Inhaltsverzeichnis

1. QT Grundlagen: Widgets....1
1.1. Ziele....1
1.2. Vorkenntnisse....1
1.3. QWidget1-1: Ein leeres Fenster....2
1.3.1. Das main-Programm....2
1.3.2. Die eigene Klasse: Widget....3
1.3.3. Aufgabe: QWidget1 - resize()....4
1.4. QWidget1-2: Ein leeres Fenster mit QPushButton....4
1.4.1. Aufgabe: Einen QPushButton erstellen....4
1.4.2. Aufgabe: QPushButton Signale....5
1.4.3. Aufgabe: QPushButton: Slot-Signal....5
1.5. Qwidget1-3: Layouts verwenden....6
1.5.1. Aufgabe: QWidget1-3 – Add 1 QLabel und 3 QPushButtons....6
1.5.2. Aufgabe: QWidget1-3 – Layout-Funktionalität....7
1.6. QWidget1-3: Signale/Slots verwenden....7
1.6.1. Aufgabe: QWidget1-3 -Signal/Slot....8
1.7. QWidget2 - mit dem QT-Designer....8

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.

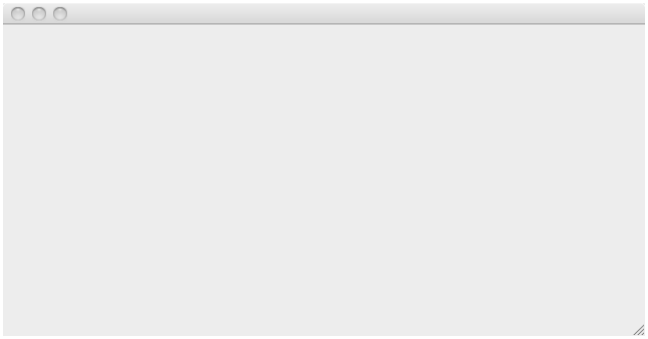
1. Explorer: Workspace erstellen: zb: schule\ws-qt
2. Qt-Creator starten
3. Neues Projekt: Name und Ordner wählen
4. **Qt-Gui-Anwendung** wählen und folg. Einstellungen vornehmen

   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. QWidget1-1: Ein leeres Fenster

Projekt: QWidget1-hello



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()
{
```

```
}
```

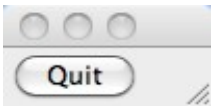
1.3.3. Aufgabe: QWidget1 - resize()

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. 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. 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. 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. QWidget2 - mit dem QT-Designer

Projekt: QWidget2-hello-designer

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!");
}
```

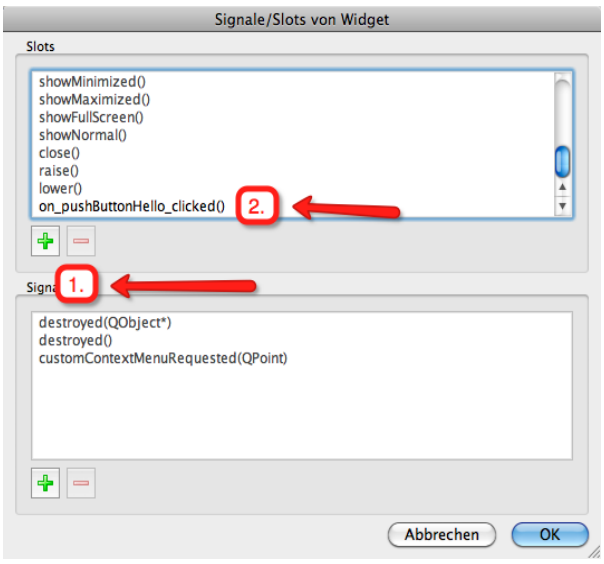
Im Designer kann man diesen Slot automatisch generieren lassen, durch:

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!