

Eine Kurz-Einführung in das Qt-Rahmenwerk



Christian Rahn

AB Kognitive Systeme
Fachbereich Informatik
Universität Hamburg

KOGS Oberseminar - 26.Mai 2004

Was ist das Qt-Toolkit?

Qt (www.trolltech.com) **ist hauptsächlich ein objekt-orientiertes GUI-Toolkit für C++:**

- Fenster
- Knöpfe
- Menüs
- ...

Darüberhinaus bietet es viele Utility Klassen an:

- Ein- / Ausgabe
- XML
- Lokalisiertes String-Handling (Unicode)
- Zugriff auf SQL-Datenbanken
- ...

Das Qt-Framework umfasst ca. 420 Klassen.

Warum Qt benutzen?

Alternativen: MFC, .net, Gtk, Motif, Swing, WxWindows, etc.

Qt läuft auf sehr vielen Plattformen:

- Unix (sehr viele)
- Mac OS X
- Windows (ab 95)

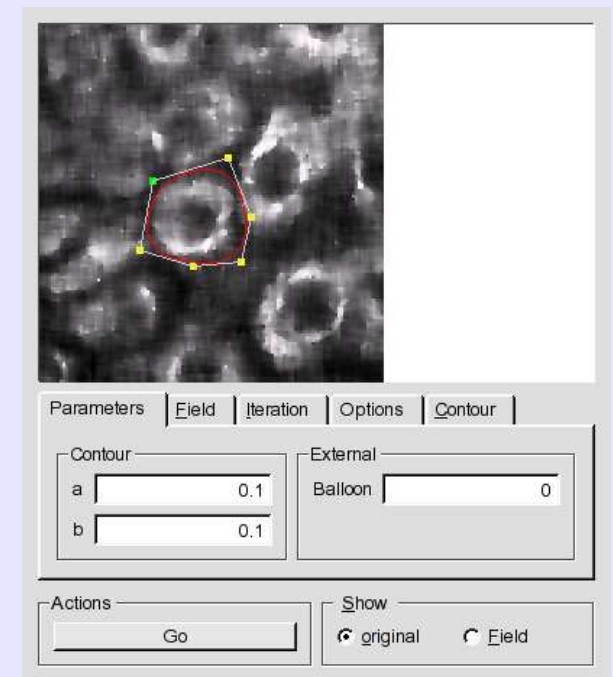


Qt läuft zB für Qtopia auch auf Embedded Systemen (mit Linux und ohne X11), wie z.B.

- PDAs
- Mobile Phones
- Tablet PCs
- ...

Warum Qt für BV benutzen?

- In der digitalen Bildverarbeitung ist oft die Spezifikation von **Parametern** gefragt:
 - Je nach Anwendung kann eine **interaktive Auswahl** oder **Variation** sehr hilfreich sein
 - Aber: **Script-Fähigkeit** (CLI) sollte stets eingeplant werden!
- **Visualisierung** von (Zwischen-) Ergebnissen
- Bereitstellung eines Verfahrens für **Anwender**



Die Vorteile von Qt

- exzellente Online-**Dokumentation** inkl. Beispiele, Tutorials und HowTos
- sehr kurze **Einarbeitungszeit**
- unterstützt schnelle Entwicklung
- stark **objekt-orientiert**, gut konzeptioniert
- sehr flexibel und sehr gut **erweiterbar** (Vererbung)
- das Arbeiten mit Qt macht **Spaß**
- unter Linux und Mac OS steht Qt für freie Anwendungen unter der QPL, für Windows gibt es eine non-commercial Version
- Qt wird kommerziell betreut
-> konstante Weiterentwicklung, Qualitätsmanagement

Die Nachteile von Qt

- Qt wird kommerziell betreut, resultierende Applikationen können nicht beliebig verwendet werden
- Qt muss installiert bzw. weitergegeben werden
- Qt ist in der allg. Software-Welt noch wenig verbreitet (Trend zunehmend)
- Qt ist sehr auf C++ zugeschnitten
- Qt nutzt moderne C++-Features nur in begrenztem Maße (STL, Namespaces, ...)
- Keine echte Model / View Trennung
 - soll mit Qt 4 eingeführt werden

Aufbau vieler Qt-Programme

Qt-Programme sind meist aufgebaut wie folgt:

```
#include <anything>

int main( int argn, char **argv )
{
    QApplication app( argn, argv );

    //-- Initialisierung, Widgets aufbauen

    int rc = app.exec();

    //-- Clean up

    return rc;
}
```

Konkret: Hello World!

```
#include <qlabel.h>
#include <qapplication.h>

int main( int argn, char **argv )
{
    QApplication app( argn, argv );

    QLabel *win = new QLabel( 0 );
    win->setText( „Hello World!“ );
    win->show();

    app.setMainWidget( win );

    return app.exec();
}
```


Kompilierung von Qt-Programmen

- **Qt benötigt viele Header-Dateien**
 - -> `include path` muss gesetzt werden
- **Benötigt Laufzeitumgebung**
 - -> `libqt.so` / `qt.dll`, `qt.lib` müssen eingebunden werden
- **Eventuell muss der moc-Präprozessor verwendet werden**
 - *moc*: Meta Object Compiler
- **Für GUI-Designer muss uic gestartet werden**
 - *uic*: User Interface Compiler
- **Forms, Icons, Translations einbinden**

Läßt sich alles mittels `qmake`-Tool verbergen!

-> ~System-unabhängige Konfiguration möglich

Das qmake-Tool

qmake liest eine (OS-unabhängige) **.pro* Datei und erstellt daraus ein *Makefile* oder **.vcproj*

```
CONFIG          += qt warn_on release opengl
TEMPLATE        = app
INCLUDEPATH     += include/

HEADERS         += model.h view.h
SOURCES         += main.cpp model.cpp view.cpp
FORMS           += settings.ui
TARGET          = qtvview

unix {
    QMAKE_CXXFLAGS += -O4
}
win32:SOURCES += win_debug.cpp
```



QApplication


Die Klasse QApplication erfüllt wichtige Aufgaben:

- Verwaltung von Event processing (mouse, keyboard, timer, ...)
- Globale Einstellungen (look'n'feel, Pfade, ...)
- I18N
- Session Management
- Kommunikation mit OS
- quit(), ...
- setMainWidget() lässt Programm automatisch bei Schließen des Fensters beenden

Ein QApplication-Objekt ist zwingend notwendig!
Seine Instanziierung initialisiert viele Qt-Features

RTFM

Read the *Fine* Manual

Home | All Classes | Main Classes | Annotated | Grouped Classes | Functions 

Qt Reference Documentation (Free Edition)

Notice: This edition is for the development of [Free and Open Source](#) software only; see [Commercial Editions](#).

Qt Community	Getting Started	General
<ul style="list-style-type: none"> Mailing lists <i>Qt Quarterly</i> newsletter User contributed Qt additions How to report a bug 	<ul style="list-style-type: none"> How to Learn Qt Tutorial #1, Tutorial #2 Examples Whitepaper 	<ul style="list-style-type: none"> About Qt Commercial Editions Free Edition About Trolltech
All Classes <ul style="list-style-type: none"> All Classes Main Classes Grouped Classes Annotated Classes Inheritance Hierarchy Class Chart (clickable image) All Functions (long) Header File Index FAQs Change History 	<ul style="list-style-type: none"> Network OpenGL SQL Table Workspace XML 	<ul style="list-style-type: none"> Window Geometry Events and Event Filters Internationalization (i18n) Debugging Techniques Thread Support in Qt Qt Plugins Standard Accelerators
Porting & Platforms	Tools	Licenses & Credits
<ul style="list-style-type: none"> Window system specific notes ActiveQt Framework Motif Extension Mac OS X development Porting from Qt 2.x to Qt 3.x 	<ul style="list-style-type: none"> All Tools Qt Designer Qt Linguist Qt Assistant qmake 	<ul style="list-style-type: none"> Q Public License GNU General Public License Third Party Licenses used in Qt Other Licenses used in Qt Credits

Copyright © 2004 Trolltech Trademarks Qt 3.3.1

Mail an:
qt-interest@trolltech.com

QLabel

Hello
World!

QLabel kann (RTF-)Text oder Grafiken anzeigen:

```
QLabel *label = new QLabel( win, „label1“ );

label->setText( „<P>Hello World!</P>“ );
label->setNum( 3.1415 );

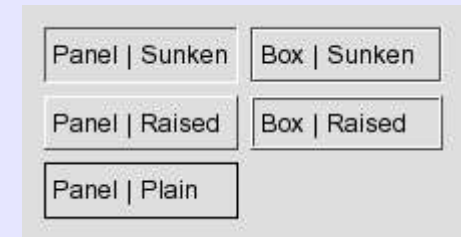
QPixmap pm( „image.png“ );
label->setPixmap( pm );

QMovie mov( „animation.mng“ );
label->setMovie( mov );
```

**Qt-Klassen bieten i.A. sehr viele Methoden,
dennoch wenig Redundanz**

QFrame

QLabel ist abgeleitet von QFrame.



QFrame zeichnet Rahmen:

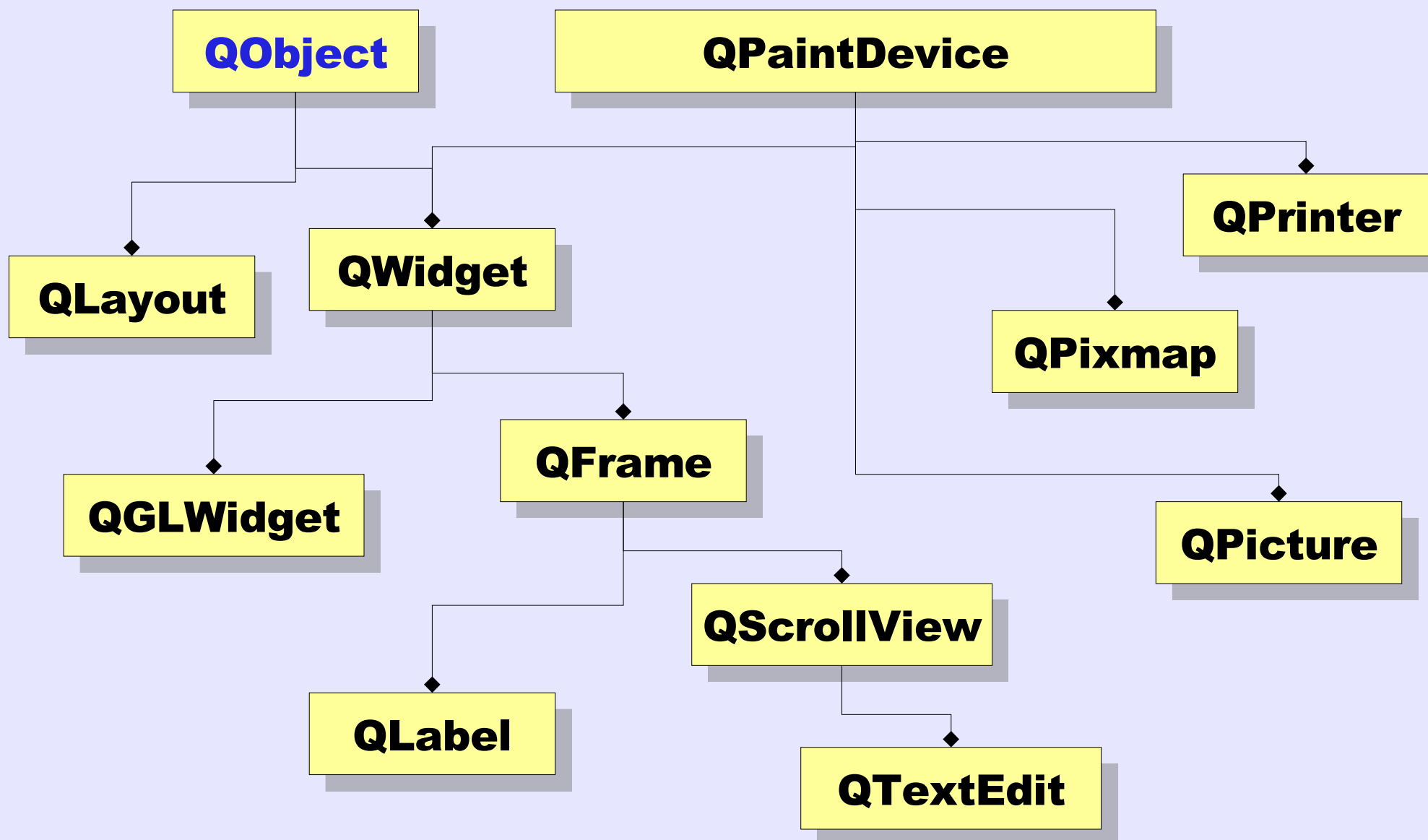
- `setFrameStyle(QFrame::Panel | QFrame::Raised);`
- `setFrameWidth(2);`

**Sehr viele Widgets sind von QFrame abgeleitet,
zB QMenuBar, QLabel, QLineEdit, ...**

QFrame ist von QWidget abgeleitet, ...

**..., QWidget ist ein QObject und QPaintDevice
(Mehrfachvererbung) ...**

Klassen-Verwandschaft



Die QObject-Klasse (1)

- QObject ist Basis vieler Klassen
- implementiert Signal-Slot-Mechanismus
- erlaubt (runtime-) Introspektion von Objekten
- errichtet Objekt-Hierarchie, löscht children automatisch

```
QLabel *label = new QLabel( mainWindow );

if( widget->isA( „QLabel“ )) { delete widget; }

label->setName( „scaleLabel“ );

int timer = label->startTimer( 50 );

emit valueChanged( 500 );

connect( speedInput, SIGNAL(valueChanged(int)),
        label, SLOT(setNum(int)) );
```


Die QObject-Klasse (2)

Signal / Slot-Mechanismus, Introspektion und Properties sind durch eine Erweiterung der C++ Sprache realisiert:

- Der Präprozessor **moc** erweitert mit **Q_OBJECT** markierte Klassen implizit um wenige spezielle Methoden, Macros
- -> es fällt eine weitere **moc_*.cpp** Datei je QObject-Klasse an, die eingebunden werden muss (qmake)
- **Properties** können mittels **metaObject()** zur Laufzeit abgefragt werden
- **className()**, **children()**, **deleteLater()**
- QObject's senden z.B. *destroyed*-Signal aus
- **Ein Objekt sendet Signal, viele Objekte können es empfangen**

Die QObject-Klasse (3)

```
class MouseListener : public QObject
{
    Q_OBJECT
    Q_PROPERTY( int speed READ speed WRITE setSpeed )

public:
    MouseListener( QObject *parent = 0,
                  const char *name = 0 );
    virtual ~MouseListener();
    int speed( void ) const;

public slots:
    void setSpeed( int );
    void visit( int x, int y );
signals:
    void selected( int x, int y );
private:
    int _speed;
};
```

Die QWidget-Klasse (1)

QWidget ist Basis für (fast) alles auf dem Screen

- Ein Widget **A** kann Fenster (TopLevel-Widget) werden oder Teil eines anderen Widgets **B**
- Ist **A** ein sub-Widget von **B**, so wird **A** **automatisch** gemeinsam mit **B** **vernichtet** (Memory Management)
- Es können (und sollen!) neue Widget-Klassen von bestehenden abgeleitet werden
- Ein Widget stellt **Event-Handlers** zur Verfügung; diese können überschrieben werden (mousePress, paint, ...)
- Ein Widget ist ein rechteckiger Bildausschnitt, der
 - sich selbst zeichnen kann
 - Eingaben empfangen kann
- `void setCaption(const QString &title);`
- `void setIcon(const QPixmap &pm);`

Die QWidget-Klasse (2)

```
#include <qlabel.h>
#include <qapplication.h>

int main( int argn, char **argv )
{
    QApplication app( argn, argv );
    QLabel *win = new QLabel( 0 );

    win->setText( "<p>Hello World!</p>" );

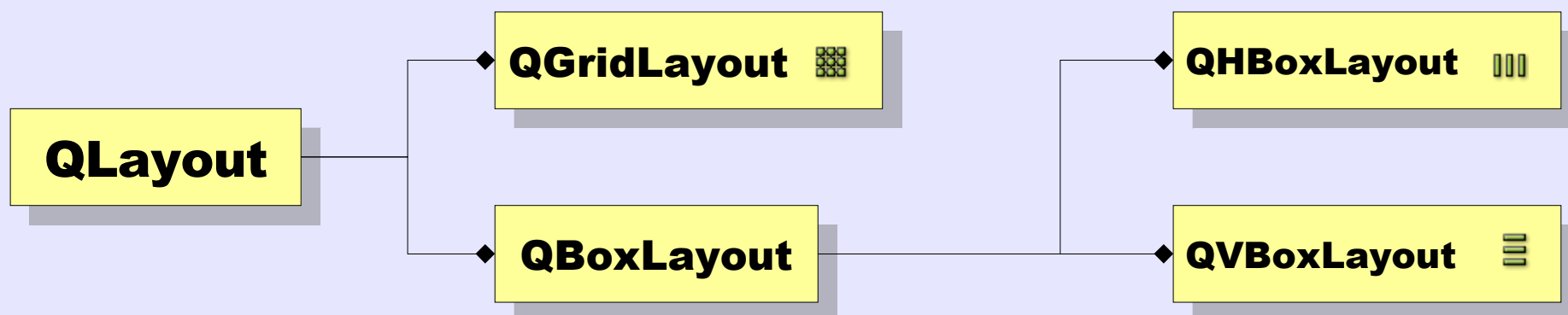
    QLabel *label = new QLabel( win );
    label->setText( "<p>Hallo Welt!</p>" );
    label->setBackgroundColor( Qt::red );
    // label->setGeometry( 0,0, 100,20 );

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



Das Layout-System (1)

- **Layouts setzen die Geometrie von Widgets automatisch**
- **reagieren dynamisch bei Fenster-Vergrößerung**
- **können geschachtelt werden**
 - jedes Widget liefert `sizeHint()`
 - hat `minimumSize()` und `maximumSize()`
 - kennt `setFixedSize(const QSize &size)`



Das Layout-System (2)

```
#include <qlabel.h>
#include <qapplication.h>
#include <qlayout.h>

int main( int argn, char **argv )
{
    QApplication app( argn, argv );
    QWidget *win = new QWidget( 0 );
    QVBoxLayout *lo = new QVBoxLayout( win );

    QLabel *l1 = new QLabel( „Hello 1“, win );
    lo->add( l1 );

    lo->addSpacing( 5 );

    QLabel *l2 = new QLabel( „Hello 2“, win );
    lo->add( l2 );

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



Das Layout-System (3)

- **Durch das Layout-System können praktisch alle Dialoge und Fenster dynamisch vergrößerbar sein**
- **es müssen keine festen Koordinaten mehr angegeben werden**
- **wichtig z.B. für**
 - unterschiedliche Screen-Auflösungen
 - unterschiedliche default-Fonts vom OS
 - bei I18N
- **Noch einfacher verwendbar:**
 - die Widgets QGrid, QVBoxLayout, QHBoxLayout, ...

Signal-Beispiel

```
#include <qapplication.h>
#include <qlayout.h>
#include <qslider.h>
#include <qlcdnumber.h>

int main( int argn, char **argv )
{
    QApplication app( argn, argv );
    QWidget *win = new QWidget( 0 );
    QVBoxLayout *lo = new QVBoxLayout( win );

    QLCDNumber *lcd = new QLCDNumber( win );
    lo->add( lcd );

    QSlider *slider = new QSlider( win );
    slider->setOrientation( Qt::Horizontal );
    lo->add( slider );
    QObject::connect( slider, SIGNAL(valueChanged(int)),
                      lcd, SLOT(display(int)) );

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



Event-Handler (1)

```
#include <qwidget.h>

class CoolView : public QWidget
{
    Q_OBJECT

public:
    CoolView( QWidget *parent = 0, const char *name = 0 );
    virtual ~CoolView();
    void setData( const CoolData *data );

public slots:
    void updateView();

signals:
    void viewChanged();
    void visitPosition( int x, int y );

protected:
    virtual void paintEvent( QPaintEvent *e );
    virtual void keyPressEvent( QKeyEvent *e );
    virtual void mousePressEvent( QMouseEvent *e );
    void paintTo( QPainter *painter );
};
```

Event-Handler (2)

```
void CoolView::paintEvent( QPaintEvent *e )
{
    QPainter painter( this );
    paintTo( &painter
}
```

```
void CoolView::paintTo
{
    painter->setPen( Q
    painter->drawEllip
}
```

```
void CoolView::keyPressEvent( QKeyEvent *e )
{
    switch( e->key() ) {
    case Qt::Key_Escape:
        qApp->quit();
        break;
    default:
        e->ignore();
        return;
    }
```

```
void CoolView::mousePressEvent( QMouseEvent *e )
{
    if( e->button() & Qt::LeftButton ) {
        e->accept();
        emit visitPosition( e->x(), e->y() );
    } else {
        e->ignore();
    }
}
```

Zeichnen mit QPainter

QPainter(QPainterDevice *dev)

- drawLine(QPoint, QPoint), drawRect(QPoint, QSize), setPen(QColor), ...
- setViewport(), setWindow()
- translate(), rotate(), scale(), shear()
- setWorldMatrix(const QWMMatrix & m)
- drawImage(), drawPixmap()

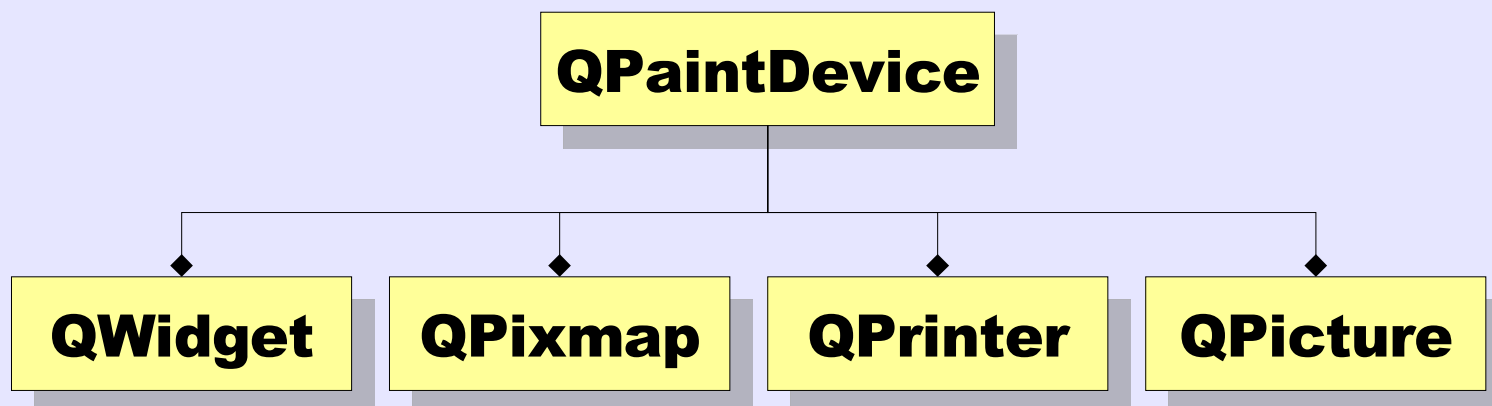


Image Processing

Es gibt zwei Klassen für Bilddaten in Qt:

QPixmap:

- optimiert für Darstellung
- kann mit QPainter verwendet werden

QImage:

- direkter Zugriff auf Pixel möglich (qRgb-Macro!)
 - setPixel(), pixelIndex()
- alpha-Buffer
- Hardware-unabhängig

Beide Klassen können diverse **Bildformate lesen und schreiben, **Plugins** für neue Formate möglich**

Weitere Widget-Klassen (1)

QLineEdit



- Eingabe von Text
- Eingabe von double-Werten mit [QDoubleValidator](#) möglich
- **Signals:**
 - textChanged(const QString &)
 - returnPressed()
 - selectionChanged()
- **Slots:**
 - setText(const QString &text)
 - clear()
 - cut(), copy(), paste()
 - setSelection(int start, int length)
- **Properties:**
 - text

Weitere Widget-Klassen (2)

QCheckBox

- ja / nein
- **Signals:**
 - clicked()
 - toggled(bool)
 - stateChanged(int)
- **Slots:**
 - setChecked(bool)
 - toggle(void)
- **Properties**
 - checked
 - tristate



QRadioButton

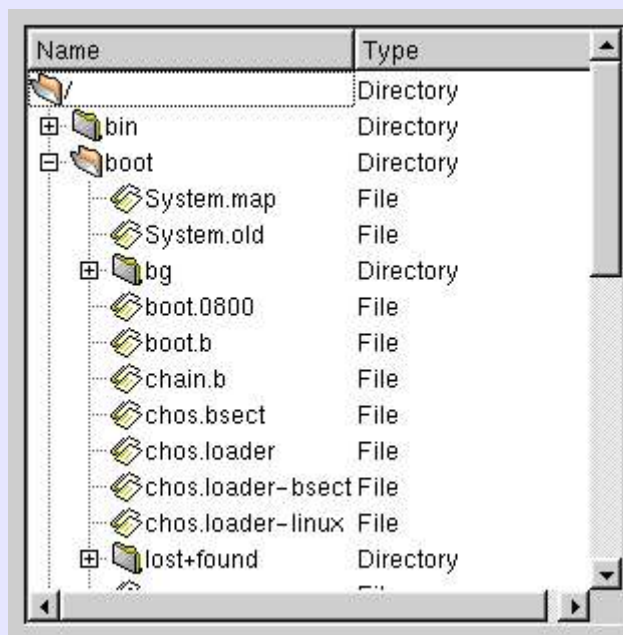
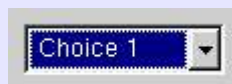
- 1-aus-n
- **Signals:**
 - clicked()
 - toggled(bool)
- **Slots:**
 - setChecked(bool)
- **Properties**
 - checked



**Zu verwenden mit einer
QButtonGroup**

Weitere Widget-Klassen (3)

- **QPushButton**
- **QComboBox**
- **QListView**



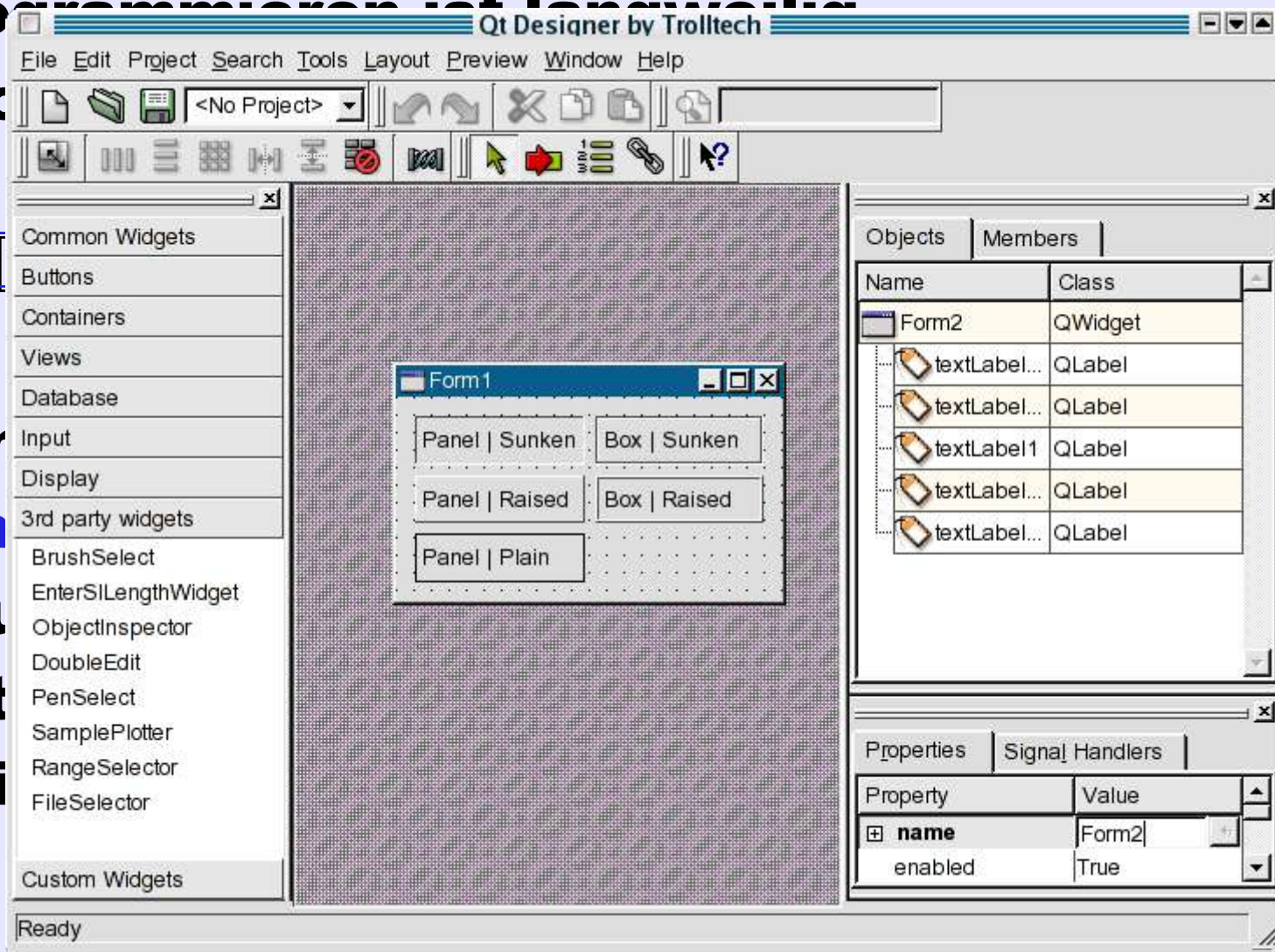
- **QTextBrowser**
- ...

Der Qt-Designer (1)

- GUIs programmieren ist langweilig
- GUI-Layout ist kompliziert

-> Qt

- lässt korrekt GUIs programmieren
- Funktionale GUIs
- eingebauter Compiler
- kann mit C++ programmiert werden
- GUI-Designer
- werden



Der Qt-Designer (2)

1. Es wird ein **Dialog**, **Widget** oder **MainWidget** mit dem **Designer** erstellt, ein sog. Form: **MyForm.ui**
2. Der **uic** erzeugt daraus **C++ Dateien**: **MyForm.cpp** und **MyForm.h**, etwaige **Icons** werden berücksichtigt
3. **moc** kümmert sich um alles und baut es zusammen
4. Es steht nun eine **Klasse MyForm** zur Verfügung

Der Qt-Designer (3)

Wie kommt die Funktionalität in die GUI?

- 1) Im **Text-Editor** des Designers können **Funktionen** und **Slots** direkt bearbeitet werden (Datei: MyDialog.ui.h);
für einfache Dialoge
- 2) Im Designer wird nur die GUI als **MyDialogBase** erstellt; dann via **MyDialog : public MyDialogBase** die Funktionalität abgeleitet, Funktionen überschrieben;
für komplexe Dialoge
- /* 3) Nur Layout im Designer (like 1) erstellen, Funktionalität durch aufrufende Klasse; stark eingeschränkt, fehleranfällig, kein OO */

Komplexe GUI-Applikationen

- **Toolbars, Statusbar, Menüs, Kontextmenüs, Key-Shortcuts, etc.**
 - die Programm-Funktionen sind oft redundant aufrufbar
 - können ev. via Script-Sprache extern gesteuert werden
- **QAction** abstrahiert die Aktivierung einer Programm-Funktion von der Aufruf-Modalität
- **QAction** sendet **activate**-Signal
- **QActions** können gruppiert werden (zB links-/rechtsbündig/Blocksatz)

Ein- / Ausgabe mit Qt

- **QFile file(const QString &filename)**
 - open, read, close, write
 - kann transparent CR/LF konvertieren: Unix <-> M\$
- **QTextStream stream(QIODevice *dev)**
 - Ein-/ Ausgabe von Text
 - a la std::cout, cin, cerr
 - stream << „Hello world!“ << endl;
 - benutzt Unicode sowie diverse Zeichensätze
- **QDataStream stream(QIODevice *dev)**
 - serialisiert (atomare) Binärdaten
 - Plattform-unabhängig (big / little endianess)
- **QClipboard, QMimeSource, QSettings, QSocket, XML, ...**

Allgemeines

Viele Qt-Klassen benutzen Shared data

- Call by value ist effizient
 - QString, QPixmap, ...
- `QString QWidget::caption () const`

Jedoch ist call by reference meist noch effizienter:

- `void QWidget::setCaption(const QString &)`

Es wird Gebrauch von Pointern für Individual-objekte gemacht:

- `void QApplication::setMainWidget (QWidget * mainWidget)`

Diese Konventionen sollten für eigene Slots, etc. eingehalten werden!

Ausblick

Was wurde hier ausgelassen?

- QRegExp: Reguläre Ausdrücke
- Netzwerk-Fähigkeiten
- XML: DOM und SAX
- Datenbank-Zugriff via SQL
- OpenGL Interface
- Netscape Plugins
- Internationalization
- Nur für Spiele? Canvas-Graphik
- Multi-Threading
- Qt Scripting Architecture: QSA
- /* Active X */
- Plugins erstellen
- ...

Das Ziel

**Viel Spaß
mit Qt!**

