

QT – C++ GUI Toolkit

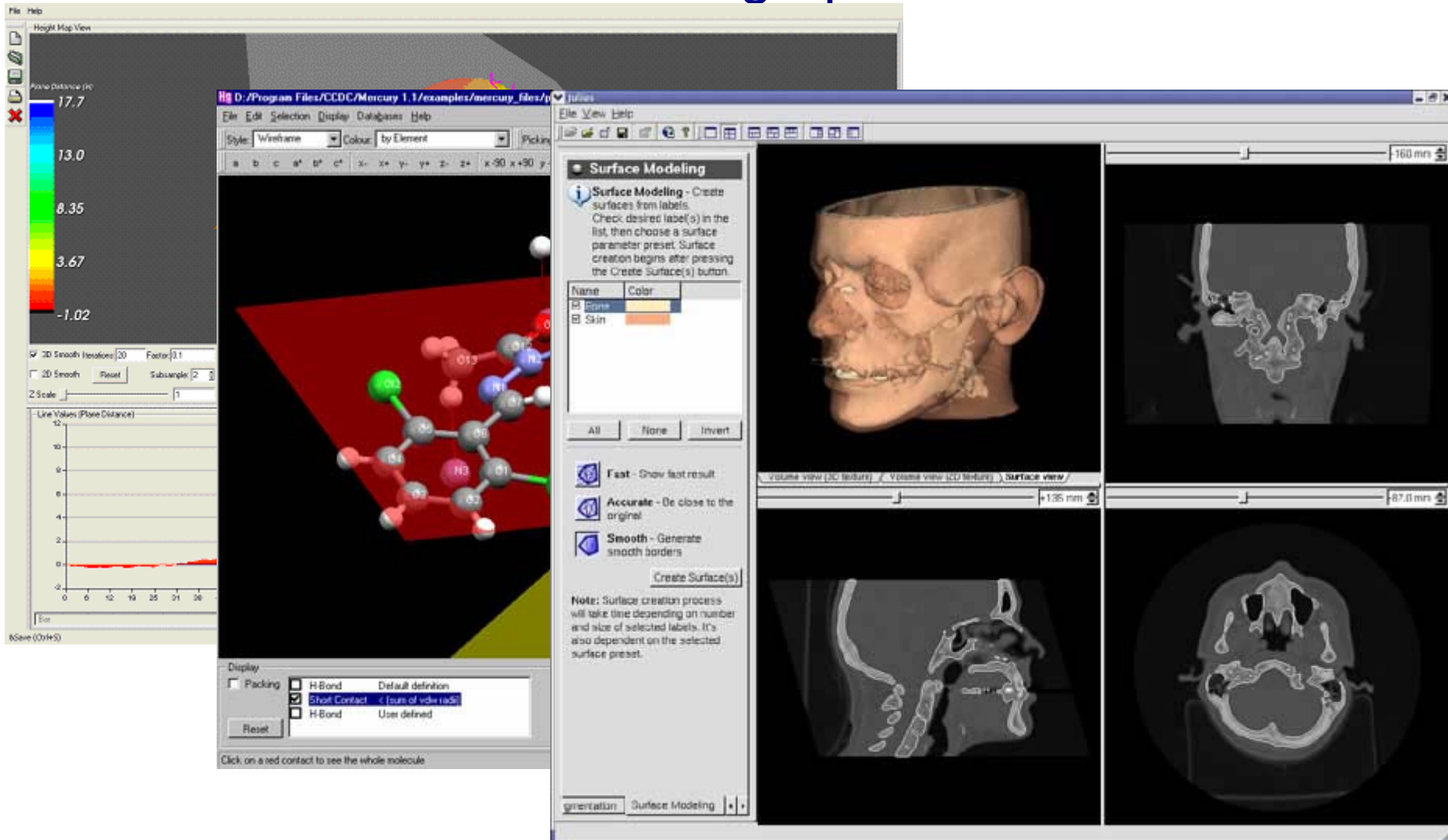
Tobias Sielhorst

06 December 2006

chair for computer aided medical procedures

department of computer science | technische universität münchen

QT - Motivation für eine graphische Oberfläche



Inhalt

- QT - Teil I (diese Woche)
 - Einführung in QT
 - Widgets (Fenster und Dialoge)
 - Hello World
 - Signals and Slots (Neues Programmierkonzept)
 - GUI
 - Layout Manager
 - Weiterführende Tutorials/Referenzen

- QT - Teil 2 (nächste Woche)
 - OpenGL in QT
 - Manipulation von 3D Szenen

Einführung in QT

- C++ Framework
- Cross Plattform (Win, Mac, Unix/Linux,...)
- Komplette Objektorientiert
- Grosse Auswahl an *Widgets* (diverse GUI Funktionalitäten)
- Erweiterung des *C++ Object Models* mit neuen Methoden für die Kommunikation zwischen Objekten (*Signals and Slots*)

Widgets

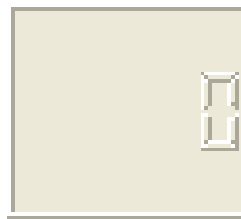
- Widgets sind visuelle Elemente einer graphischen Benutzerschnittstelle
- Grosse Auswahl an vordefinierten Widgets für GUIs
- Einfache Vererbung von vordefinierten Widgets um individuelle Funktionalität hinzuzufügen
- Können beliebig kombiniert werden
- Einfache Knöpfe und Textfelder bis zu komplexen Dialogen

- Widget Gallery
<http://doc.trolltech.com/4.0/gallery.html>

Widget Gallery – Display Widgets



- QProgressBar

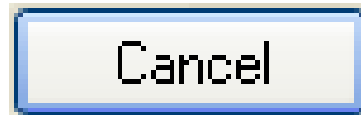


- QLCDNumber



- QLabel

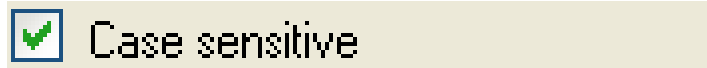
Widget Gallery – Buttons



- QPushButton



- QToolButton

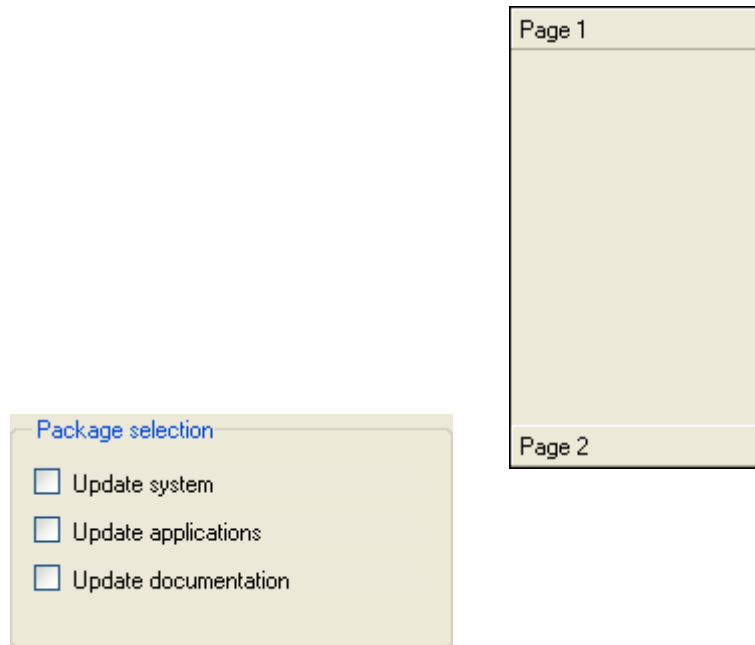


- QCheckBox



- QRadioButton

Widget Gallery – Containers

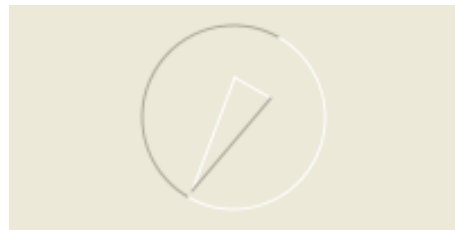


▫ QToolBox

▫ QGroupBox

▫ QTabWidget

Widget Gallery – Input Widgets



The **QTextEdit** class provides a widget that is used to edit and display both plain and rich text.

QTextEdit is an advanced *WYSIWYG* viewer/editor that can display images, lists and tables.



- QComboBox
- QLineEdit
- QSlider

- QDial

- QTextEdit

- QScrollBar

Hello World Button – Code Snippet

```
#include <QtGui/QApplication>
#include <QtGui/QPushButton>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    QPushButton hello("Hello world!");
    hello.resize(100, 30);

    hello.show();
    return app.exec();
}
```

Layout Manager

- QVBoxLayout
- QHBoxLayout
- QGridLayout

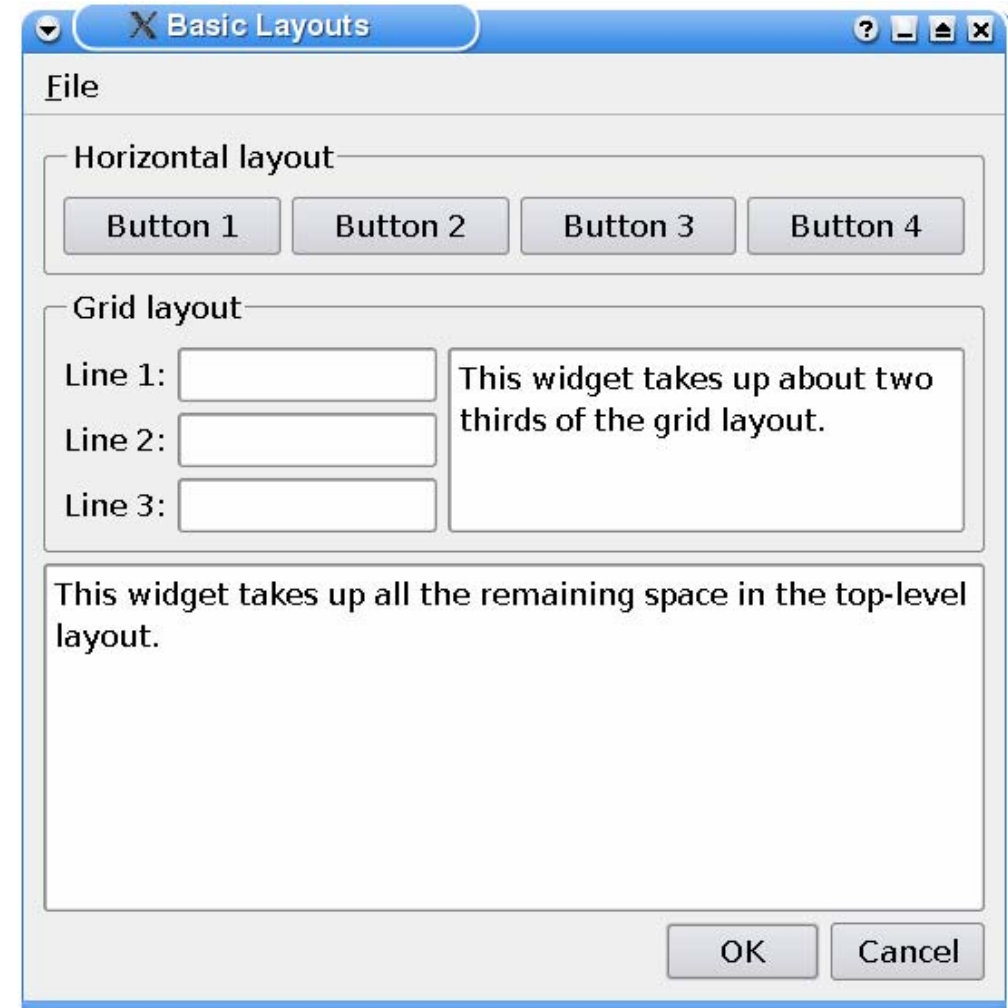
`myLayout->addWidget(myWidget)`

`myLayout->addLayout(myInnerLayout)`

`myLayout->addSpacing(int space)`

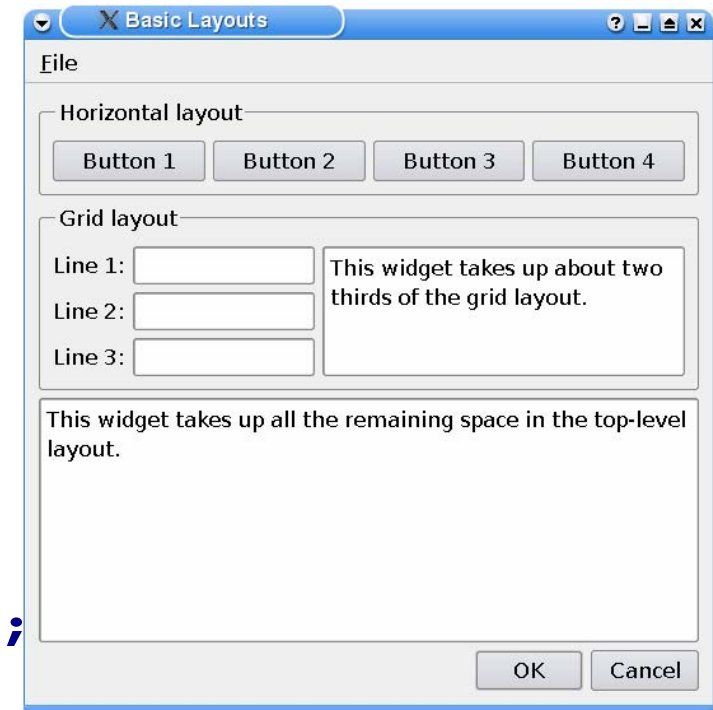
or

`insert<...>(int index, <...> myWidget)`



Layout Manager – Code Snippet

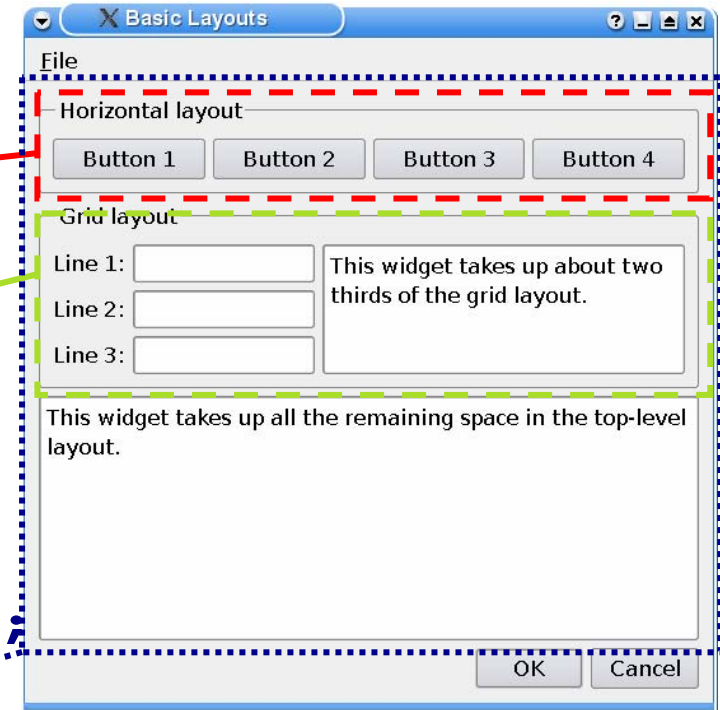
```
myHLayout->addWidget(button1);  
myHLayout->addWidget(button2);  
(...)  
myGLayout->addWidget(0,0,label1);  
myGLayout->addWidget(0,1,textField1);  
(...)  
myGLayout->addWidget(0,2,0,2,textEdit1);  
(...)  
myVLayout->addLayout(myHLayout);  
myVLayout->addLayout(myGLayout);  
myVLayout->addWidget(textEdit2);
```



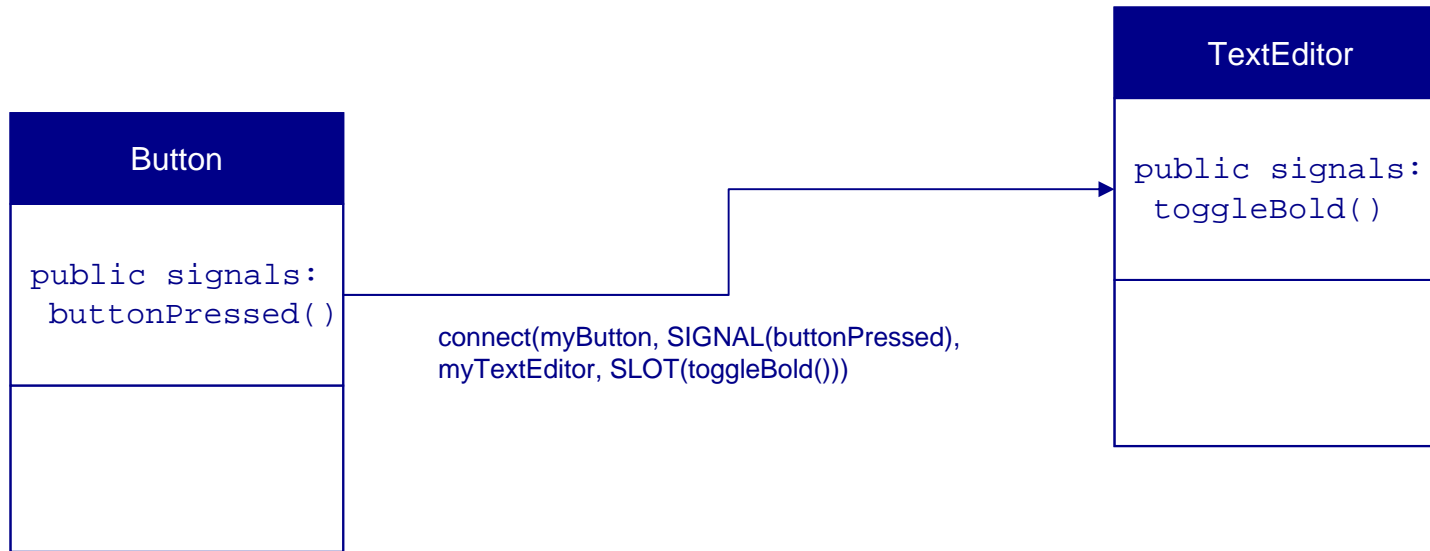
Layout Manager – Code Snippet

```

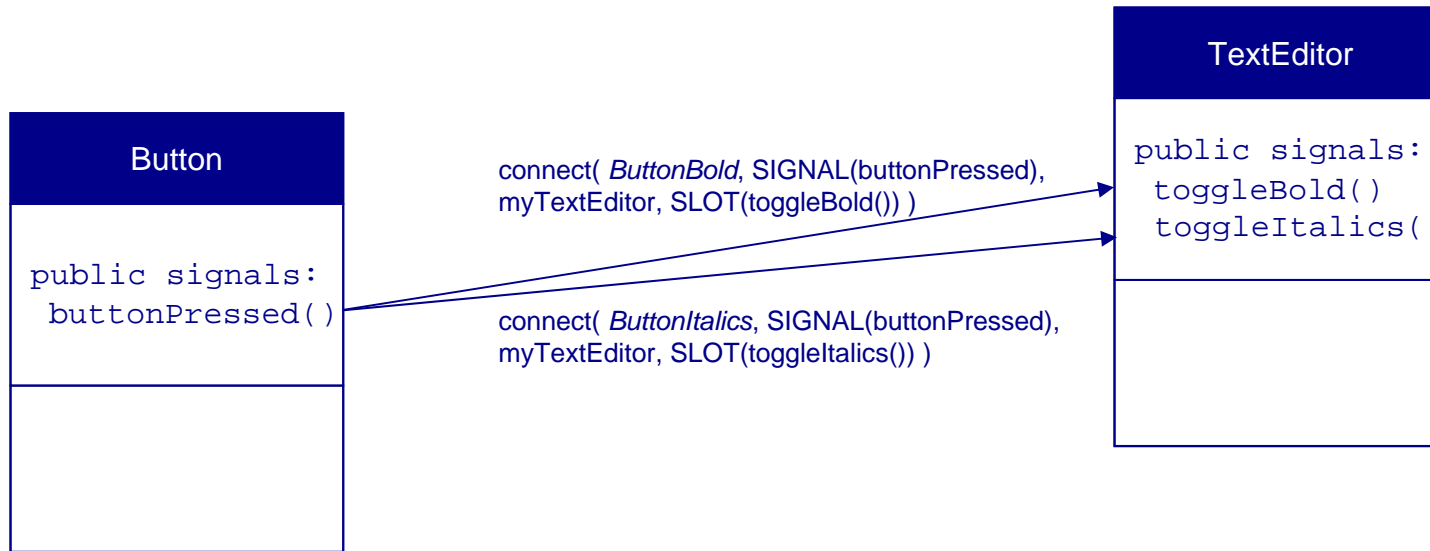
myHLayout->addWidget(button1);
myHLayout->addWidget(button2);
(...)
myGridLayout->addWidget(0,0,label1);
myGridLayout->addWidget(0,1,textField1);
(...)
myGridLayout->addWidget(0,2,0,2,textEdit1);
(...)
myVLayout->addLayout(myHLayout);
myVLayout->addLayout(myGridLayout);
myVLayout->addWidget(textEdit2);
    
```



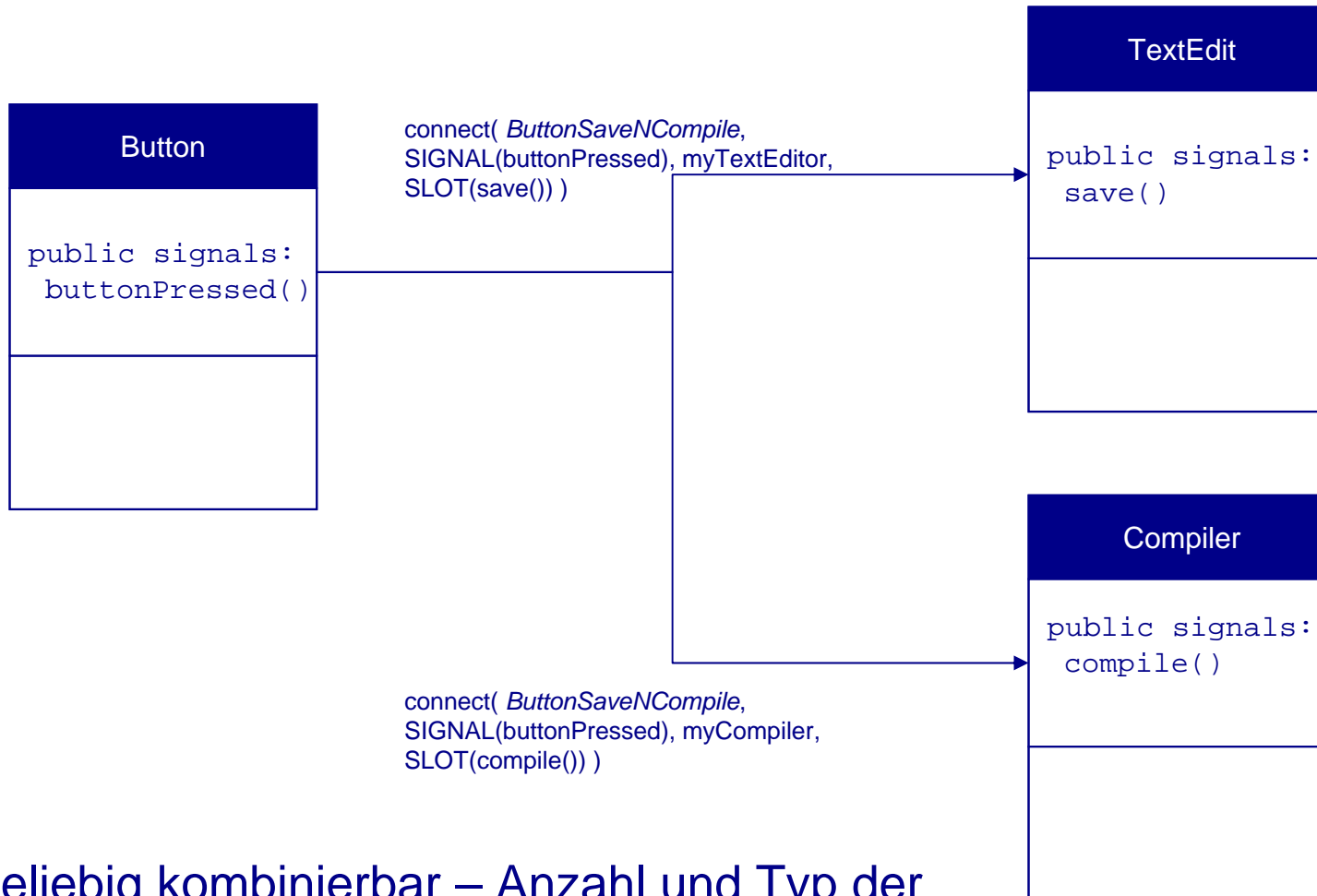
Signals and Slots



Signals and Slots



Signals and Slots



Beliebig kombinierbar – Anzahl und Typ der Parameter müssen jedoch übereinstimmen

Slots and Signals

- **Signals:** Widgets erzeugen Signale wenn bestimmte Aktionen ausgeführt werden (z.B. Button -> `clicked()`)
- **Slots:** Funktionen die von *Signals* (mittels `connect()`) oder auch normal aufgerufen werden können
- **Connect:** Verbindet ein *Signal* mit einem *Slot*
- **Beispiel:**
`connect(button, SIGNAL(clicked()), QMainWindow, SLOT(quit()));`

Benutzerdefinierte Slots and Signals

- Um eine Klasse mit *Signals* und *Slots* zu erweitern, muss die Klasse von `QObject` oder einer Unterklasse vererbt sein
- Das `Q_OBJECT` macro muss in der Klassendefinition stehen
- *Slots* können in der *public slots*, *protected slots* oder *private slots* Sektion stehen
- Es muss der `moc` compiler vor dem Präprozessor verwendet werden
 - Header-Datei `mocen`
 - Entstandene `cpp`-Datei in das Projekt einfügen

Der Moc

- Übersetzt Qt spezifischen Code in echten C++ code
- In VS: Header→Property→Custom Build Step

Command line: `moc.exe "$(InputPath)" -o`

`"$(InputDir)moc_$(InputName).cpp"`

Description: MOC `$(InputName)`

Outputs: `$(InputDir)moc_$(InputName).cpp`



Ausgabe: Name der Ausgabedatei

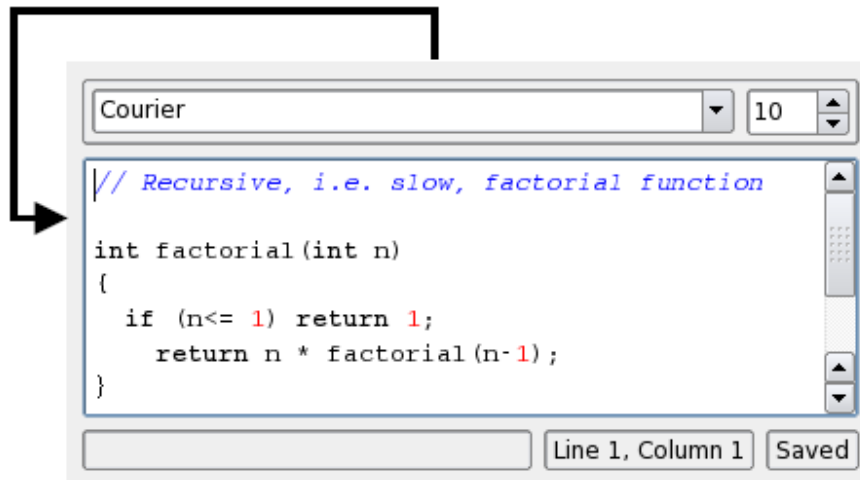
Beschreibung: Das wird im Output-Fenster während des Kompilierens ausgegeben

Die Kommandozeile – mit der Zeile wird aus der Datei `meineDatei.h` zu `moc_meineDatei.cpp` übersetzt

moc.exe bitte mit dem korrekten Pfad angeben!

Slots and Signals – Beispiel

```
connect(fontFamilyComboBox, activated(const QString &),
        textEdit, setFontFamily(const QString &))
```



```
connect(fontSizeSpinBox, valueChanged(int),
        textEdit, setFontSize(int))
```

Geht so nicht, da setFontSize
qreal u. nicht int braucht!

```
connect(textEdit, modificationChanged(bool),
        customStatusBar, modificationStatus(bool))
```

Tutorials und Referenzen

- Reference Manual
<http://doc.trolltech.com/4.1/index.html>
- Whitepaper
<http://www.trolltech.com/pdf/whitepapers/qt40-whitepaper-a4.pdf>