Einführung in die Programmierung mit Qt

Dr. Ralf Schüler

5.11.2009

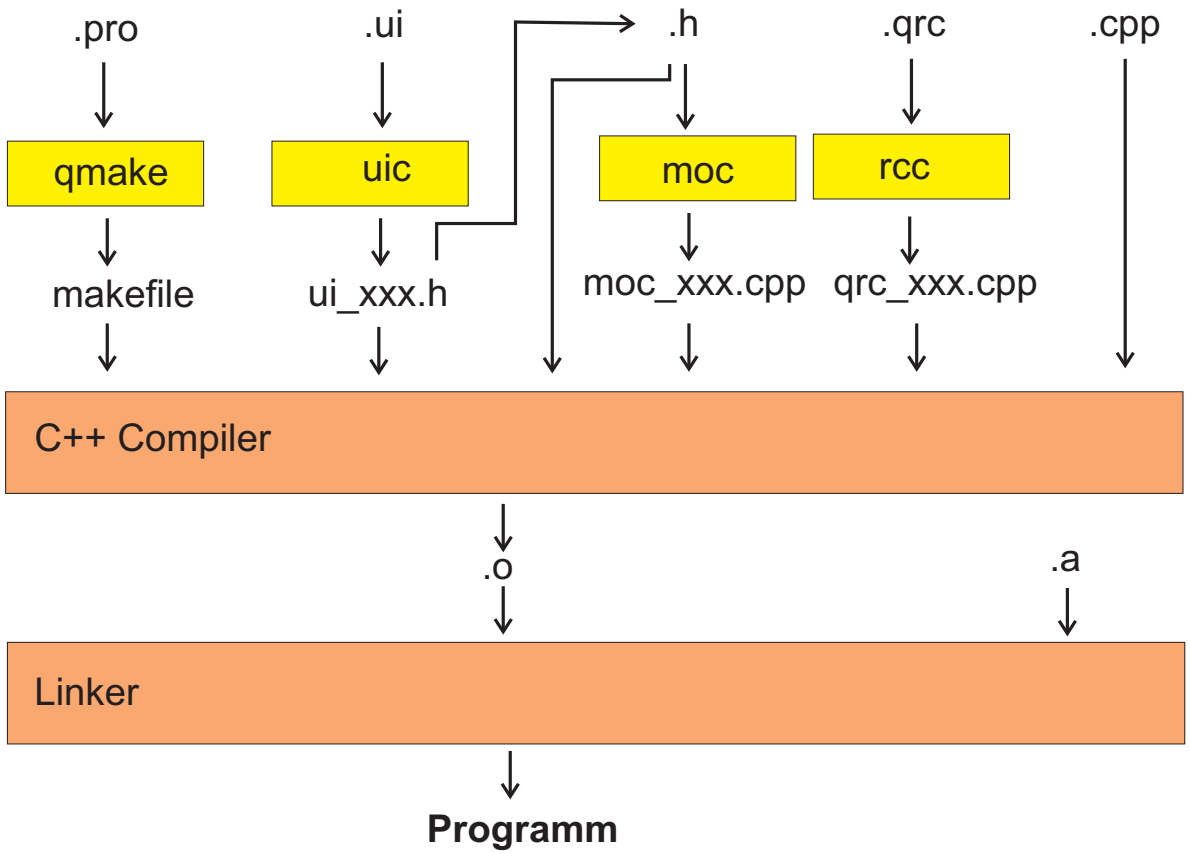
Was ist Qt?

- sprich: [kju:t]
- Klassenbibliothek für C++ (ferner Java, Python u.a.)
- sehr umfangreich: GUI, Stringverarbeitung, reguläre Ausdrücke, Netzwerk, Multithreading, XML ...
- Multi-Plattform: Windows, Linux (X11), Linux embedded (framebuffer), Mac, WinCE, S60
- von Nokia (früher Trolltech) unter verschiedenen Lizenzen angeboten: GPL, LGPL, Commercial
- KDE-Desktop basiert auf Qt

Unser heutiges Ziel

- Kennenlernen des Prinzips hinter Qt
- Kennenlernen der wichtigsten Entwicklungswerkzeuge (QtCreator, Designer, Assistant)
- Erstellen eines einfachen Programms mit einigen GUI-Elementen und etwas Stringverarbeitung

Übersicht über die Tools



„Am Anfang steht das Makefile“

- *make* steuert automatisch die Tools
- *make* wird über ein makefile gesteuert
- Qt bietet einfache makefile-Syntax in eigenem Makefile-Format. (.pro)
- *qmake* erstellt aus .pro-File ein makefile für *make* (an den jeweiligen Compiler, Zielumgebung und Installationsbesonderheiten angepasst)

Das sehen wir später genauer.

IDEs für Qt

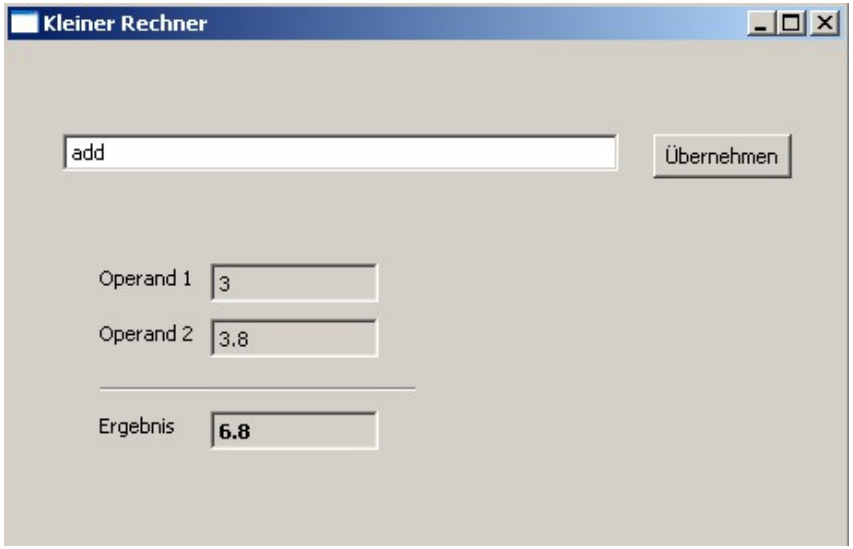
- Designer, Assistant (Helpsystem), Linguist

- jeder beliebige Editor mit Programmierunterstützung
- Eclipse (gibt ein Qt-Eclipse-Plugin)
- Visual-Studio (nur für Windows, bisher nur kommerzielle Lizenz)
- **QtCreator** (Qt eigene IDE)

Wegen Geschwindigkeit und bessere Integration der Tools nehmen wir heute mal den QtCreator.

Kleines Beispielprogramm

- kleiner Minitaschenrechner
- wird über eine Eingabezeile bedient
- Wir lernen die Basiskonzepte, etwas GUI und ein wenig Stringverarbeitung
- ... und sorry, Lernbeispiele sind meist etwas sinnlos :-)



Qt Makefile (fosslc1.pro)

```
QT          += network

;DEFINES -= UNICODE

TARGET = fosslc1
TEMPLATE = app


SOURCES += main.cpp\
           samplewidget.cpp

HEADERS  += samplewidget.h

FORMS    += samplewidget.ui
```

main.cpp

```
// globale Variablen an einem festen Ort deklarieren
// zum Beispiel hier in main.cpp

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    // Initialisierung

    //Anlegen Hauptfenster
    SampleWidget w;
    w.show();

    //Hauptschleife
    int ret=a.exec();
    // Aufräumen

    return ret;
}
```

samplewidget.h: Mehrfachableitung ist schöner

```
namespace Ui
{
    class SampleWidget;
}

class SampleWidget : public QWidget
{
    Q_OBJECT
public:
    SampleWidget(QWidget *parent = 0);
    ~SampleWidget();
private:
    Ui::SampleWidget *ui;
};
```

```
// die Includedatei vom Designer muss rein
#include "ui_samplewidget.h"

class SampleWidget : public QWidget,
                     public Ui_SampleWidget
{
    Q_OBJECT
protected:
int Parse(QString s);
public:
    SampleWidget(QWidget *parent = 0);
    ~SampleWidget();
public slots:
    void on_setButton_clicked();
    void on_lineEdit_returnPressed();
};
```

samplewidget.cpp: Auch hier ändern wir was am generiertem Code

```
SampleWidget::SampleWidget(QWidget *parent)
    : QWidget(parent),
      ui(new Ui::SampleWidget)
{
    ui->setupUi(this);
}

SampleWidget::~SampleWidget()
{
    delete ui;
}
```

```
SampleWidget::SampleWidget(QWidget *parent)
    : QWidget(parent)
{
    setupUi(this);
}

SampleWidget::~SampleWidget()
{
}

void SampleWidget::on_setButton_clicked()
{
    on_lineEdit_returnPressed();
}

void SampleWidget::on_lineEdit_returnPressed()
{
    int error=Parse(lineEdit->text());
}

int SampleWidget::Parse(QString s)
{
    //hier kommt nachher der Hauptcode rein
    return 0;
}
```

Qt Spracherweiterungen

- Qt erweitert C++ um einige Elemente und bedient sich dazu eigenem Preprocessor (moc)
- *Q_OBJECT* markiert ein Object für den moc
- Am wichtigsten: Signale und Slots
  - *signals:* und *public slots:* leiten in Objektdeklaration besondere Funktionen ein
  - *emit signalname();* sendet Signal und alle mit diesem Signal verbundene Slots (normale Funktionen die als Slot markiert wurden) werden automatisch aufgerufen
  - intern über Listen von Funktionen realisiert
  - wird ein Object zerstört, werden die Slotfunktionen automatisch aus der Signalliste entfernt
  - funktioniert auch über Threadgrenzen (mit Aufruf im richtigen Context)
  - es können auch Parameter übergeben werden

Signale und Slots verbinden

Hierfür gibt es mehrere Möglichkeiten:

1. Zuweisung im Designer
   geht nicht immer, Slot und Signal müssen im Designer bekannt sein
2. explizit über *connect*-Makro

   ```
   connect( lineEdit ,SIGNAL(returnPressed()),this ,SLOT(on_lineEdit_returnPressed ()));
   ```

   geht immer, aber nicht sehr bequem.
3. implizit über speziellen Slotnamen
   einfach Slotnamen entsprechend wählen und die Slots werden automatisch verbunden.

   ```
   on_<objektMitSignal>_<Signalname>();
   ```

   z.B. `void on_lineEdit_returnPressed ();`
   Das ist extrem komfortabel!!

Alles klar?

- Wir compilieren jetzt mal. (Strg-B)
- ... und schauen uns auch mal die entstandenen Dateien an

Jetzt müsste das Programmgerüst soweit klar sein und wir können anfangen, die eigentliche Funktionalität zu füllen.

Nächster Schritt: Fehlerausgabe

Zuerst die wichtigsten includes

```
#include <QtCore>
#include <QtGui>
```

dann die Fehlerausgabe (mal ganz einfach)

```
void SampleWidget::on_lineEdit_returnPressed()
{
    int error=Parse(lineEdit->text());

    //wenn Parse einen Fehler zurückgibt, dann sagen wir das dem Nutzer
   if(error) QMessageBox::warning(this,"Fehler",
    QString("Beim Auswerten der Eingabe ist Fehler %1 aufgetreten.").arg(error));
   lineEdit->setText("");
}

int SampleWidget::Parse(QString s)
{
    //zum Ausprobieren mal einen Fehler zurückgeben
    return 1;
}
```

Syntax der Eingabezeile

Wir parsen zunächst mal folgende Befehle:

`clear`	löscht alle Felder
`set op1 15.3`	setzt Operand 1 (hier auf 15.3)
`set op2 8`	setzt Operand 2
`add`	addiert Operanden
`mult`	mutlipliziert Operanden

Erweitern und ändern können wir das dann noch zu jeder Zeit.

Parse-Funktion 1

```
s=s.simplified();
// aufsplitten an den Leerzeichen (brauchen wir nur für "set")
QStringList list=s.split(' ',QString::SkipEmptyParts);

if(list.isEmpty()||list[0].isEmpty())return 10;

if(list[0].compare("CLEAR",Qt::CaseInsensitive)==0){
    op1Label->setText("");
    op2Label->setText("");
    resultLabel->setText("");
    return 0;
}else if(list[0].compare("SET",Qt::CaseInsensitive)==0){
    // wir brauchen noch 2 Argumente
    if(list.count()!=3)return 11;
    if(list[1].compare("OP1",Qt::CaseInsensitive)==0){
        op1Label->setText(list[2]);
        return 0;
    }else if(list[1].compare("OP2",Qt::CaseInsensitive)==0){
        op2Label->setText(list[2]);
        return 0;
    }else return 2;
```

Parse-Funktion 2

```
    }else if(list[0].compare("ADD",Qt::CaseInsensitive)==0){
        double op1,op2,result;
        bool ok=false;
        op1=op1Label->text().toDouble(&ok);
        if(!ok)return 3;
        op2=op2Label->text().toDouble(&ok);
        if(!ok)return 4;
        result=op1+op2;
        resultLabel->setText(QString("%1").arg(result));
        return 0;
    }else if(list[0].compare("MULT",Qt::CaseInsensitive)==0){
        double op1,op2,result;
        bool ok=false;
        op1=op1Label->text().toDouble(&ok);
        if(!ok)return 3;
        op2=op2Label->text().toDouble(&ok);
        if(!ok)return 4;
        result=op1*op2;
        resultLabel->setText(QString("%1").arg(result));
        return 0;
    }
    else return 1;
}
```

„So werden Sie geholfen“

- Mitgelieferte Online-Dokumentation und Beispiele (auch QtCreator „Willkommen“-Seite, „Schnelleinstieg“ und „Community“ Tabs.
- Internet:
  - Qt `http://qt.nokia.com` `http://labs.trolltech.com`
  - Qt Tutorial `http://www.digitalfanatics.org/projects/qt_tutorial/`
  - Qt Centre Community `http://www.qtcentre.org/`
  - Google ist immer dein Freund
- Bücher: `http://qt.nokia.com/developer/books`