Inhaltsverzeichnis

1. QT Grundlagen: Widgets

1.1. Ziele

☑ Erste Schritte in QT

- ☐ GUI-Anwendungen:Button, Label, LineEdit, ComboBox, Slider, RadioButton, Listbox, ...
- ☐ Layout,QTimer

☑ Quellen Cpp und Qt:

- ☐ http://zetcode.com/gui/qt5/ (SUPER)
- ☐ https://www.youtube.com/playlist?list=PLS1QulWo1RIZiBcTr5urECberTITj7gjA (SUPER)
- ☐ http://www.bogotobogo.com/Qt/Qt5_GridLayout.php (SUPER)
- ☐ http://doc.qt.io/qt-5/qtexamplesandtutorials.html

1.2. Vorkenntnisse

Wir arbeiten mit dem QT-Creator und wollen einfache Qt-GUI-Anwendungen erstellen. Dazu gibt es folgende Klassen:

- **QMainWindow** (bereits mit Menu, Toolbar, …) - wird in einem eigenen Arbeitsblatt behandelt
- **QWidget** … Basis-Klasse für das Erzeugen von einfachen GUI-Formularen
- **QDialog** … Basis-Klasse für das Erzeugen von typischen Dialogelementen

Wir wollen zunächst einfache Formulare/Fenster erzeugen und verwenden dazu die Oberklasse **QWidget**.

1. Explorer: Workspace erstellen: zb: schule\ws-qt

2. Qt-Creator starten
3. Neues Projekt: Name und Ordner wählen
4. **Qt-Gui-Anwendung** wählen und folg. Einstellungen vornehmen

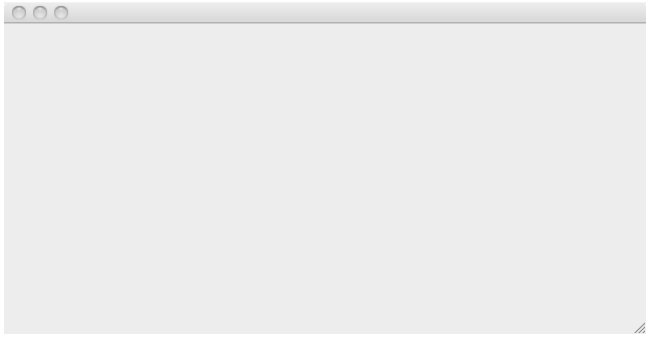
   Achtung: Die ersten Beispiele werden wir **ohne** Form-Designer erstellen.

Hinweis: Terminal Einstellungen für Mac-User

/usr/x11/bin/xterm -fa Monaco -fs 11 -sb -sl 1000 +lc -u8 -rightbar -e

1.3. Projekt: QWidget1-1: Ein leeres Fenster

Projekt: QWidget1



1.3.1. Das main-Programm

erzeugt ein Widget-Objekt, das von uns programmiert wird.

main.cpp

```
#include <QApplication>
#include "widget.h"
```

```
int main(int argc, char *argv[])
{
      /*
       * a ist ein Objekt der Klasse QApplication.
       * Das ist eine Art main-Objekt für Qt, in das
       * alle weiteren Qt GUI Komponenten eingebettet sind.
      */
    QApplication a(argc, argv);

      /*
       * hier kommt unser neues selbst-programmiertes
       * Widget (GUI-Element)
       */
    Widget w;
    w.show();


      /*
       * Die QApplication wird gestartet:
       */
    return a.exec();
}
```

Nun müssen wir noch unser eigenes Widget programmieren. Wir verwenden die Vererbung von der Klasse QWidget und sind so schnell fertig.

1.3.2. Die eigene Klasse: Widget

widget.h

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>

class Widget : public QWidget
{
    Q_OBJECT

public:
    Widget(QWidget *parent = 0);
    ~Widget();
};

#endif // WIDGET_H
```

widget.cpp

```
#include "widget.h"

// cons
Widget::Widget(QWidget *parent)
    : QWidget(parent)
{
}
```

```
//destr
Widget::~Widget()
{

}
```

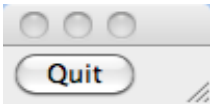
1.3.3. Aufgabe: QWidget1 - resize()

Erstellen Sie das Projekt und versuchen Sie per Programm, das Fenster auf eine Größe von 150 Breite und 50 Höhe zu bringen.

Tipp: in main.cpp
resize()

Verwenden Sie auch:
setGeometry()
setWindowTitle()

1.4. Projekt: QWidget1-2: Ein leeres Fenster mit QPushButton



Wir wollen nun einen Button hinzufügen und bei einem CLICK-Ereignis soll das Fenster geschlossen werden.

In Qt werden
- die Ereignisse **SIGNAL** genannt und
- die Ereignisprozeduren **SLOT** genannt.

widget.h

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QPushButton>

class Widget : public QWidget
{
    Q_OBJECT
private:
    QPushButton * qButtonQuit;

public:
    Widget(QWidget *parent = 0);
    ~Widget();
};

#endif // WIDGET_H
```

1.4.1. Aufgabe: Einen QPushButton erstellen

Im Konstruktor unserer Klasse Widget muss ein QPushButton-Objekt erzeugt werden und im Destruktor muss das Objekt wieder frei gegeben werden.

widget.h

```
private:
  QPushButton * qButtonQuit;
```

widget.cpp

```
im Konstruktor:
  qButtonQuit= new QPushButton("quit", this);

  ...
im Destruktor:
  delete qButtonQuit;
```

1.4.2. Aufgabe: QPushButton Signale

In Qt werden
- die Ereignisse **SIGNAL** genannt und
- die Ereignisprozeduren **SLOT** genannt.

Wir wollen nun auf ein CLICK-Ereignis bei Button reagieren können.

Frage:
Welche Signale erbt QPushButton von seiner Oberklasse QAbstractButton?

Hinweis:
gehe zu
QT-Creator → Hilfe → suche QPushButton → Reimplemented Protected Functions

Antwort: 4 Signals inherited from QAbstractButton

void	**clicked** (bool *checked* = false)
void	**pressed** ()
void	**released** ()
void	**toggled** (bool *checked*)

1.4.3. Aufgabe: QPushButton: Slot-Signal

Ein Ereignis (also SIGNAL) mit einer Ereignisprozedur (also SLOT) zu verbinden, geht man wie folgt vor:

Frage:
Wie lautet die connect-Anweisung um den Button btnQuit die Anwendung beenden zu lassen?

Antwort:

```
// Das Signal clicked() wird mit der Funktion/Slot close() verbunden

QObject::connect(qButtonQuit, SIGNAL(clicked()),   this, SLOT(close()));
                  SENDER ------------------------> EMPFÄNGER
```

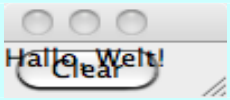
1.5. Projekt: QWidget1-3 Layouts verwenden

Wir wollen mehrere Elemente implementieren.

1.5.1. Aufgabe: QWidget1-3 – Add 1 QLabel und 3 QPushButtons

Aufgabe:
Fügen Sie in die Klasse Widget ein QLabel namens qLabelHello hinzu.
Fügen Sie in die Klasse Widget einen QPushButton namens qButtonHello hinzu.
Fügen Sie in die Klasse Widget einen QPushButton namens qButtonClear hinzu.
Fügen Sie in die Klasse Widget einen QPushButton namens qButtonQuit hinzu.

Frage:
Was passiert?
Antwort:
Alle Elemente sind an derselben Stelle positioniert.



Lösung1:
Die Elemente positionieren und ihre Größe festlegen.

qLabelHello->setGeometry(x,y,w,h);

oder besser

Lösung2:
LayoutManager verwenden (s. Nächstes Kapitel)

Zur besseren Anordnung der Elemente verwenden wir die **Layout-Manager von Qt.**

widget.h (Auszug)

```
#include <QVBoxLayout>
#include <QHBoxLayout>
```

widget.cpp (Auszug)

```
  // 1. QPushButton und QLabel erzeugen
     …
```

```
// 2. Layout festlegen
  vlayout = new QVBoxLayout();
  hlayout = new QHBoxLayout();

  // 2.1. die Buttons horizontal
  hlayout->addWidget(qButtonHello);
  hlayout->addWidget(qButtonClear);
  hlayout->addWidget(qButtonQuit);

  // 2.2. das Label und die Buttons vertikal
  vlayout->addWidget(qLabelHello);
  vlayout->addLayout(hlayout);

  // 2.3. ins Formaular damit
  this->setLayout(vlayout);
```

1.5.2. Aufgabe: QWidget1-3 – Layout-Funktionalität

Integrieren Sie die Layout-Funktionalität, sodass das Programm folg. Aussehen hat.

1.6. Projekt: QWidget1-3: Signale/Slots verwenden

Projekt: QWidget2Hallo

Immer, wenn der Benutzer einen Button drückt (clicked()) (=Ereignis), soll eine entsprechende Aktion ausgeführt werden.

Wir wollen also
ein **SIGNAL** (zB. clicked()) mit einem sogenannten **SLOT** (zB. Clear(),close(),...) verbinden.

Es gibt eine Vielzahl geerbter Signale und Slots().

```
QWidget::connect(qButtonClear,      SIGNAL(clicked()),
                 qLabelHello,       SLOT(clear()));


QWidget::connect(qButtonQuit, SIGNAL(clicked()),
                 this,              SLOT(close()));
```

Achtung:
Im ersten Beispiel ist qLabelHello der Empfänger des Signals. Es wird der geerbte-Slot() clear aufgerufen.

Im zweiten Beispiel ist this (d.h. das Widget/Window) der Empfänger des Signals. Es wird der geerbte-Slot() close() aufgerufen.

Frage:

Aber was, wenn die geerbten Slots nicht ausreichen?

Wir wollen beim Click() des qButtonHello den Text des qLabelHello setzen mit "Hallo, Welt".

Die connect-Anweisung könnte folgendes Aussehen haben:

```
QWidget::connect(qButtonHello,      SIGNAL(clicked()),

                 this,              SLOT(on_qButtonHello_clicked()));
```

Nun interessiert uns der Slot() namens on_qButtonHello_clicked();

Es handelt sich dabei um einen frei wählbaren Namen, der in der Klasse Widget definiert werden muss:

widget.h

```
class Widget : public QWidget
{
    Q_OBJECT

private:
/* Dialogelemente */
      ...
/* Layout */
      ...

/* Slots sind Ereignisprozeduren */
 public slots:
    void on_qButtonHello_clicked();
    ...

public:
    Widget(QWidget *parent = 0);
      ...
    ~Widget();
};

#endif // WIDGET_H
```

widget.cpp

```
/* Slots sind Ereignisprozeduren */

void Widget::on_qButtonHello_clicked(){
    qLabelHello->setText("Hallo, Welt!");
}
```

1.6.1. Aufgabe: QWidget1-3 -Signal/Slot

Integrieren Sie die Signal/Slot-Funktionalität.

1.7. Projekt: QWidget2 - mit dem QT-Designer

Projekt: QWidget2

Wir wollen nun den QT-Designer kennenlernen. Es handelt sich dabei um einen GUI-Assistenten, mit dem man auf sehr einfache Art und Weise GUI-Anwendungen erstellen kann.

Designer starten

1. Qt-creator
2. Neue GUI-Anwendung: QWidget2HalloDesigner
3. Editieren Ansicht
4. Projekt evtl. aktiv schalten
5. Formular-Dateien
6. widget.ui doppelklick
7. wir sind nun im Designer

Dialogelemente einfügen und Layout festlegen

1. Buttons ins Widget/Formular reinziehen
    1. Im Eigenschaftsfenster(Property) Button umbenennen: ZB: pushButtonHallo
    2. Doppelklick auf Button, dann Text eingeben zB. Hallo
2. Horizontales Layout f. Buttons festlegen (alle Markieren und horiz. Layout ganz oben festlegen.
3. Label (s. li. Unten Display Widgets) reinziehen
    1. Im Eigenschaftsfenster(Property) Label umbenennen: ZB: labelHallo
    2. Doppelklick auf Label, dann Text eingeben zB. Hallo
4. Vertikales Layout f. Gesamtes Formular festlegen (alle Buttons Markieren und dann Label zusätzlich markieren) und vertikales Layout ganz oben festlegen.

5. Mit Menu(Erstellen → Ausführen) können Sie das Programm testen.

Signale und Slots festlegen: http://doc.trolltech.com/4.3/designer-connection-mode.html

1. Mit Menu(Bearbeiten → Signale/Slots) können Sie vererbte Signale/Slot Verbindungen zuordnen.
2. buttonClear mit li. Maustaste anwählen und bei weiterhin gedrückter Maustaste zum labelHallo bewegen.
   1. Ein Fenster mit zwei Listen wird angezeigt. Sie können nun eine connection auswählen: Links wählen Sie das Signal(clicked()) und in der rechten Liste wählen Sie den geerbten Slot(clear()) für das labelHallo.
3. Wir wollen nun den buttonQuit mit dem Signal(close()) des Widget/Formular verbinden:
   1. wählen Sie den buttonQuit mit der li. Maustaste und ziehen Sie bei weiterhin gedrückter Maustaste ins Widget/Formular.
   2. Wieder geht ein Fenster auf mit li. Und re. Liste. Diesmal klicken Sie aber links unten auf die Checkbox im linken Fenster (Signale und Slots von QWidget anzeigen). Damit werden auch die geerbten Signale/Slots angezeigt.
   3. Links wählen Sie nun clicked() und rechts close()
   4. fertig.

Spezielle Slots
Wie im vorigen Kapitel sind die speziellen Slots zu programmieren.

widget.h

```
...
 public slots:
    void on_pushButtonHello_clicked();
    ...
```

widget.cpp

```
void Widget::on_pushButtonHello_clicked(){
    ui->labelHello->setText("Hallo, Welt!");
}
```

Im Designer kann man diesen Slot automatisch generieren lassen, durch:

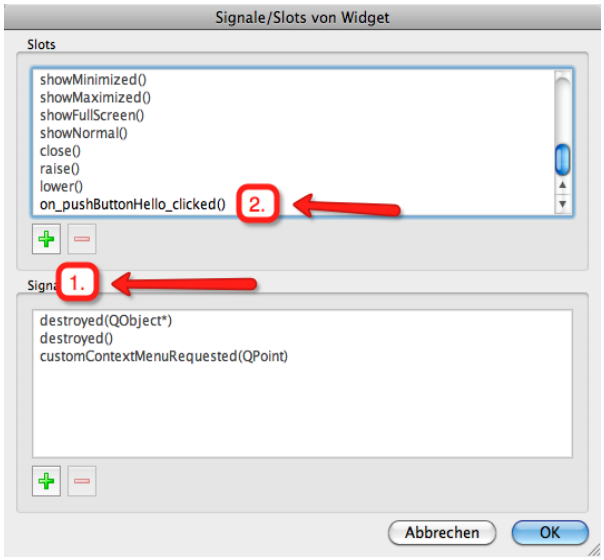
select QpushButton → re.Maus → GOTO SLOT

+Zusatz: +

Wenn wir diesen Slot(on_pushButtonHello_clicked()) mit dem QT-Designer verwenden wollen, müssen wir diesen slot() der Liste der Slots hinzufügen:

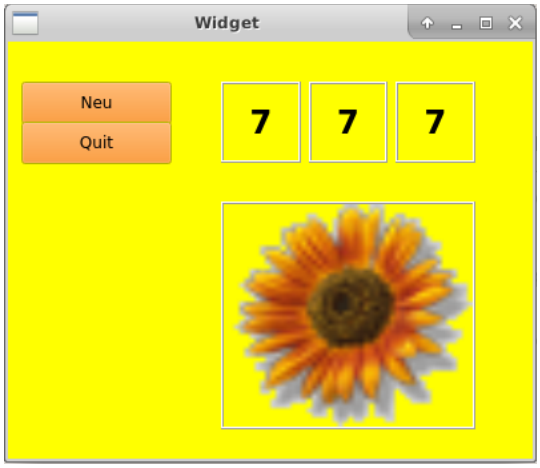
1. Menu (Bearbeiten → Signal/Slots bearbeiten)
2. pushButtonHello mit li. Maustaste anklicken und bei weiterhin gedrückter li. Maustaste in das Widget/Formular bewegen und Maus loslassen.

3. Im nun geöffneten Fenster müssen Sie im rechten Fensterbereich den Button Ändern anklicken. Daraufhin öffnet sich ein neues Fenster.



4. Hier drücken Sie im oberen Bereich den + Button und geben dann den neuen Slotnamen an: Hier on_pushButtonHello_clicked().
5. Wählen Sie dann ok und im zuvor geordneten Fenster können Sie nun diesen neuen Slot verwenden.
6. Fertig!

1.8. Projekt: QLucky7 - Label, Bilder, Zufallszahlen



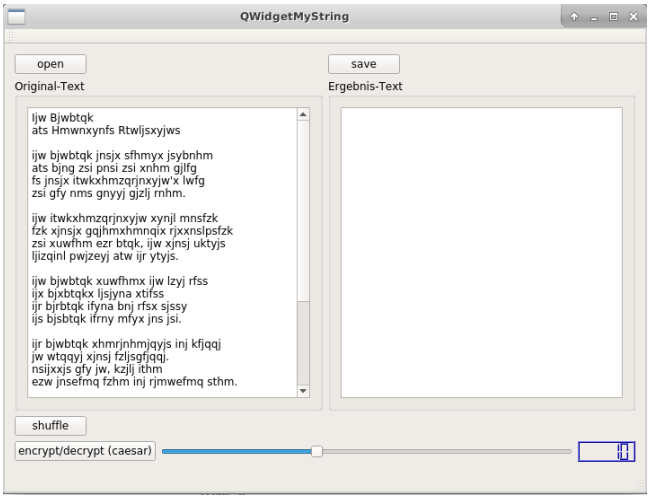
- mit Form-Designer (Fixed Size Policy)
- mit qrc (add New … → Qt Resource File)
- Label → Pixmap / scaledContents
- qrand()

- label.setNum() um eine Zahl in ein Label zu schreiben

```
QPixmap p(":/img/Sunflower.gif");

ui->labelBild->setPixmap(p);
ui->labelBild->setVisible(true);

int z1= qrand() % 9 + 1;
ui->labelBild->setNum(z1);
```

1.9. Projekt: QWidgetMyString

- Text aus a) Datei b) direkte Eingabe einlesen / speichern
- Text ver/entschlüsseln (caesar)
- Text mischen



Eine QWidget-APP mit Designer:

* Ein Textfile mit **QFileDialog** öffnen/speichern
* **QFile** (zum Lesen v. files)
* **QTextStream** (zum Schreiben v. files)
*

Aufbau UI:

* **QPushButton**:
  - open
  - save
  - shuffle
  - encrypt/decrypt
* **QSlider**:
* **QLCDNumber**:

* **QTextEdit** 2x (Original, Ergebnis)

Eigene Klassen hinzufügen:

* MyString (s.u.)
* CaesarStream (s.u.)

1.9.1. Hinweis: QFileDialog,QFile,QTextStream

```
void MainWindow::on_pushButton_open_clicked()
{
        QString fileName = QFileDialog::getOpenFileName(this,
                tr("Open Textfile"), ".", tr("Text Files (*.txt *.doc)"));
        if (!fileName.isEmpty()){
                QFile file(fileName);
                if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
                                return;
                ui->textEdit_Original->setText(file.readAll());
                file.close();
        }
}
void MainWindow::on_pushButton_save_clicked()
{
        QString fileName = QFileDialog::getSaveFileName(this,
                tr("Save Textfile"), ".", tr("Text Files (*.txt *.doc)"));
        if (!fileName.isEmpty()){
                QFile file(fileName);
                if (!file.open(QIODevice::WriteOnly | QIODevice::Text))
                                return;
                QString txt= ui->textEdit_Ergebnis->toPlainText();
                QTextStream out(&file);
                out<< txt;
                out.flush();
                file.close();
        }
}
```

1.9.2. Aufgabe: MYSTRING_SHUFFLE

Fügen Sie die Klasse MyString zu ihrem Projekt und erweitern Sie die Klasse MyString, sodass folg. möglich wird:

```
    string s= ui->textEdit_Original->toPlainText().toStdString();

    MyString mystring(s.c_str());

    s=mystring.shuffle();
    // pro Wort:
    //    1 und letztes Zeichen bleiben unverändert.
    //    alle anderen Zeichen werden per Zufall mehrfach untereinander
    //    ausgetauscht.
    // Der text in mystring bleibt aber unverändert.

    QString qstring(s.c_str());

    ui->textEdit_Ergebnis->setPlainText(qstring);
```

Beispiel:

Aus dem Original-Text:

```
der werwolf eines nachts entwich
von weib und kind und sich begab
an eines dorfschulmeister's grab
und bat ihn bitte beuge mich.
```

wird z.B. der Ergebnis-Text:

```
der wwoelrf eiens ntcahs ecwtinh
von weib und knid und scih bgaeb
an eenis driturshlmescfeo's garb
und bat ihn bttie bgeue mchi.
```

1.9.3. Aufgabe: Die Klasse CaesarStream (caesarstream.h)

Erstellen Sie die Klasse CaesarStream, die folgendermaßen verwendet werden kann:

Und bauen Sie diese in das Programm QWidgetMystring ein.

Datei: test-caesarstream.cpp

```
//test-caesarstream.cpp

#include "caesarstream.h"

int main() {
      CaesarStream cs(1);
      string sPlain= "Hallo";
      string sCipher;

      cs << sPlain;
      cout << cs << endl;  // gibt Ibmmp aus

      cs >> sCipher; // liefert den verschlüsselten Text
      cout << sCipher <<endl;  // gibt Ibmmp aus

      return 0;
}
```

Hinweis: Caesar-Chiffre
Cäsar-Chiffre: Zur Codierung seiner Nachrichten soll Cäsar folgendes Verfahren angewandt haben: Statt des Buchstabens, den er meinte, schrieb er den Buchstaben auf, der im Alphabet k Positionen rechts vom gemeinten Buchstaben ist.
Wenn man bei 'z' ankommt, muß man bei 'a' weiterzählen.

Beispiel:
Mit k=4 wird aus "hello world" der verschlüsselte Text "lipps asvph".
Dieses Verfahren wollen wir für den Rechner anwenden. Es werden nur alle Buchstaben (a-z bzw. A-Z) verschlüsselt. Alle anderen Zeichen (!?,. usw.) sollen nicht verschlüsselt werden.

Anmerkung1:

```
unsigned char ch;
...
ch=ch+key;
```

```
if(ch>'Z') ch= ch-26;
```

Entschlüsselt wird mit key=26-key;
Beispiel:
Hello, world! (key=4)
Lipps, asvph! (entschluesselt wird mit 22) (vgl: 26 - 4)

1.10. Projekt: QWebEngine-mini - Ein kleiner Webbrowser

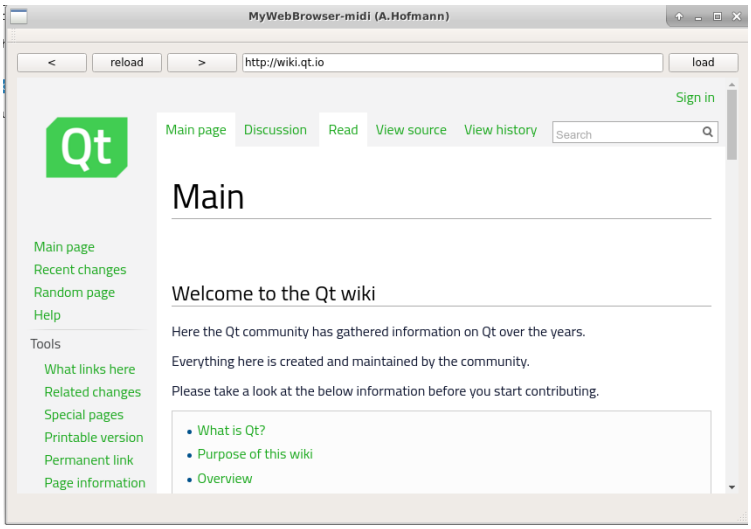
- Qt console app

```
QT += core
QT += gui
QT += webenginewidgets
CONFIG += c++11
TARGET = QWebEngine-mini
TEMPLATE = app
SOURCES += main.cpp
```

- main.cpp

```
#include <QApplication>
#include <QWebEngineView>
int main(int argc, char *argv[])
{
        QcoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);

        QApplication app(argc, argv);

        QWebEngineView view;

        view.setUrl(QUrl(QStringLiteral("https://fs-v3.htl-
salzburg.ac.at/WebUntis")));

        view.resize(1024, 750);
        view.show();

        return app.exec();
}
```

1.11. Project: QWebEngine-midi (QWebEngineView)



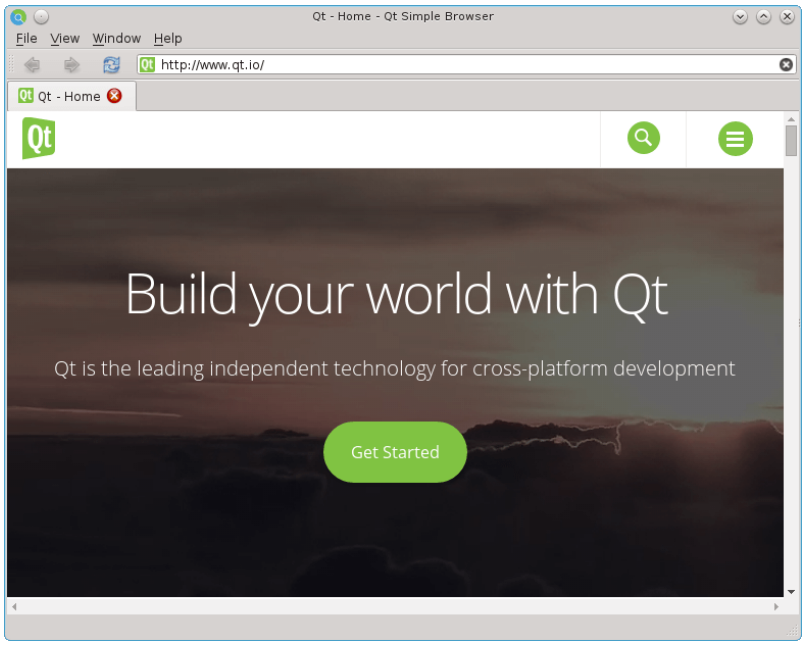
- Designer:
  - 4 Buttons (btnReload, btnBack, btnForward, btnLoad)
  - 1 lineEditURL
  - 1 Widget mit **Promote to** … (Basis: Qwidget, Klassenname: **QWebEngineView**)

Signal/Slot

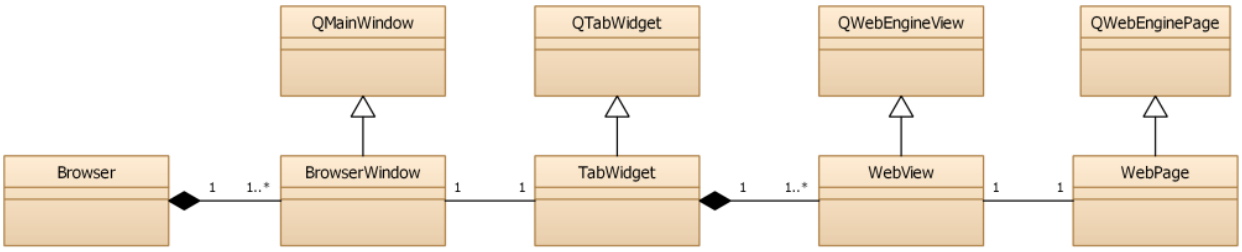
```
        //#include <QScreen>
        resize(QApplication::primaryScreen()->availableSize());


        //
        connect(ui->pushButtonLoad, SIGNAL(clicked(bool)),
                        this, SLOT(on_pushButtonLoad_clicked()));
        connect(ui->lineEditURL, SIGNAL(returnPressed()),
                        this, SLOT(on_pushButtonLoad_clicked()));

        // back, forward, reload
        connect(ui->pushButtonBack, SIGNAL(clicked(bool)),
                        ui->widgetWebEnigneView, SLOT(back()));
        connect(ui->pushButtonForward, SIGNAL(clicked(bool)),
                        ui->widgetWebEnigneView, SLOT(forward()));
        connect(ui->pushButtonReload, SIGNAL(clicked(bool)),
                        ui->widgetWebEnigneView, SLOT(reload()));
```

1.12. + Projekt: QWebEngine (UML)

http://doc.qt.io/qt-5/qtwebengine-webenginewidgets-simplebrowser-example.html



Class Hierarchy
We start with sketching a diagram of the classes that we are going to implement:



- `Browser` is a singleton class managing the application windows.
- `BrowserWindow` is a QMainWindow showing the menu, a navigation bar, `TabWidget`, and a status bar.
- `TabWidget` is a QTabWidget and contains one or multiple browser tabs.
- `WebView` is a QWebEngineView, provides a view for `WebPage`, and is added as a tab in `TabWidget`.
- `WebPage` is a QWebEnginePage that represents website content.

1.13. Projekt: QWidgetQRcode-process

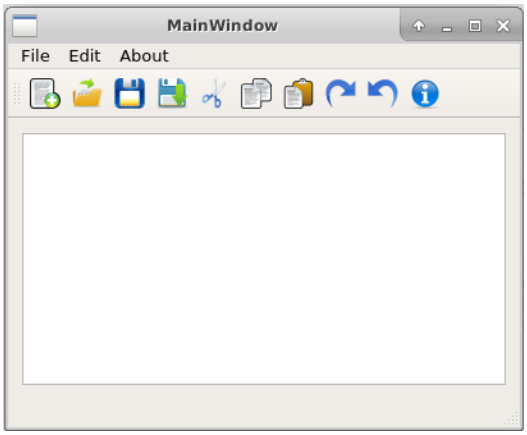
* sudo apt-get install libqrencode-dev
* qrencode -t PNG -o google.png http://www.google.at
* https://code.google.com/archive/p/qrencode-win32/

```
void MainWindow::on_pushButtonGetPicture_clicked()
{
	QString program = "/usr/bin/qrencode";
	QStringList arguments;
	arguments << "-t" << "PNG";
	arguments << "-o" << "google.png";
	arguments << ui->plainTextEdit->toPlainText();

	QProcess *myProcess = new QProcess(this);
	myProcess->setWorkingDirectory("/tmp");
	myProcess->start(program, arguments);
	myProcess->waitForFinished();


	QPixmap pix("/tmp/google.png");
	ui->labelQRpicture->setPixmap(pix);
	ui->labelQRpicture->update();
}
```

* siehe auch:
* git clone https://github.com/yoneal/qtqrencode.git

* https://ralgozino.wordpress.com/2011/05/29/introducing-qtqr-a-qr-code-generator-and-decoder/

1.14. Project: QNotepad



- http://www.codebind.com/cpp-tutorial/qt-tutorials-for-beginners-simple-notepad-application/
- http://www.iconarchive.com/search?q=about
- https://www.youtube.com/watch?v=j4j1gbYJtUw&index=26&list=PLS1QulWo1RIZiBcTr5urECberTITj7gjA

- DESIGNER / MainWindow
- **QMenue**
- **TextEdit** (horiz. layout)

- **QMenu** (mit Icons)

File
    New
    Open

    Save
    Save As
Edit
    Cut
    Copy
    Paste

    Redo
    Undo

- add qrc für die Icons (02-ueben/icons)
- im Designer ->Action fenster-> Doppelklick auf actionNew

    Icons auswählen

    ... auch alle anderen
- im Designer ->Action Fenster-> drag actionNew zur Toolbar

    ... auch alle anderen
- im Designer ->Action fenster-> go to slot ...

    ... auch alle anderen

```
void MainWindow::on_actionNew_triggered()
{

}
```

- copy,cut,paste

```
void MainWindow::on_actionCut_triggered()
{
      ui->textEdit->cut();
}
```

- FILE-NEW:
  class MainWindow
  private:
  QString file_path;

```
void MainWindow::on_actionNew_triggered()
{
      file_path="";
      ui->textEdit->clear();
}
```

includes:

```
#include      <QFile>
#include      <QFileDialog>
#include      <QTextStream>
```

* FILE-OPEN:
  * QFileDialog
  * file_path setzen
    * file_path= filename

* FILE-SAVE
  * file_path nutzen

* FILE-SAVE-AS
  * QFileDialog
  * file_path setzen
    * file_path= filename

```
void MainWindow::on_actionOpen_triggered()
{
      QString filename= QFileDialog::getOpenFileName(this, "Open File");
      if (!filename.isEmpty()){


            QFile file(filename);

            if (!file.open(QFile::ReadOnly|QFile::Text)){
                  QMessageBox::warning(this, "..", "file  not open");
                  return;
            }
```

```
                // remember !!!
                file_path= filename;

                QTextStream in(&file);
                ui->textEdit->setText( in.readAll());
                file.close();
          }
     }
```

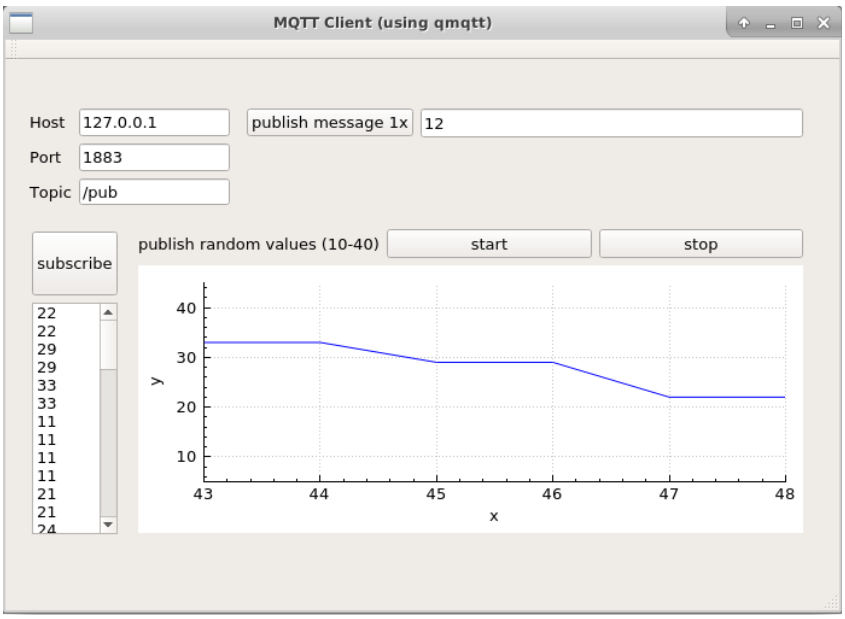
```
     void MainWindow::on_actionSave_triggered()
     {

          QFile file(file_path);

          if (!file.open(QFile::WriteOnly|QFile::Text)){
                QMessageBox::warning(this, "..", "file  not open");
                return;
          }

          QTextStream out(&file);
          QString sout= ui->textEdit->toPlainText(); //!!
          out << sout;
          file.flush();
          file.close();

     }
```

```
     void MainWindow::on_actionSave_As_triggered()
     {
          QString filename= QFileDialog::getSaveFileName(this, "Save as
File");
          if (!filename.isEmpty()){
                QFile file(filename);


                if (!file.open(QFile::WriteOnly|QFile::Text)){
                      QMessageBox::warning(this, "..", "file  not open");
                      return;
                }

                //remember
                file_path= filename;


                QTextStream out(&file);
                QString sout= ui->textEdit->toPlainText(); //!!
                out << sout;
                file.flush();
                file.close();
          }
     }
```

```
     void MainWindow::on_actionAbout_triggered()
     {
          QString about_text="Notepad<br>";
          about_text += "Author: A.Hofmann<br>";
          about_text += "<a href='https://www.youtube.com/watch?
v=j4j1gbYJtUw&index=26&list=PLS1QulWo1RIZiBcTr5urECberTITj7gjA'>Quelle:
```

```
URL</a>";

            QMessageBox msgBox(this);
            msgBox.setWindowTitle("About");
            msgBox.setTextFormat(Qt::RichText);
            msgBox.setText(about_text);
            msgBox.exec();
      }
```

1.15. Projekt: QMQTT-Plot (Internet of Things)



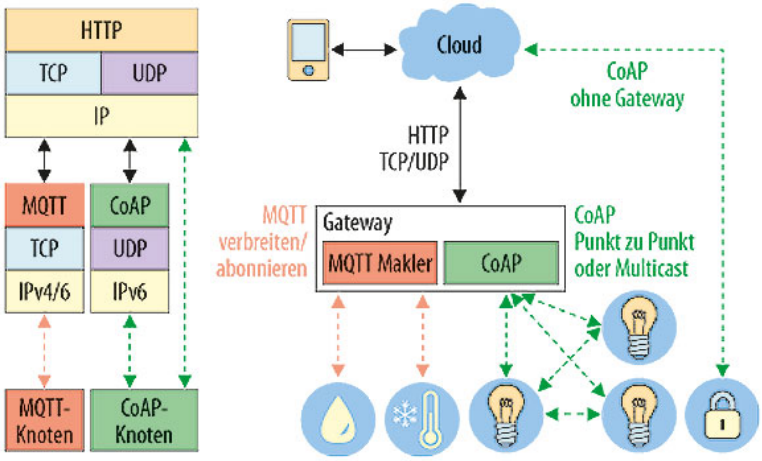
1.15.1. MQTT

https://de.wikipedia.org/wiki/MQTT
"MQTT (**MQ Telemetry Transport** oder **Message Queue Telemetry Transport**) ist ein offenes Nachrichtenprotokoll für Machine-to-Machine-Kommunikation (M2M)….

Seit 2013 wird MQTT über die Organization for the Advancement of Structured Information Standards (OASIS) als Protokoll des Internet der Dinge standardisiert."

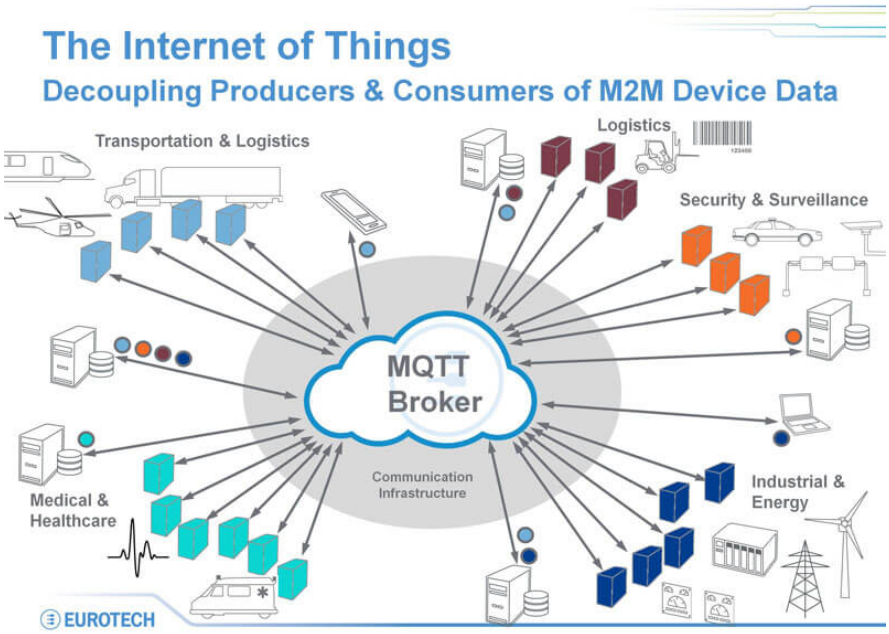
1.15.2. The Big Picture

Das MQTT Protokoll:



http://cdn2.weka-fachmedien.de/media_uploads/images/1450354944-105-bild-2.jpg

Anwendungen:



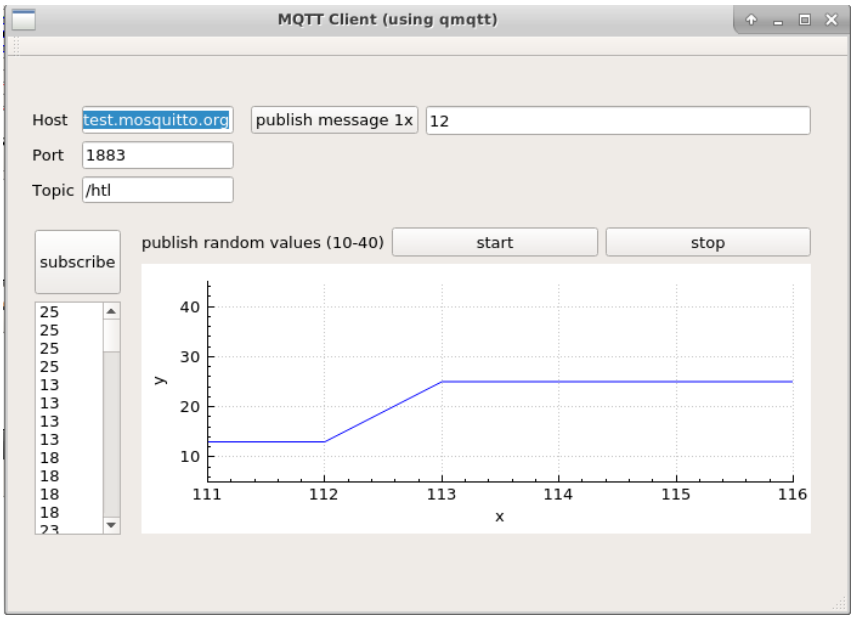
Quelle: https://cdn.thenewstack.io/media/2016/04/M2M.jpg

1.15.3. Mosquitto: Installieren und testen mit mqtt-spy

https://mosquitto.org/documentation/

Aufgabe:
Installieren sie einen lokalen Mosquitto-Server und testen Sie diesen z.B. mit mqtt-spy

Aufgabe:
Nutzen Sie den Test-Server: test.mosquitto.org mit dem Topic /htl



1.15.4. Projekt: QMQTT – Ein MQTT-Client (publish und subscribe)

git clone https://github.com/emqtt/qmqtt
Qt: Projekt: QMQTT
copy qmqtt/src/mqtt to QMQTT
add existing directory

```
INCLUDEPATH += mqtt
```

QMQTT.pro

```
#-------------------------------------------------
#
# Project created by QtCreator 2017-04-13T11:59:03
#
#-------------------------------------------------

QT       += core gui

# qmqtt needs
QT       += network

# wegen QCustomPlot
# http://www.qcustomplot.com/index.php/tutorials/settingup
# http://www.qcustomplot.com/index.php/tutorials/basicplotting
#
QT       += printsupport

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = MQTT-client-with-qmqtt
TEMPLATE = app

# qmqtt als sourcen hinzufügen
# project->add exsting directory
# ordner qmqtt/include und qmqtt/src

SOURCES += main.cpp\
```

```
        mainwindow.cpp \
        qcustomplot.cpp \
    mqtt/qmqtt_client.cpp \
    mqtt/qmqtt_client_p.cpp \
    mqtt/qmqtt_frame.cpp \
    mqtt/qmqtt_message.cpp \
    mqtt/qmqtt_message_p.cpp \
    mqtt/qmqtt_network.cpp \
    mqtt/qmqtt_routedmessage.cpp \
    mqtt/qmqtt_router.cpp \
    mqtt/qmqtt_routesubscription.cpp \
    mqtt/qmqtt_socket.cpp \
    mqtt/qmqtt_ssl_network.cpp \
    mqtt/qmqtt_ssl_socket.cpp \
    mqtt/qmqtt_timer.cpp

INCLUDEPATH += mqtt

HEADERS  += mainwindow.h \
        qcustomplot.h \
    mqtt/qmqtt.h \
    mqtt/qmqtt_client.h \
    mqtt/qmqtt_client_p.h \
    mqtt/qmqtt_frame.h \
    mqtt/qmqtt_global.h \
    mqtt/qmqtt_message.h \
    mqtt/qmqtt_message_p.h \
    mqtt/qmqtt_network_p.h \
    mqtt/qmqtt_networkinterface.h \
    mqtt/qmqtt_routedmessage.h \
    mqtt/qmqtt_router.h \
    mqtt/qmqtt_routesubscription.h \
    mqtt/qmqtt_socket_p.h \
    mqtt/qmqtt_socketinterface.h \
    mqtt/qmqtt_ssl_network_p.h \
    mqtt/qmqtt_ssl_socket_p.h \
    mqtt/qmqtt_timer_p.h \
    mqtt/qmqtt_timerinterface.h

FORMS    += mainwindow.ui
```

1.15.5. Node red and node red dashboard

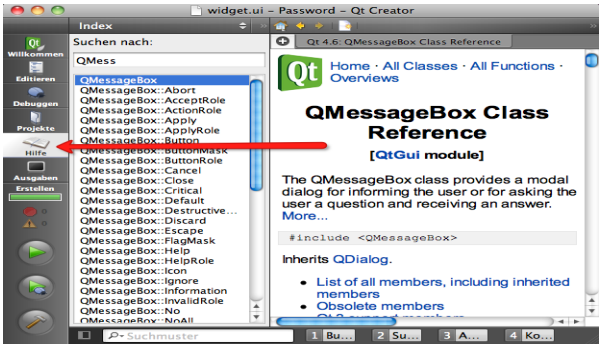
todo ????

1.16. Projekt: QRSA

1.17. Projekt: QGameOfLife

1.18. Weitere GUI-Programme

Verwenden Sie die im QT-Creator integrierte Hilfe. Sie ist wirklich einen Blick wert.



1.18.1. Password: Timer

Hinweise:

- QTimer
- QMessageBox

1.18.2. Bildlauf: Slider, Positionieren

1.18.3. Pizza: Check-, Listbox, EditorLine, …

Siehe:

HOWTO-Video: qwidgetPIZZA-1

HOWTO-Video: qwidgetPIZZA-2

1.18.4. Projekt: ImageViewer

http://www.bogotobogo.com/Qt/Qt5_QMainWindow_QAction_ImageViewer.php

Eine MainWindow-App zum Anzeigen,Zoomen,Skalieren u. Drucken von Bildern (mit Menü) unter Verwendung des Qt-Designers.

Anmerkung: Zoomen versus Fit to Window

Im Menu View kann eine Checkbox für Fit to Window angewählt werden.

- Wenn diese selektiert ist, übernimmt die ScrollArea das Skalieren des Bildes auf die

gesamte Area, d.h. das Zoomen an sich ist nicht möglich. Das Bild wird in seiner Gesamtheit dargestellt.
QscrollArea::widgetResizeable ist true.

- Wenn die Checkbox für Fit to Window NICHT selektiert ist (QscrollArea::widgetResizeable ist false), ist das Zoomen möglich, weil dann das QLabel das Skalieren übernimmt. (QLabel::scaledContents)
  ...
  imageLabel->setScaledContents(true);

  ...

```
if (!ui->actionFit_to_Window->isChecked())
      ui->imageLabel->adjustSize();
      // oder: das geht auch
      // ui->imageLabel->resize(ui->imageLabel->pixmap()->size());
}
```

1.18.4.1 Slot: fitToWindow()

fitToWindow() wird bei jedem click auf die Checkbox Fit to Window aktiviert.

```
void ImageViewer::fitToWindow()
{
    bool fitToWindow = fitToWindowAct->isChecked();
    scrollArea->setWidgetResizable(fitToWindow);
    if (!fitToWindow)
        normalSize();
    updateActions();
}
```

Bei aktivem Fit to Window, soll die ScrollArea sein Child (imageLabel) resize

1.18.4.2 UI und der ImageViewer Konstruktor

- **QLable** (zum Bildanzeigen inkl. QLabel::scaledContents)
- **QScrollArea** (zum automatischen Anpassen an den Inhalt durch QScrollArea::widgetResizable)
- **QPainter** (zum Drucken)

ui-properties: (werden mit QT-Designer gesetzt)

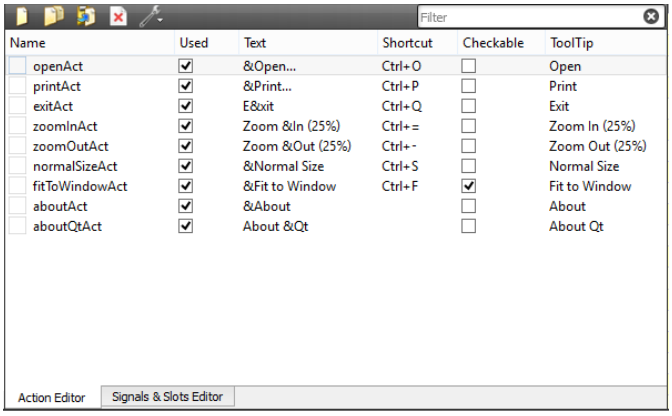
```
// Hintergrundfarbe setzen mit vordefinierter Farbe
   imageLabel->setBackgroundRole(QPalette::Base);

// Damit wird beim Zoomen das Bild skaliert ohne daß autom. Scroll-Balken erscheinen.
// Diese Scroll-Balken würden bei  QSizePolicy::Preferred automatisch erscheinen, was hier
// nicht gewollt ist.
   imageLabel->setSizePolicy(QSizePolicy::Ignored, QSizePolicy::Ignored);
   // Das Bild soll immer den gesamten Bereich des Labels ausfüllen.
   imageLabel->setScaledContents(true);
```

```
scrollArea->setBackgroundRole(QPalette::Dark);
scrollArea->setWidget(imageLabel);
setCentralWidget(scrollArea);
```

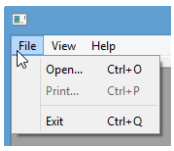
1.18.4.3 Menü: ActionEditor

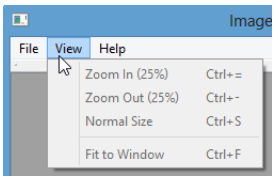
- File
  - Open
  - Print
  - Exit
- View
  - Zoom In - Scale the image up by 25%
  - Zoom Out - Scale the image down by 25%
  - Normal Size - Show the image at its original size
  - Fit to Window - Stretch the image to occupy the entire window
- Help
  - About

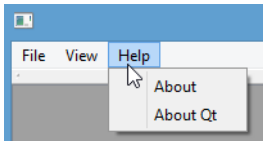
Wir verwenden den Action Editor im Qt-Designer:



Folgende Menüs (beachte Property: enabled)

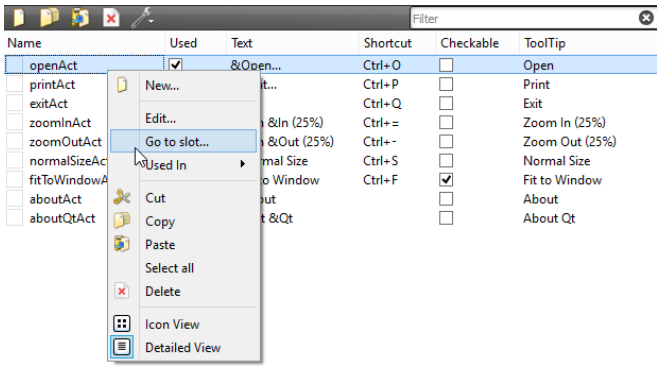


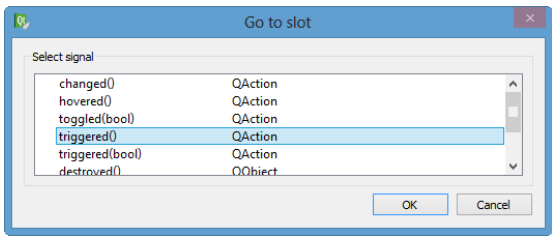




1.18.4.4 Menü: Signal/Slot

ActionEditor → Goto Slot … (triggered())





1.18.4.5 Image laden

```
void MainWindow::on_actionOpen_triggered()
{
        qDebug() << "open()";
        QString fileName = QFileDialog::getOpenFileName(this,
                                tr("Open File"), QDir::currentPath());

        if (!fileName.isEmpty()) {
                QImage image(fileName);
                if (image.isNull()) {
                        QMessageBox::information(this,
                          tr("Image Viewer"),
                          tr("Cannot load %1.").arg(fileName));
                        return;
                }
                ui->imageLabel->setPixmap(QPixmap::fromImage(image));

                // default Skalierungs-faktor setzen
                this->scaleFactor = 1.0;

                // default MenueItems
                ui->actionPrint->setEnabled(true);
                ui->actionFit_to_Window->setEnabled(true);

//              updateActions();

                // Wenn Fit to Window nicht aktiviert wurde,
                // muss das imageLabel für die richtige Größe sorgen.
                // Wenn  Fit to Window aktiviert wurde,
                // kümmert sich die ScrollArea, sodass sein 'Kind',
                // das imageLabel größenmäßig angepasst wird
                if (!ui->actionFit_to_Window->isChecked())
                   ui->imageLabel->adjustSize();
                   // oder: das geht auch
```

```
                    // ui->imageLabel->resize(ui->imageLabel->pixmap()->size());
        }
}
```

1.18.4.6 Printer:

```
QT       += printsupport
```

mainwindow.h

```
...
private:
      #ifndef QT_NO_PRINTER
          QPrinter printer;
      #endif

      double scaleFactor;
```

todo

```
    void updateActions();
    void scaleImage(double factor);
    void adjustScrollBar(QScrollBar *scrollBar, double factor);
```

1.19. Hinweise / FAQ

1.19.1. Hinweis: Konvertierungen: QString. toInt() u. QString.setNum()

QString → int

```
      QString str("1234");

      int wert= str.toInt();
```

int → QString

```
      QString str;
     str.setNum(1234);       // str == "1234"
```

int/double in ein Label,... schreiben

muss nicht extra in einen QString konvertiert werden, sondern:

```
    ui->labelZahl->setNum(wert);
```

1.19.2. Hinweis: QString und arg

```
QString line = tr("<b>%1</b> says: <i>%2</i>").arg(nick).arg(message);
```

1.19.3. Hinweis: Window Größen festlegen

Wir wollen diesmal auch die Maximale und minimale Größe definieren können.

```
    setFixedSize(200, 120);

    setGeometry(62, 40, 75, 30);
```

Oder den gesamten Bildschirm nutzen

```
#include <QScreen>

resize(QApplication::primaryScreen()->availableSize());
```

1.19.4. Hinweis: Umlaute

Wenn der Editor mit UTF-8 arbeitet:

```
QString::fromUtf8( "überhaupt nicht" )
```

1.19.5. Hinweis: Fonts

```
    quit->setFont(QFont("Times", 18, Qfont::Bold));
```

1.19.6. Hinweis: Layout

widget.h (Auszug)

```
#include <QtGui/QVBoxLayout>
#include <QtGui/QHBoxLayout>
```

widget.cpp (Auszug)

```
  // 2. Layout festlegen
    vlayout = new QVBoxLayout();
    hlayout = new QHBoxLayout();

    // 2.1. die Buttons horizontal
    hlayout->addWidget(qButtonHello);
    hlayout->addWidget(qButtonClear);
    hlayout->addWidget(qButtonQuit);

    // 2.2. das Label und die Buttons vertikal
    vlayout->addWidget(qLabelHello);
    vlayout->addLayout(hlayout);

    // 2.3. ins Formaular damit
    this->setLayout(vlayout);
```

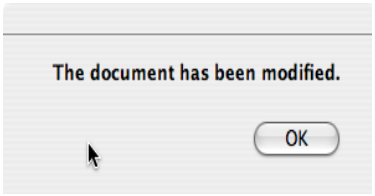
1.19.7. Hinweis: QDebug

#include <QDebug>

qDebug() << "The document has been modified.";

1.19.8. Hinweis: QMessageBox

```
QMessageBox msgBox;
msgBox.setText("The document has been modified.");
msgBox.exec();
```



1.19.9. Hinweis: QTimer

```
    // TImer ------------
    QTimer* timer= new QTimer();
    timer->setInterval(1000);
    timer->start();

    QObject::connect(timer, SIGNAL(timeout()),
                     this, SLOT(on_timer_timeout()));
```

in der cpp-Datei:

```
void Widget::on_timer_timeout(){
      ...
}
```

in der h-Datei:

```
public slots:
    void on_timer_timeout();
```

1.19.10. Hinweis: QButton/QLabel Farbe setzen

```
//Für einen Button:
// neuen Button anlegen
QPushButton *button = new QPushButton;

// Palette des Buttons holen
```

```
QPalette palette = button->palette();

// gewünschte Werte ändern
palette.setColor(QPalette::Button,QColor(255,255,0));

// Palette setzen
button->setPalette(palette);


// Für eine Label:
// neues Label anlegen
QLabel *label = new QLabel;
// Palette des Labels holen
QPalette palette = label->palette();
// gewünschte Werte ändern
palette.setColor(QPalette::Background,QColor(255,255,0));
// Palette setzen
label->setPalette(palette);
```

Designer:

QWidget::setAutoFillBackground muß enabled sein

QWidget::Pallete setzen

1.19.11. Hinweis: QLabel Bild laden

```
QPixmap p("ubuntu.png");

ui->labelBild->setPixmap(p);

ui->labelBild->setVisible(true);
```

1.19.12. Hinweis: QRadioButton, QComboBox, QCheckBox, QListWidget,QList

```
radioButtonGROSS->setChecked(true);

checkBoxOLIVEN->setChecked(true);

listWidgetZUSATZ->setSelectionMode(QAbstractItemView::MultiSelection);
listWidgetZUSATZ->addItem(QString::fromUtf8( "Fanta" ));
listWidgetZUSATZ->addItem("Cola");
listWidgetZUSATZ->addItem("Clausthaler");


comboBoxBEZAHLUNG->addItem(QString::fromUtf8( "Bar" ));
comboBoxBEZAHLUNG->addItem(QString::fromUtf8( "überhaupt nicht" ));
comboBoxBEZAHLUNG->setCurrentIndex(1);
```

```
    QString s;

    if (ui->radioButtonGROSS->isChecked()) ...
    if (ui->checkBoxOLIVEN->isChecked()) ...
```

```
    QList<QListWidgetItem*> zusaetze;
    zusaetze= ui->listWidgetZUSATZ->selectedItems();

    for (int i=0; i < zusaetze.size(); i++){
        s+= "    " + zusaetze.at(i)->text() + "\n";
    }
```

1.19.13. Hinweis: QFileDialog,QFile,QTextStream

```
void MainWindow::on_pushButton_open_clicked()
{
        QString fileName = QFileDialog::getOpenFileName(this,
                tr("Open Textfile"), ".", tr("Image Files (*.txt *.doc)"));
        if (!fileName.isEmpty()){
                QFile file(fileName);
                if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
                                return;
                ui->textEdit_Original->setText(file.readAll());
                file.close();
        }
}
void MainWindow::on_pushButton_save_clicked()
{
        QString fileName = QFileDialog::getSaveFileName(this,
                tr("Save Textfile"), ".", tr("Image Files (*.txt *.doc)"));
        if (!fileName.isEmpty()){
                QFile file(fileName);
                if (!file.open(QIODevice::WriteOnly | QIODevice::Text))
                                return;
                QString txt= ui->textEdit_Original->toPlainText();
                QTextStream out(&file);
                out<< txt;
                out.flush();
                file.close();
        }
}
```

1.19.14. Hinweis: QListView und QStringList, QStringListModel

```
// 1. View erstellen
QListView* listView = new QListView();

//2. Model erstellen und Model u. View verbinden
QStringListModel* model = new QStringListModel(listView);
listView->setModel(model);

//3. Daten erzeugen
QStringList list;

list << "eins" << "zwei"<<"drei";

//4. Daten anzeigen
model->setStringList(list);
```

1.19.15. Hinweis: externe Libraries hinzufügen

http://doc.qt.io/qt-5/third-party-libraries.html

1.20. Ausblick

Wir wollen uns in der Folge mit den Themen: Grafik, Mauseingaben, … beschäftigen