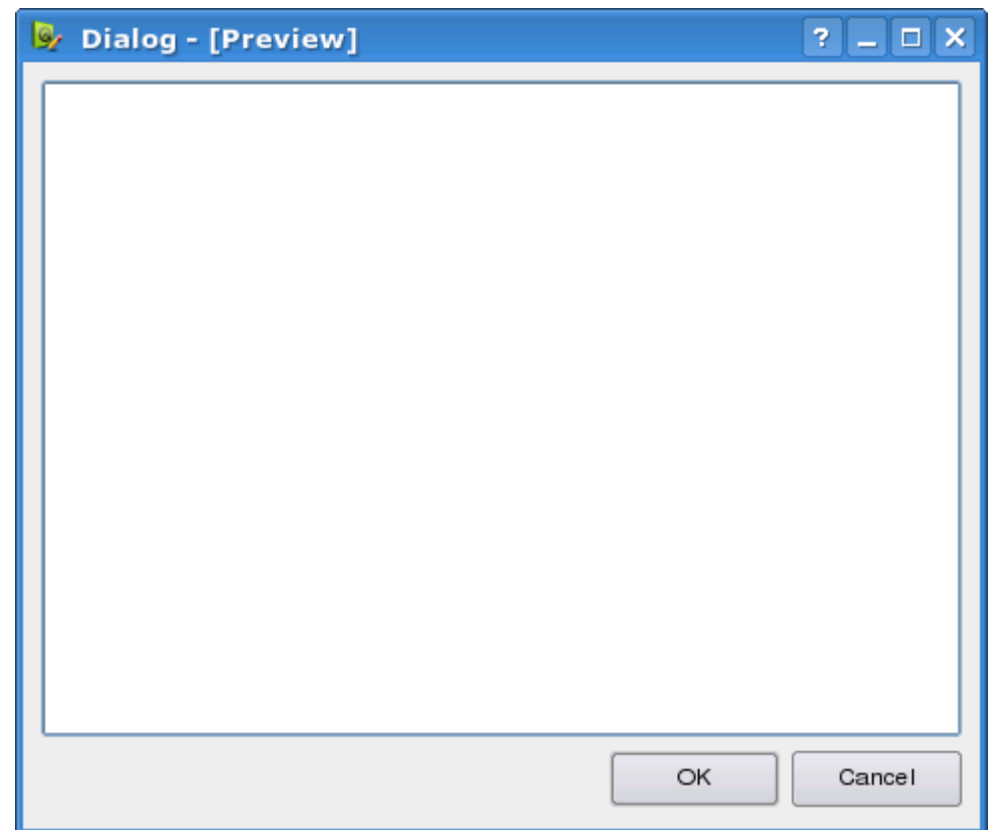
// in einer beliebigen Methode
Counter a,b;
QObject::connect(&a,
                 SIGNAL(valueChanged(int)),
                 &b, SLOT(setValue(int)));
```

- emit verdeutlicht nur Semantik
- a.setValue()-Aufrufe beeinflussen m_val in b
- Umgekehrt jedoch nicht! Wie wäre dies lösbar?
- Versuchen Sie, die Klasse zu implementieren und in main() aufrufen. Verwenden Sie **QDebug()** zur Ausgabe:

```
#include <QDebug>
QDebug() << a.value();
```

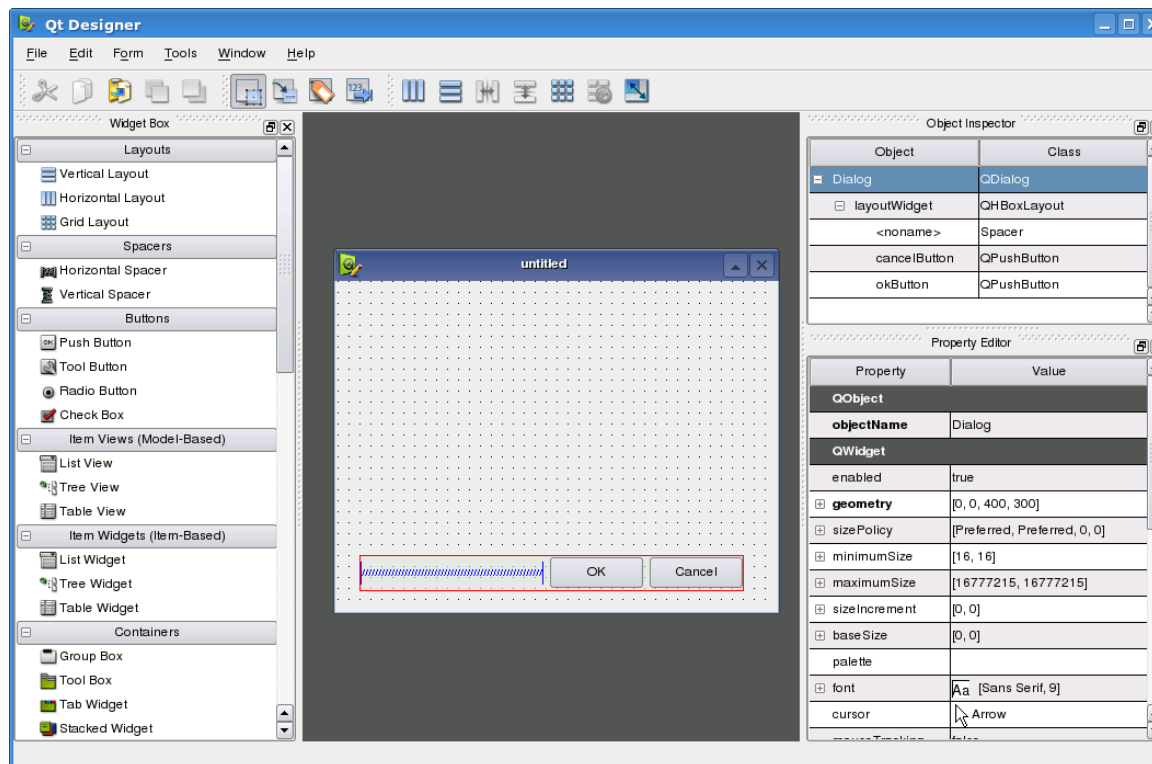
Nie mehr statisch: Layouts

- Hart codierte Widgetgrößen sind problematisch, weil abhängig von DPI-Angaben, Sprache, Schriftgröße
- Außerdem umständlich zu platzieren (GUI-Editoren machen es *etwas* einfacher, aber haben Sie mal unter Windows andere DPI-Größen eingestellt?)
- Lösung: Layouts (vertikal, horizontal, Tabelle), die eine Menge von Widgets verwalten



RAD mit Qt Designer 4

- Designer produziert XML-Datei (.ui)
- User Interface Compiler (uic) erzeugt daraus speziellen *Gluecode* (anders als in Qt 3!), dessen Anatomie wir später klären.



Wir bauen einen Doku-Browser

- Nicht ganz so toll wie der Assistant, aber schon brauchbar
- Der Rohbau: Schritt-für-Schritt-Anleitung:
 - Designer öffnen
 - ein leeres Main-Window auswählen
 - Werkzeugleiste und Menüleiste per Kontextmenü entfernen
 - Elemente einfügen
 - Text-Browser in der Mitte platzieren (Drag & Drop)
 - Zwei Schaltflächen (Push Buttons) nebeneinander links über dem Text-Browser anordnen. Daneben eine Eingabezeile (Line Edit)
 - Unter den Text-Browser rechts eine weitere Schaltfläche, links davon ein lang gestrecktes horizontales Layout

Schaltflächen

- Die oberen beiden Schaltflächen sollen die Navigation realisieren
 - Benennen Sie die rechte Schaltfläche „<<“ und die linke „>>“
- Mit der unteren Schaltfläche verlassen wir unser Programm
 - Benennen Sie es „&Quit“
 - Frage: Warum das &?

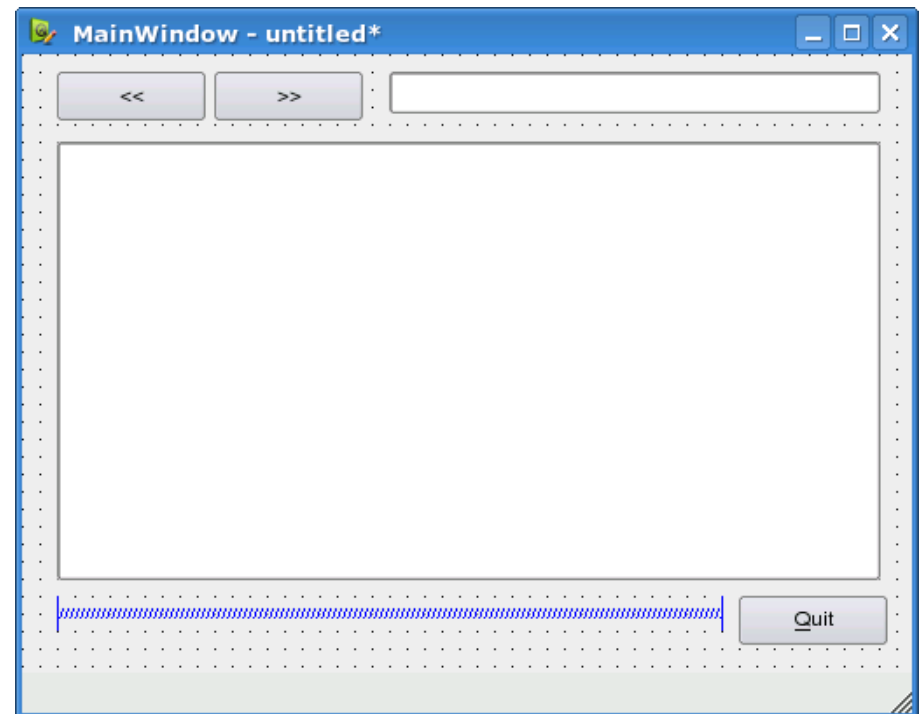
Zwischenstand bisher:



Layouten

- Oberste Widgetzeile auswählen
 - Kontext-Menü => Layout => Lay out horizontally
- Diese Schritte für die Schaltfläche und den Spacer wiederholen
- Nun auf eine freie Fläche auf dem Dialog klicken
 - Kontext-Menü => Layout => Lay out vertically
- Das ist unser globales Layout

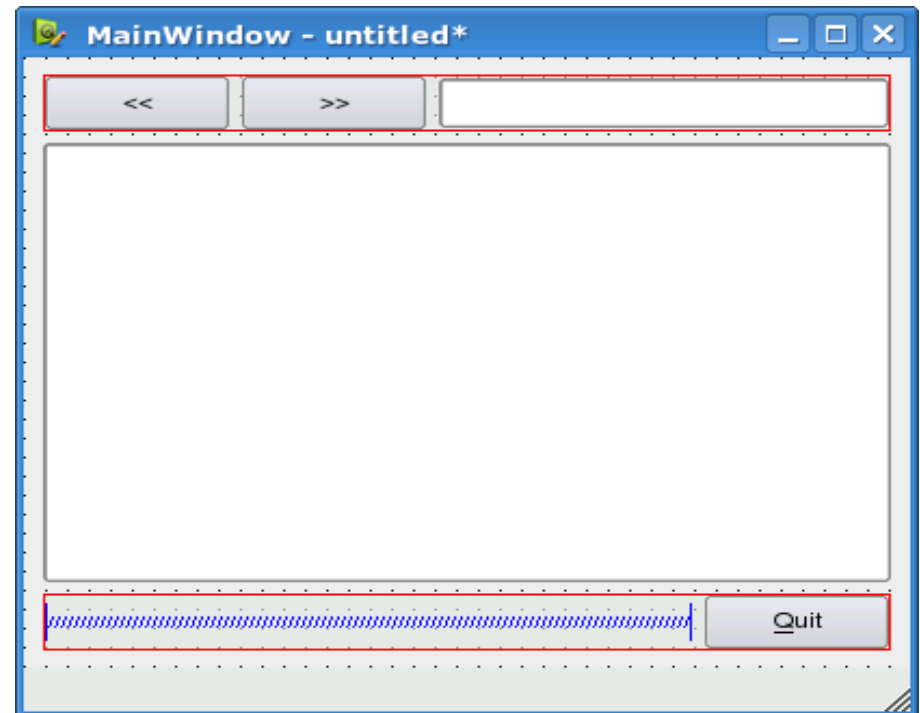
Das haben wir bislang:



Verfeinern

- Setzen Sie im Property-Editor für die beiden Navigations-Schaltflächen die Eigenschaft **enabled** auf **false**, denn gewöhnlich gibt es anfänglich nichts zu navigieren
- gleichsam sollte die Quit-Schaltfläche Standard sein. Also ändern wir den Wert von **default** von **false** auf **true**

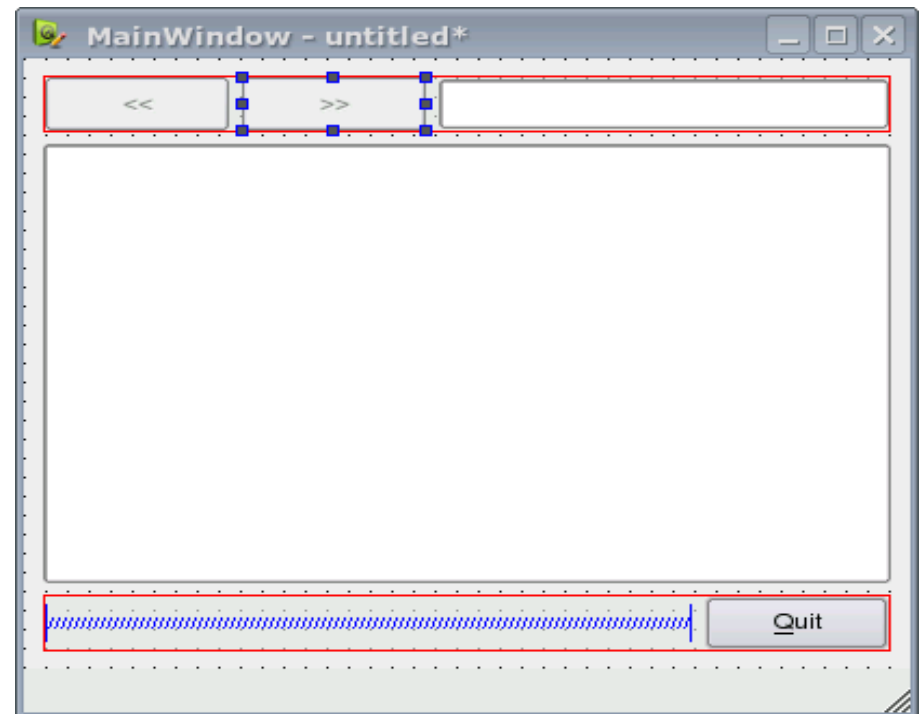
Das haben wir bislang:



Signale und Slots

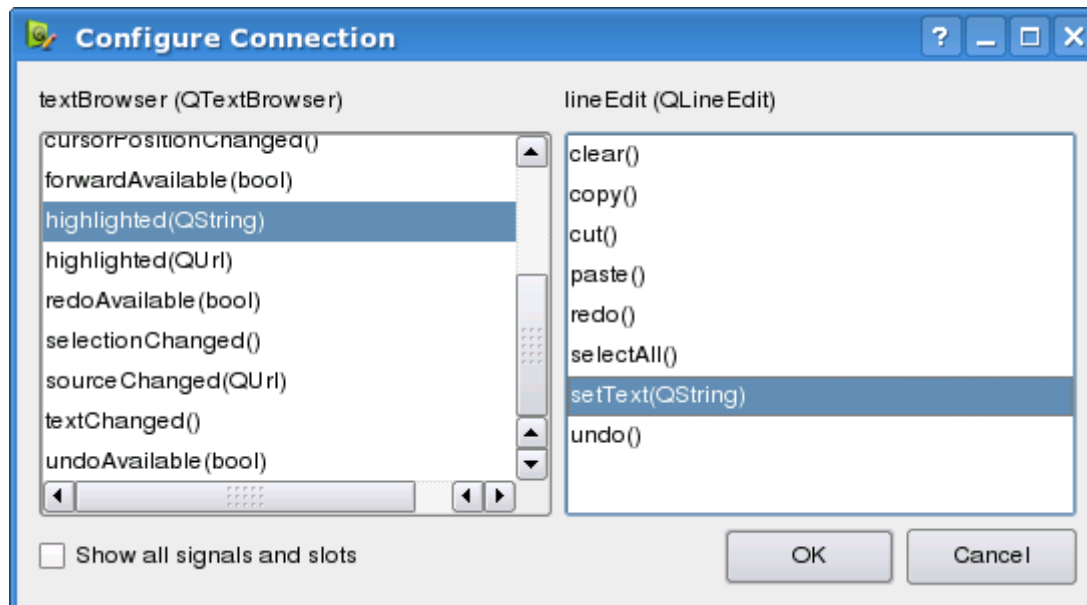
- Jetzt kommen wir an den Punkt, an dem wir die Widgets miteinander verbinden müssen
- Dazu gibt es im Designer den Modus für Signale und Slots (*Edit -> Edit Signals/Slots* oder *F4*)
- Prinzip: Verbindungen zwischen Quell- und Zielwidget ziehen, dann Maustaste loslassen

Das haben wir bislang:



Signale und Slots (II)

- Bedienung eigentlich ganz simpel
 - Quellsignal auswählen (ggf. „Show all Signals and Slots“ anwählen)
 - dazu passendes Zielsignal auswählen
 - Dialog mit „OK“ beenden



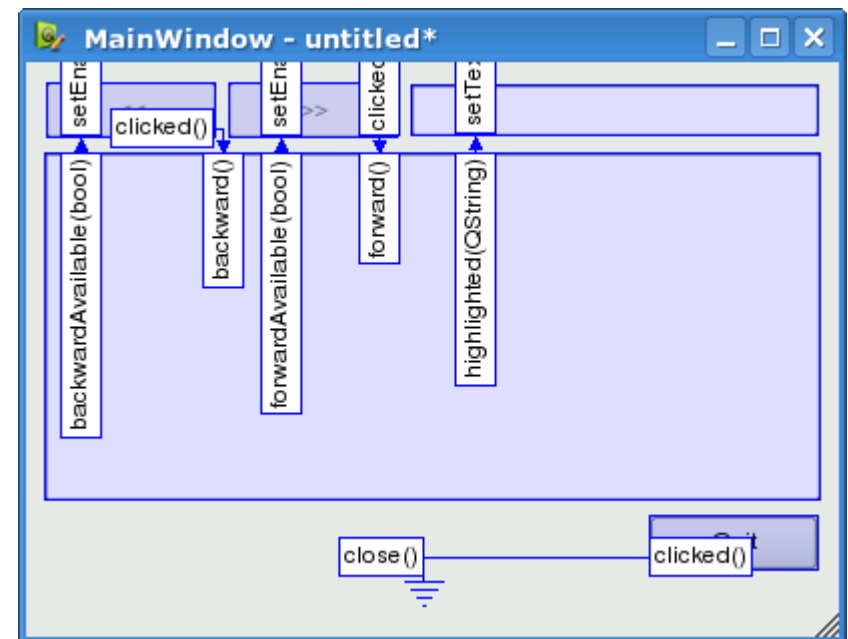
Signale und Slots (III)

- Verbinden Sie nun:
 - vom Text-Browser:
 - highlighted() mit setText() des LineEdits
 - forwardAvailable() mit setEnabled() des „>>“-PushButtons
 - backwardAvailable() mit setEnabled() des „<<“-PushButtons
 - bei den Schaltflächen (PushButtons):
 - clicked() des rechten oberen PushButtons mit QTextBrowser::forward()
 - clicked() des linken oberen PushButtons mit QTextBrowser::backward()
 - clicked() des unteren PushButtons mit close() des MainWindow

Dialog speichern

- Alles verbunden?
- Dann in einem neuen Verzeichnis unter „mainwindow.ui“ speichern
- Im Property-Editor setzten wir den **windowTitle** des MainWindow auf „DocBrowser“
- Nun brauchen wir nur noch eine main()-Funktion, die das Ganze aufruft

Verbindungen im Qt Designer:



Vom Dialog zum Programm

```
#include <QtGui>
#include "ui_mainwindow.h"

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

    QMainWindow w;
    Ui::MainWindow ui;
    ui.setupUi(&w);
    ui.textBrowser->setSource(
        QUrl("/opt/qt41/doc/"
            "html/index.html"));
    w.show();
    return app.exec();
}
```

- QtGui umfasst alle Header von QtGui und QtCore
- Designer generiert die Datei ui_mainwindow.h, falls sie von einer Datei im Projekt inkludiert wird
- Sie enthält Zeiger auf alle vom Designer generierten Widgets sowie die Methode setupUi()
- setupUi() erstellt die Widgets sowie Signal/Slot-Verbindungen

Homerun!

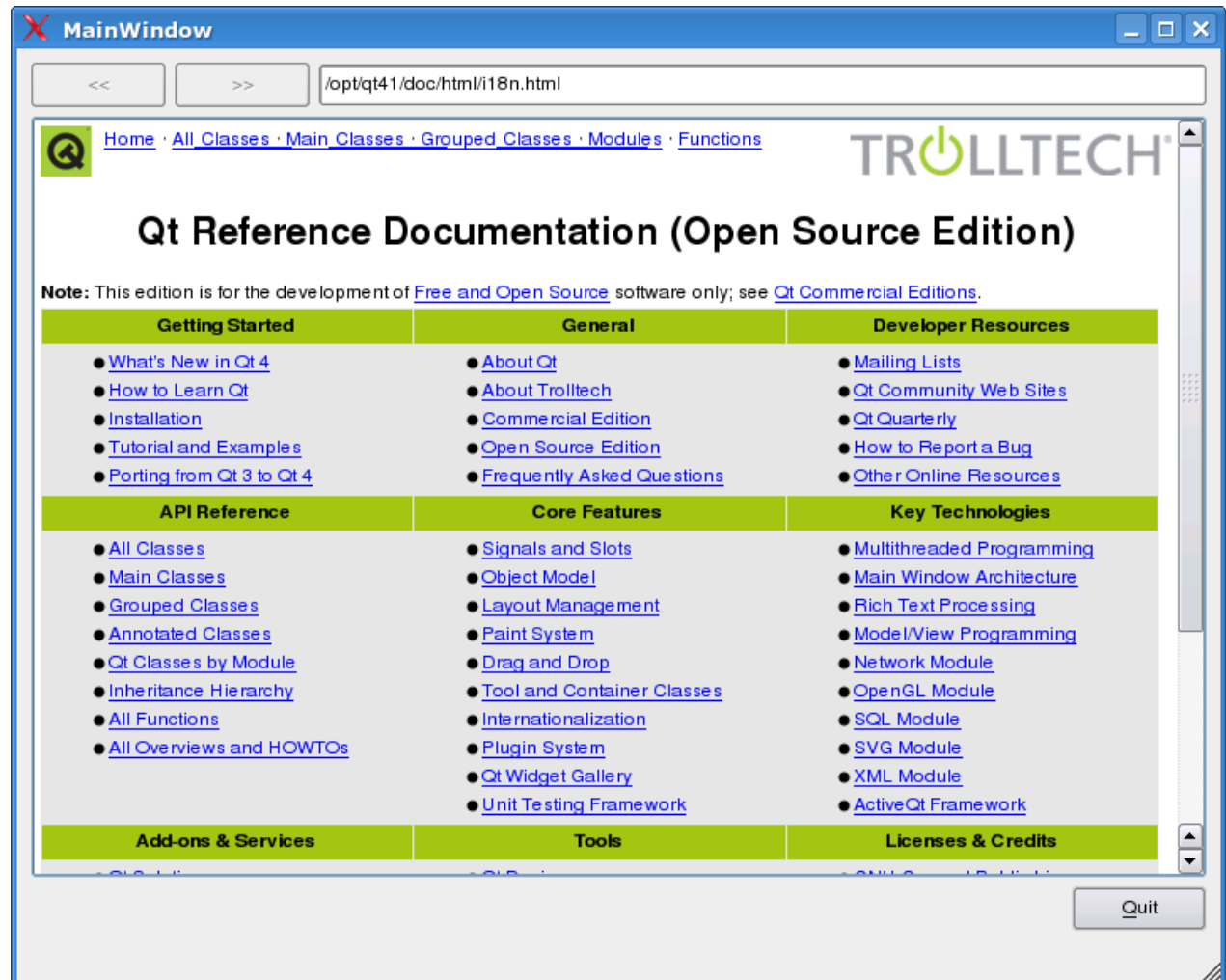
- Nun nur noch kompilieren...

- qmake -project
- qmake
- make

- ...und starten

- ./docbrowser

- Fertig!



Das wars!

- Literatur
 - Offizielle Qt-4-Dokumentation
<http://doc.trolltech.com> (engl.)
 - „Qt 4“, OpenSource Press, ISBN 3-937514-12-0 (dt., ca. ab Juli)
 - „Entwickeln von Applikationen unter Qt 4“: <http://www.pro-linux.de/work/qt4/qt4-teil1.html>
- Unterstützung
 - <http://www.qtcentre.org> (engl.)
 - <http://www.qtforum.de> (dt.)
 - IRC: #qt auf irc.freenode.net



Torsten Rahn

<rahn@kde.org>

Daniel Molkentin

<molkentin@kde.org>