





Ablauf

- Einleitung
 - wer, was, wann, wo
- kleines Beispiel
- Schlüsselkonzepte von Qt
 - Die Klasse QWidget
 - Events
 - Meta Object System
 - Signale und Slots
 - Threading



Einleitung

Ressourcen und Quellen aus denen dieser Vortrag entstanden ist:

qt.nokia.com



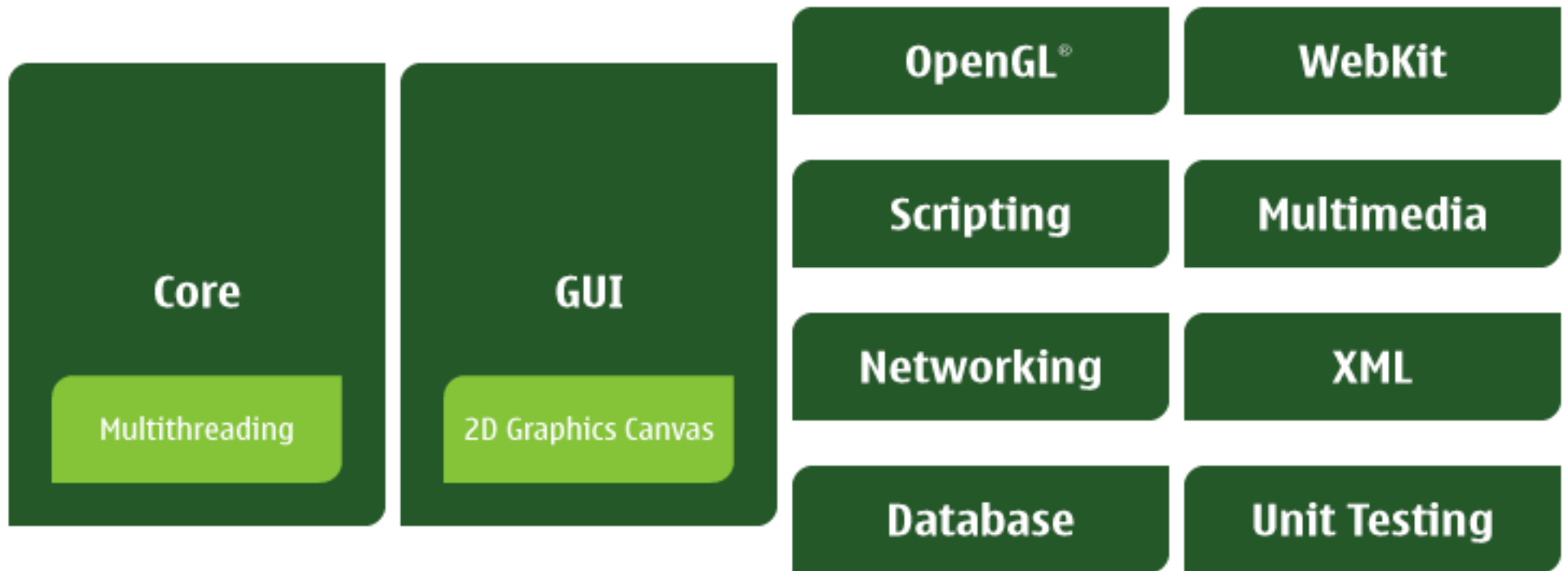
Einleitung

Was ist Qt?

- plattformunabhängige GUI-Bibliothek
- Standardsprache ist C++
- Komponenten für SQL, XML, Multithreading, Netzwerk usw.
- Wird wie „cute“ ausgesprochen...



Einleitung





Einleitung

Geschichte:

- 1991 begannen *Haavard Nord* und *Eirik Chambe-Eng* in Norwegen mit den Arbeiten an Qt.
- 1994 wurde Quasar-Technologies gegründet und später in Trolltech umbenannt.
- Trolltech wurde 2008 von Nokia aufgekauft.



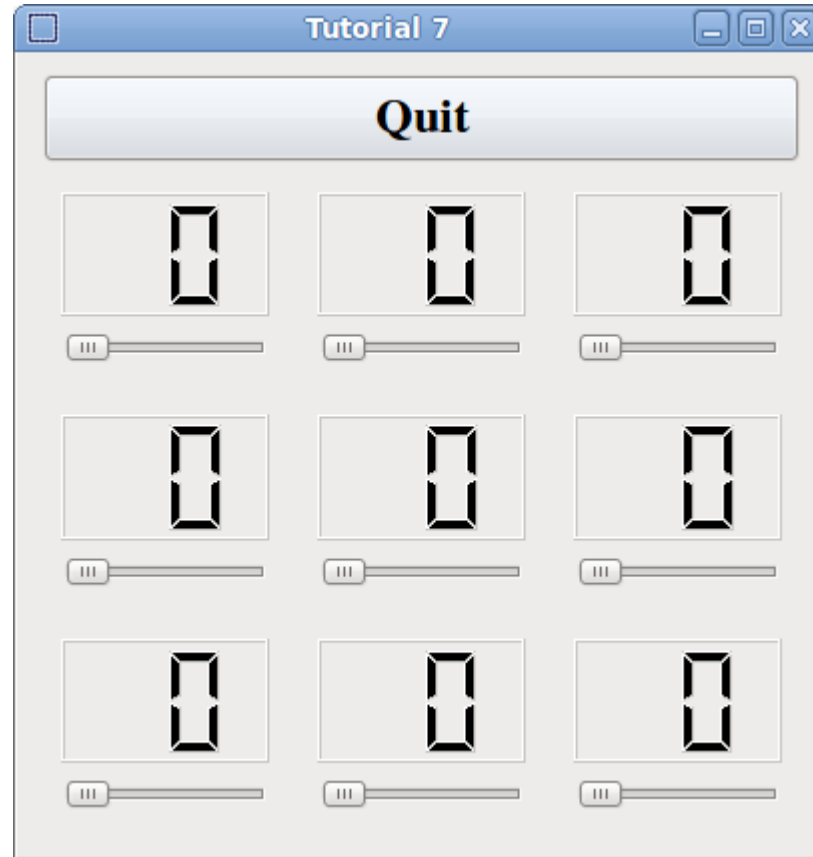
Einleitung

Lizenz:

- Qt stand anfangs unter einer proprietären Lizenz.
- 1998 wird unter anderem von der „KDE Free Qt Foundation“ eine freie Lizenz gefordert.
- 2000 wird ein duales Lizenzmodell für Linux eingeführt. (Kommerziell / GPL)
- 3. März 2009: Qt 4.5 wird zusätzlich unter der LGPL veröffentlicht.



Beispiel





Die Klasse QWidget

- Basisklasse für alle GUI Objekte.
- Wenn ein Widget kein Parent hat, wird es selbst zu einem Fenster.
- Wird oft zur Erstellung von „Composite-Widgets“ verwendet.
- QWidget ist eine Unterklasse von [QPaintDevice](#) und wird durch eine Instanz von [QPainter](#) gezeichnet.
- Alle Zeichenoperationen werden in der Methode [paintEvent\(\)](#) ausgeführt.



Die Klasse QWidget

- Die Größe wird durch die Methode `resize(int, int)` verändert.

QWidget verfügt über einige Events, die nach Bedarf (re)implementiert werden können:

- `paintEvent()`
- `resizeEvent()`
- `mousePressedEvent()`
- `keyPressedEvent()`
- `closeEvent()`
- ...



Events und Event Filter



Events und Event Filter

- Event Objekte werden von der abstrakten **QEvent** Klasse abgeleitet.
- Events können von jeder **QObject** Unterklasse verarbeitet werden.



Events und Event Filter

Übertragung von Events

Wenn ein Event auftritt wird von Qt ein Event Objekt der entsprechenden `QEvent` Unterklasse übergeben, indem die `event()` Funktion (Rückgabewert: boolean) aufgerufen wird.

Event Typen

Die meisten Events haben spezielle Klassen:

`QResizeEvent`, `QPaintEvent`, `QMouseEvent`,
`QKeyEvent`, `QCloseEvent`, usw...



Events und Event Filter

Event Handler

- Normalerweise wird eine virtuelle Funktion aufgerufen. In dieser Funktion beschreibt man das gewünschte Verhalten. Wenn man nicht alle Fälle abdecken will, ruft man die Implementierung der Basisklasse auf:



Events und Event Filter

Beispiel:

```
void MyCheckBox::mousePressEvent(QMouseEvent *event)
{
    if (event->button() == Qt::LeftButton) {
        // handle left mouse button here
    } else {
        // pass on other buttons to base class
        QCheckBox::mousePressEvent(event);
    }
}
```



Events und Event Filter

- Wenn die Funktion zur Behandlung eines bestimmten Events fehlt, muss `QObject::event()` überschrieben werden, indem die „Spezialbehandlung“ vor oder nach der gewöhnlichen Eventverarbeitung geschieht:



Events und Event Filter

Beispiel:

```
bool MyWidget::event(QEvent *event)
{
    if (event->type() == QEvent::KeyPress) {
        QKeyEvent *ke = static_cast<QKeyEvent *>(event);
        if (ke->key() == Qt::Key_Tab) {
            // special tab handling here
            return true;
        }
    } else if (event->type() == MyCustomEventType) {
        MyCustomEvent *myEvent = static_cast<MyCustomEvent *>(event);
        // custom event handling here
        return true;
    }

    return QWidget::event(event);    //“normale“ Events
}
```



Meta Object System



Meta Object System

- `QObject` ist die Basisklasse für alle Klassen, die die Vorteile nutzen möchten.
- Das `Q_OBJECT` Makro im privaten Bereich einer Klassendeklaration ermöglicht den Nutzen von Funktionen wie Signals & Slots, dynamischen Properties und Laufzeit-Typinformationen.



Meta Object System

- Der Meta-Object Compiler produziert zusätzlichen Code, wenn er in einer Klassendeklaration auf das `Q_OBJECT` Makro stößt.
- Dieser Code wird dann in kompilierter Form mit der Objektdatei der Klasse verlinkt.



Meta Object System

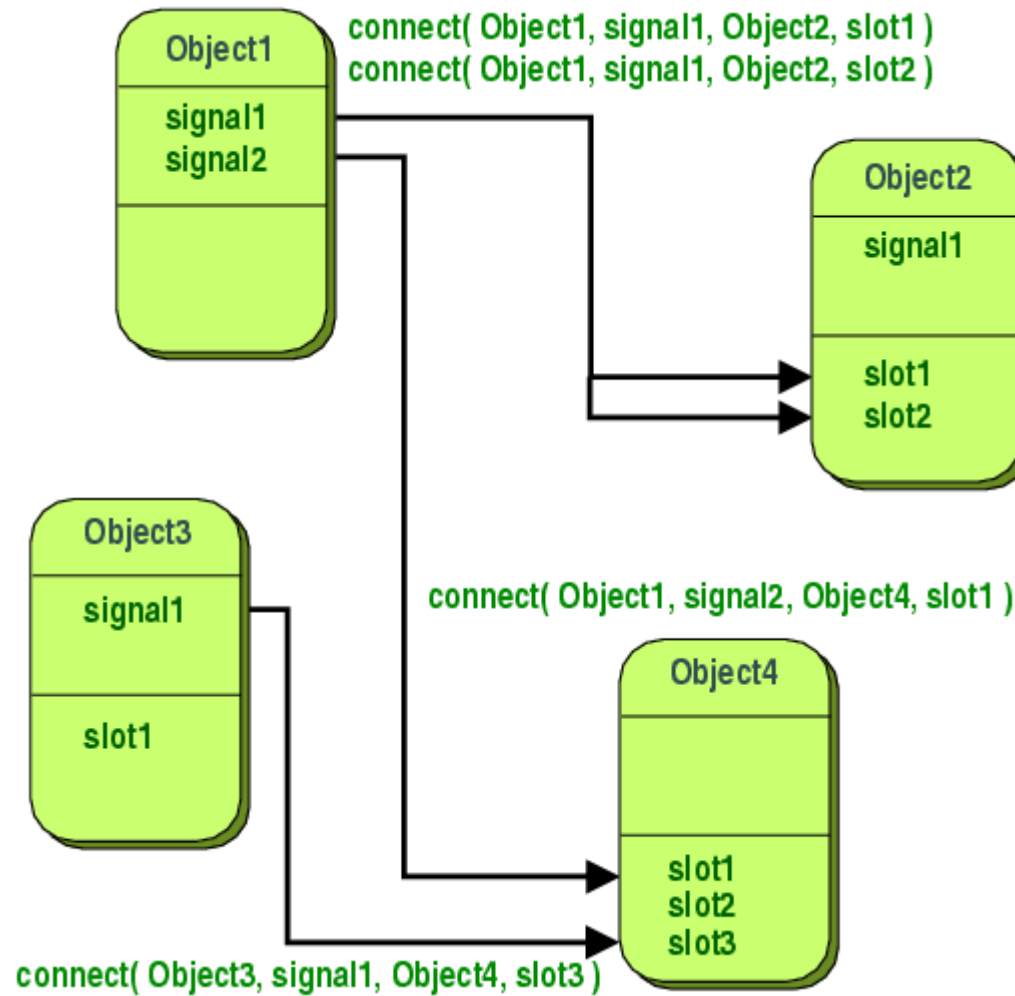
- Ist das nicht ein großer Aufwand mit diesem Meta-Object-Kompiler?
- qmake → make
- Präprozessor → MOC → C++ Kompiler



Signale und Slots



Signale und Slots





Signale und Slots

Das Makro `Q_OBJECT` muss in den privaten Bereich der Klassendefinition eingetragen werden:

```
class LCDRange : public QWidget
{
    Q_OBJECT
    ...
}
```



Signale und Slots

Signals und Slots deklarieren:

public slots:

```
void setValue(int value);
```

signals:

```
void valueChanged(int newValue);
```



Signals and Slots

Verbinden eines Signals mit einem Slot:

```
bool connect ( const QObject * sender, const char * signal,  
              const QObject * receiver, const char * method)
```

Hinweis: `connect(...)` ist eine statische Memberfunktion von `QObject`.



Signale und Slots

Aussenden eines Signals:

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



Threading



Threading

Threads starten mit QThread:

Mit `QThread::start()` wird ein neuer Thread gestartet. Die `QThread::run()` Methode muss entsprechend überschrieben werden.

Wie auch bei GUI-Komponenten muss eine Instanz von `QApplication` bzw. `QCoreApplication` zuvor gestartet worden sein.



Threading

Beispiel:

```
class MyThread : public QThread
{
    Q_OBJECT

protected:
    void run();
};

void MyThread::run()
{
    ...
}
```



Threading

Synchronisationsmöglichkeiten:

- QMutex
- QReadWriteLock
- QSemaphore
- QWaitCondition



Threading

QMutex:

- Es kann immer nur ein Thread auf den von der Mutex geschützten Bereich zugreifen.
- Ähnlich dem `synchronised` Keyword in Java
- Wenn ein anderer Thread auf eine bereits verwendete Mutex zugreift, wird er schlafen gelegt, bis die Mutex wieder freigeschaltet wird.



Threading

Beispiel zu QMutex:

```
QMutex mutex;  
int number = 6;
```

```
void method1()  
{  
    mutex.lock();  
    number *= 5;  
    number /= 4;  
    mutex.unlock();  
}
```



Threading

Beispiel zu QReadWriteLock:

```
QReadWriteLock lock;  
  
void ReaderThread::run()  
{  
    ...  
    lock.lockForRead();  
    read_file();  
    lock.unlock();  
    ...  
}  
  
void WriterThread::run()  
{  
    ...  
    lock.lockForWrite();  
    write_file();  
    lock.unlock();  
    ...  
}
```



Beispiel

ACGTATCGATCGTCAGCTGCATCGATCGTCAG
CTGCATCGACGTCAGCTGCATCGATCGTCAGC
TGCATCTACGCTGACCACAGCTGCATCGATCG
TCAGCTGCATCTCGTTCGATCGTCAGCTGCAT
CGATCGTCATGCATCGATCGTCAGCTGCATCG
ATCGTCAGCTGCATCGATCGTCAGCTGCATCG
ATCGTCAGCTGCATCGATCGTCAGATCGATCG
TCAGCTGCATCGATCGTCAGCTGCCTGCATCG
ATCGTCAGCTGCATCGATCGTCGCCAGGTTAC



ENDE