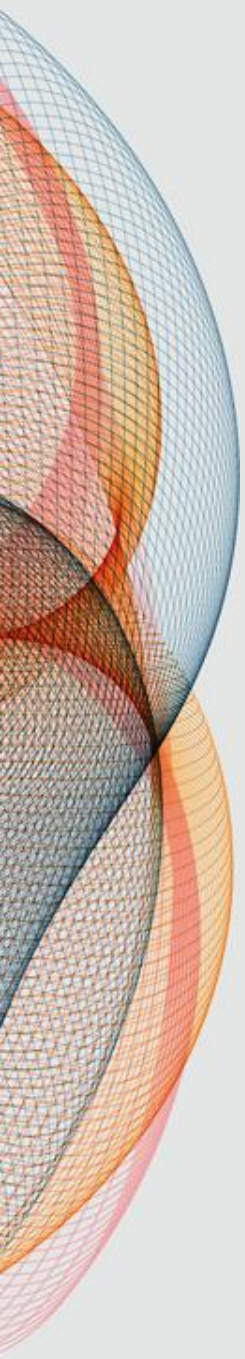
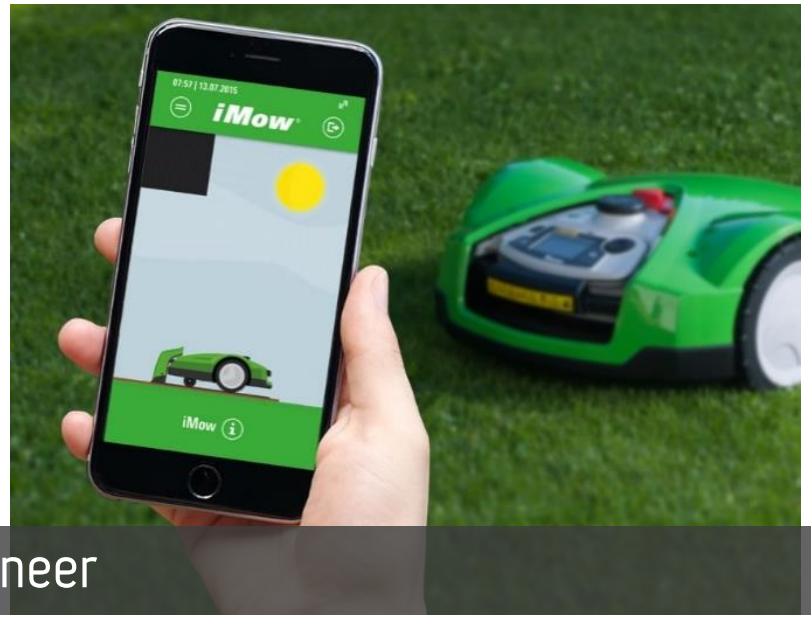


hello, technology

Connect both Qt World: QML and C++

Michael Bätzner

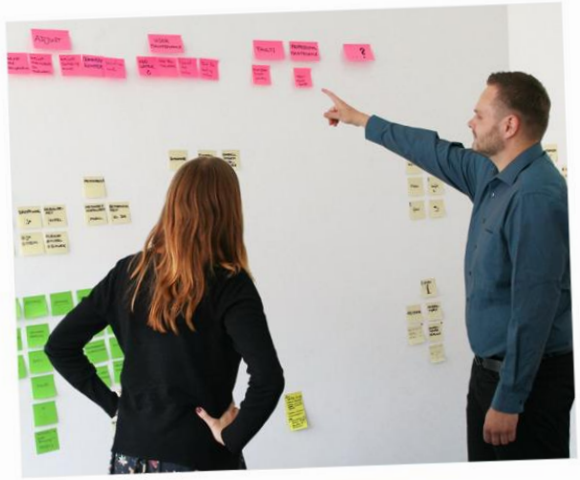
14.02.2017 | v1.0 | öffentlich



Michael Bätzner, Software Engineer

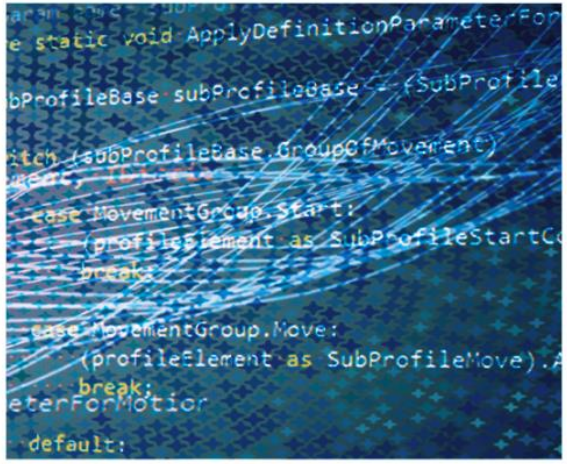
- Seit 2013 bei UID
- Spezialist für Cross-Plattform-Anwendungen mit dem Qt Framework
- Schwerpunkte: Entwicklung von User Interfaces mit QML und Anbindungen an Backends
- Branchen: Automotive, Industry und Enterprise
- Speaker auf Fachkonferenzen

UID IST …



… ein Usability-Unternehmen

… eine Designagentur



… ein Software-Haus

… eine User-Research-Company

… ein Full-Service-Dienstleister rund um User Experience Design.

AGENDA

Erläuterung

Qt

Aufbau

QObject

Meta-Object System

Signals / Slots

Q_PROPERTY

Q_INVOKABLE

Q_ENUM

Aufbau

QML

Erweitern

Typen

Verbindung C++ / QML

Praktisches Beispiel

Agenda


QT

- Cross Plattform C++ (GUI) Framework
  Windows (inkl. UWP), Linux, MacOS, Android, iOS
- Aktuelle Version 5.10 (Dezember 2017)
- Lizenzierung:
  Commercial, LGPL v3, GPL v3

- IDEs
  - » Qt Creator
  - » Visual Studio über Add-in
- Buildsysteme
  - » qmake
  - » cmake

Erläuterung

Qt Add-Ons

Android Extras	Bluetooth	Serial Bus
Quick Controls 2	Positioning	WebEngine

Qt Essentials

Core	GUI	Multimedia
Network	QML	Quick
Quick Controls	Quick Dialogs	Quick Layouts
SQL	Test	Widgets

Qt Modules

META-OBJECT SYSTEM

Basiert auf folgenden Komponenten

- QObject Basisklasse
- Q_OBJECT Macro
- Meta-Object Compiler

Ermöglicht

- Signals/Slots Connections
- Reflection
- Dynamische Properties
- Internationalisierung

- Basisklasse aller Qt Objekte
- Ermöglicht Signal / Slot Kommunikation
- Ermöglicht Objektbaum
- Ermöglicht Speicher-Management
- nicht kopierbar!

QObject

```
#ifndef FOO_H
#define FOO_H

#include <QObject>

class Foo : public QObject
{
	Q_OBJECT
public:
	explicit Foo(QObject *parent = 0);

signals:

public slots:

};
#endif // FOO_H
```

QObject

- Objekt Kommunikation
- Alternative zu Callbacks (ca. 10x langsamer)
- Typensicher
- Ermöglicht lose Kopplung

Voraussetzung

Die Klasse muss

- von QObject ableiten
- das Q_OBJECT Macro enthalten

Signals

- werden im **signals** Bereich der Header-Datei definiert
- werden nicht implementiert
- sind immer public
- können **keinen** Rückgabewert haben
- können Parameter enthalten

Slots

- werden im **private slots** oder im **public slots** Bereich der Header-Datei definiert
- sind ansonsten normale C++ Funktionen

```
#include <QObject>
class Counter : public QObject
{
        Q_OBJECT
public:
        Counter() { m_value = 0; }
        int value() const { return m_value; }

public slots:
        void setValue(int value) {
                if (value != m_value) {
                        m_value = value;
                        emit valueChanged(value);
                }
        }

signals:
        void valueChanged(int newValue);

private:
        int m_value;
};
```

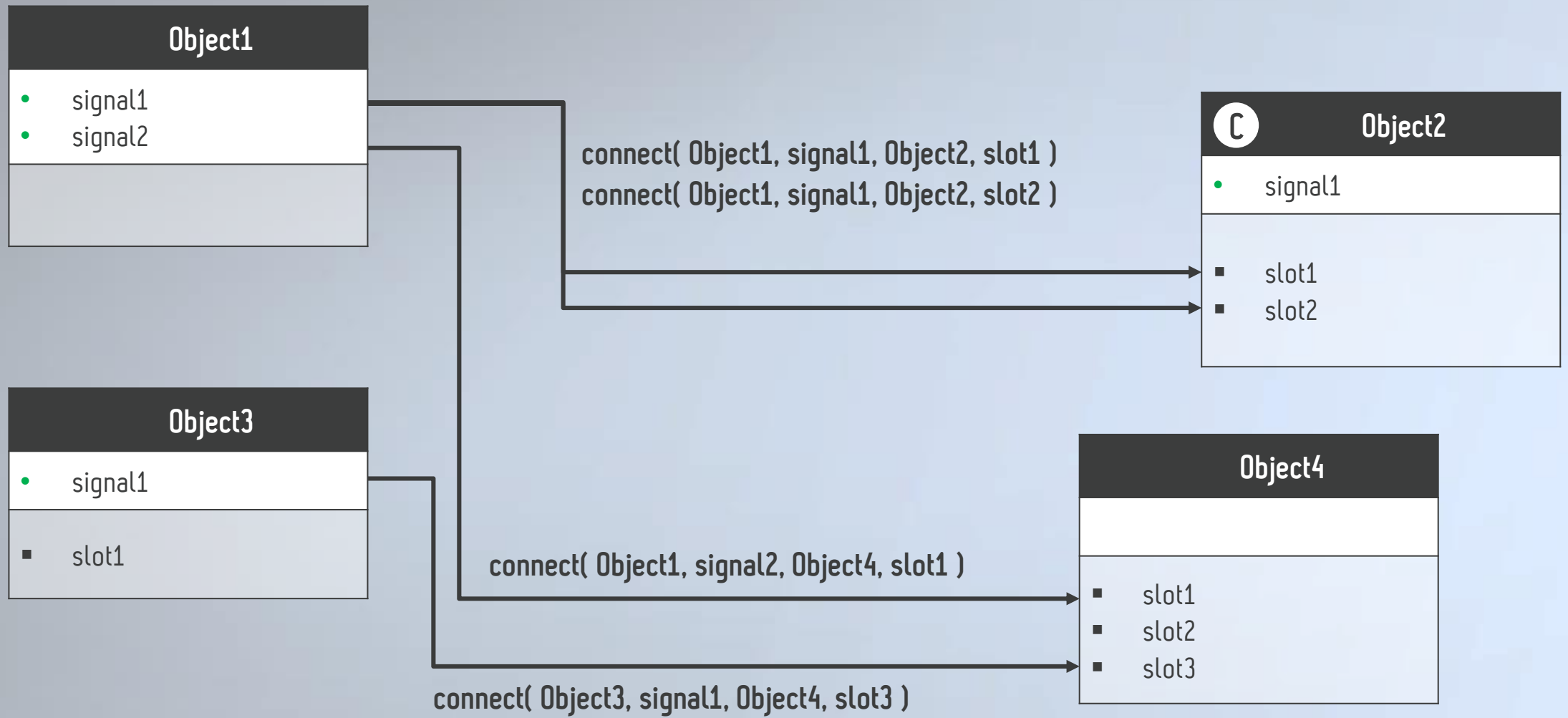
Signal / Slots

```
Counter a, b;

QObject::connect(&a, &Counter::valueChanged, &b, &Counter::setValue);
//C++ 11 Lambda Function
QObject::connect(&a , &Counter::valueChanged,
        [=]( const int &newValue ) { qDebug() << newValue;}
);


//Alternative Schreibweise
QObject::connect(&a, SIGNAL(valueChanged(int)), &b, SLOT(setValue(int)));


a.setValue(12); // a.value() == 12, b.value() == 12
b.setValue(48); // a.value() == 12, b.value() == 48
```



Signal / Slots

- werden direkt ausgeführt
  - » Ausnahme: `Qt::QueuedConnection`
- können über QObject::disconnect() getrennt werden
- jeder connect() wird ausgeführt (Duplikat möglich!)
  - » Ausnahme: `Qt::UniqueConnection`
- auch möglich Signale weiterzuleiten

```
QObject::connect(&a, &Counter::valueChanged,
                 &b, &Counter::valueChanged);
```

- definiert Properties im Meta-Object System
- Nur Datentypen möglich, die auch QVariant unterstützen
- nutzt Signals/Slots

Q_PROPERTY

```
class Foo : public QObject
{
        Q_OBJECT
        Q_PROPERTY(int priority READ priority WRITE setPriority NOTIFY priorityChanged)
        Q_PROPERTY(QString Name READ Name CONSTANT)

public:
        Foo(QObject *parent = 0);

        void setPriority(int priority);
        int priority() const;
        …

signals:
        void priorityChanged(int priority);
        …
```

Q_PROPERTY

```
Foo bar;

bar.setProperty("priority", 99);
qDebug() << bar.property("priority").toInt();
```

Ermöglicht Funktionen aus dem Meta-Object System mit dem Namen aufzurufen

Q_INVOKABLE

```
class Foo : public QObject
{
        Q_OBJECT
        ...
public:
        Foo(QObject *parent = 0);

        Q_INVOKABLE void escalatePriority();
        …

-------------------------------------------------------------------------

Foo bar;

QMetaObject::invokeMethod(&bar, "escalatePriority");
```

Q_INVOKABLE

- Ermöglicht die Verwendung einer Enumeration im Meta-Object System
  - » Properties
  - » Parameter für Signal/Slots
- Ermöglicht die Verwendung von QMetaEnum
  - » Enum per Name

Q_ENUM

```
class Foo : public QObject
{
    Q_OBJECT
    Q_PROPERTY(Priority priority READ priority WRITE setPriority NOTIFY priorityChanged)

public:
    Foo(QObject *parent = 0);

    enum Priority {
       High,
       Low,
       VeryHigh,
       VeryLow
    };
    Q_ENUM(Priority)

    void setPriority(Priority priority);
    Priority priority() const;
```

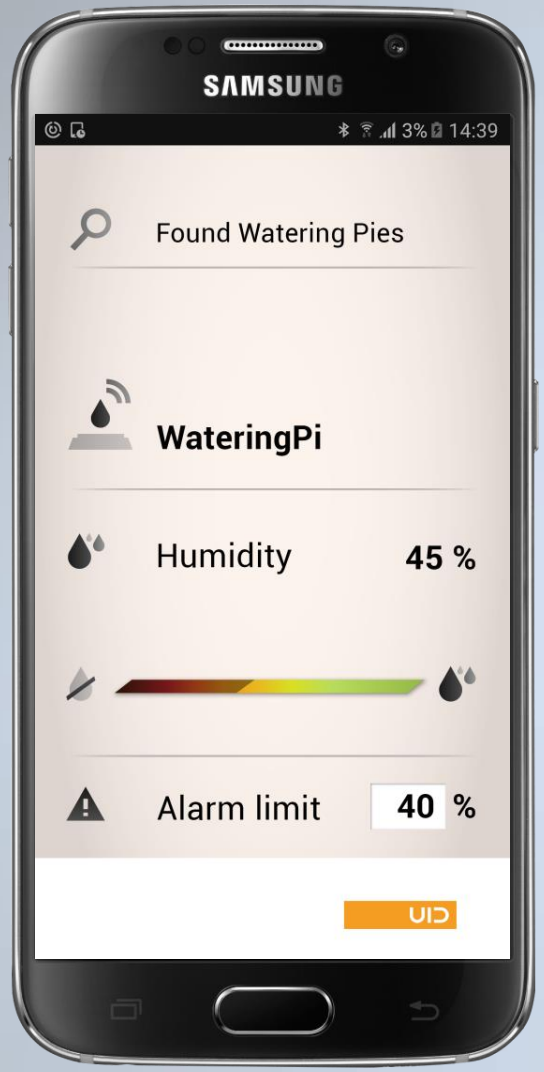
Q_ENUM

```
QMetaEnum metaEnum = QMetaEnum::fromType<Foo::Priority>();

qDebug() << metaEnum.valueToKey(Foo::Priority::VeryHigh);
Foo::Priority prio = metaEnum.keyToValue("VeryHigh");
```

Q_ENUM

QML



- Teil des Qt Quick Moduls
- Deklarative UI Sprache
- JSON ähnliche Syntax
  - » Einfach lesbar
- JavaScript ist möglich
- Entwickelt mit Fokus auf
  - » Touch
  - » Animationen
- Hoch dynamisch durch Property-Bindings

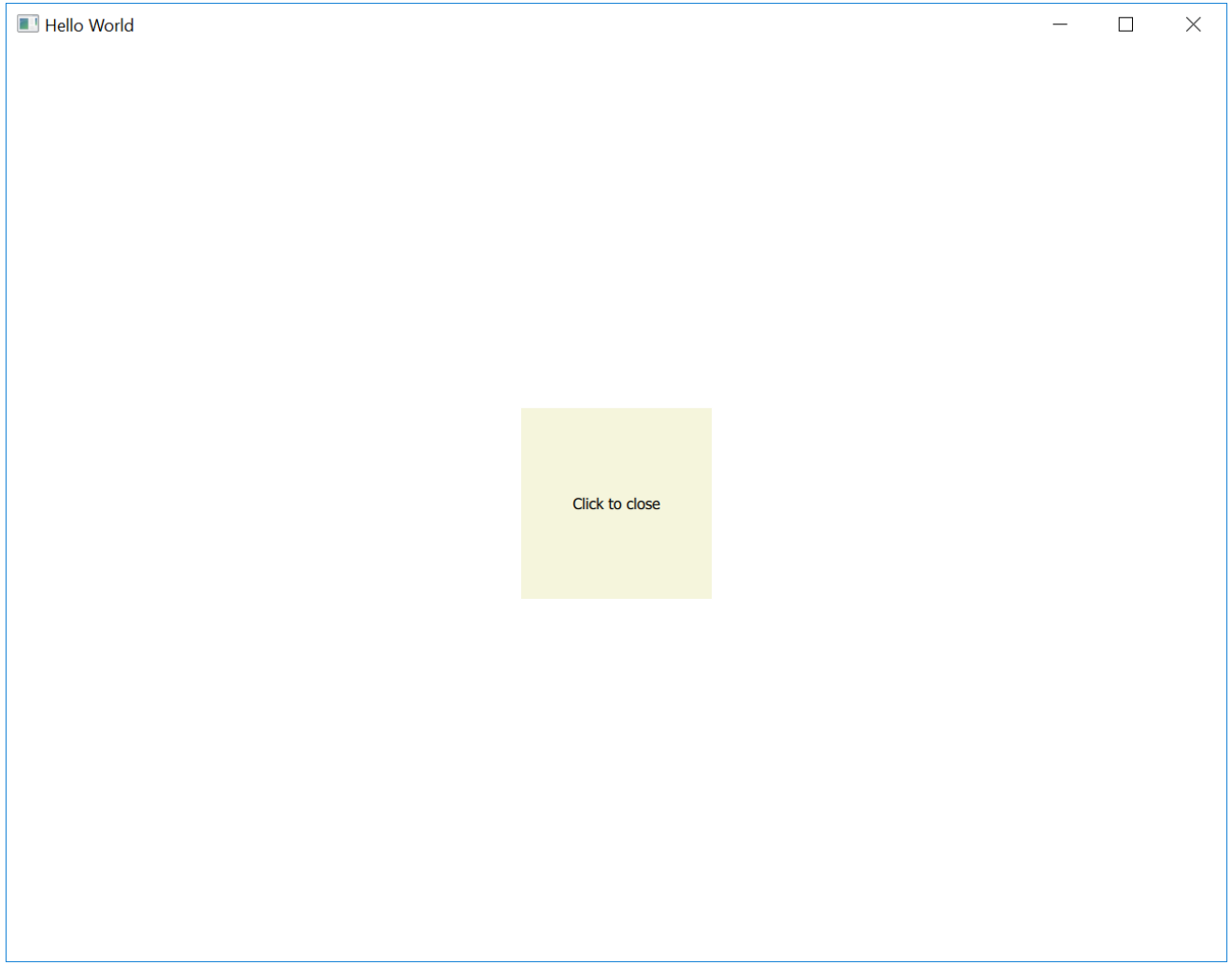
main.qml

```
import QtQuick 2.9
import QtQuick.Window 2.2

Window {
        visible: true
        width: 640; height: 480
        title: qsTr("Hello World")

        Item{
                height: 100; width: 100
                anchors.centerIn: parent

                Rectangle{
                        anchors.fill: parent
                        color: "beige"
                }
                Text {
                        text: qsTr("Click to close")
                        anchors.fill: parent
                        verticalAlignment: Text.AlignVCenter
                        horizontalAlignment: Text.AlignHCenter
                }
                MouseArea {
                        anchors.fill: parent
                        onClicked: { Qt.quit(); }
                }
        }
}
```


Hello World
Click to close

```
import QtQuick 2.9
import QtQuick.Window 2.2

Window {
        visible: true
        width: 640; height: 480
        title: qsTr("Hello World")

        Item{
                height: 100; width: 100
                anchors.centerIn: parent

                Rectangle{
                        anchors.fill: parent
                        color: "beige"
                }
                Text {
                        text: qsTr("Click to close")
                        anchors.fill: parent
                        verticalAlignment: Text.AlignVCenter
                        horizontalAlignment: Text.AlignHCenter
                }
                MouseArea {
                        anchors.fill: parent
                        onClicked: { Qt.quit(); }
                }
        }
}
```

– Import der benötigten Module

```
import QtQuick 2.9
import QtQuick.Window 2.2

Window {
        visible: true
        width: 640; height: 480
        title: qsTr("Hello World")

        Item{
                height: 100; width: 100
                anchors.centerIn: parent

                Rectangle{
                        anchors.fill: parent
                        color: "beige"
                }
                Text {
                        text: qsTr("Click to close")
                        anchors.fill: parent
                        verticalAlignment: Text.AlignVCenter
                        horizontalAlignment: Text.AlignHCenter
                }
                MouseArea {
                        anchors.fill: parent
                        onClicked: { Qt.quit(); }
                }
        }
}
```

- Pro QML Datei nur ein Root-Objekt möglich
- Objekte können meist mehrere Kinder erhalten

```
import QtQuick 2.9
import QtQuick.Window 2.2

Window {
        visible: true
        width: 640; height: 480
        title: qsTr("Hello World")

        Item{
                height: 100; width: 100
                anchors.centerIn: parent

                Rectangle{
                        anchors.fill: parent
                        color: "beige"
                }
                Text {
                        text: qsTr("Click to close")
                        anchors.fill: parent
                        verticalAlignment: Text.AlignVCenter
                        horizontalAlignment: Text.AlignHCenter
                }
                MouseArea {
                        anchors.fill: parent
                        onClicked: { Qt.quit(); }
                }
        }
}
```

Item

- Basis fast aller visuellen und nicht visuellen QML Typen
- Nicht visualisiert
- Zentrale Eigenschaften
  - » Höhe, Breite
  - » Fixe Positionierung (x,y,z)
  - » Anker Positionierung (anchors)
  - » …

```
import QtQuick 2.9
import QtQuick.Window 2.2

Window {
        visible: true
        width: 640; height: 480
        title: qsTr("Hello World")

        Item{
                height: 100; width: 100
                anchors.centerIn: parent

                Rectangle{
                        anchors.fill: parent
                        color: "beige"
                }
                Text {
                        text: qsTr("Click to close")
                        anchors.fill: parent
                        verticalAlignment: Text.AlignVCenter
                        horizontalAlignment: Text.AlignHCenter
                }
                MouseArea {
                        anchors.fill: parent
                        onClicked: { Qt.quit(); }
                }
        }
}
```

Rectangle

- Abgeleitet von Item
- Rechteck mit Füllung
- Spezielle Eigenschaften
  - color
  - gradient
  - border
  - radius

```
import QtQuick 2.9
import QtQuick.Window 2.2

Window {
        visible: true
        width: 640; height: 480
        title: qsTr("Hello World")

        Item{
                height: 100; width: 100
                anchors.centerIn: parent

                Rectangle{
                        anchors.fill: parent
                        color: "beige"
                }
                Text {
                        text: qsTr("Click to close")
                        anchors.fill: parent
                        verticalAlignment: Text.AlignVCenter
                        horizontalAlignment: Text.AlignHCenter
                }
                MouseArea {
                        anchors.fill: parent
                        onClicked: { Qt.quit(); }
                }
        }
}
```

Text

- Abgeleitet von Item
- Visualisierung von Text
- Spezielle Eigenschaften
  - » text
  - » verticalAlignment / horizontalAlignment
  - » font.pixelzise
  - » ….

```
import QtQuick 2.9
import QtQuick.Window 2.2

Window {
        visible: true
        width: 640; height: 480
        title: qsTr("Hello World")

        Item{
                height: 100; width: 100
                anchors.centerIn: parent

                Rectangle{
                        anchors.fill: parent
                        color: "beige"
                }
                Text {
                        text: qsTr("Click to close")
                        anchors.fill: parent
                        verticalAlignment: Text.AlignVCenter
                        horizontalAlignment: Text.AlignHCenter
                }
                MouseArea {
                        anchors.fill: parent
                        onClicked: { Qt.quit(); }
                }
        }
}
```

MouseArea

- Abgeleitet von Item
- Keine Visualisierung
- Fläche für Maus-Events
  - » clicked
  - » doubleClicked
  - » …

```qml
import QtQuick 2.9
import QtQuick.Window 2.2

Window {
        visible: true
        width: 640; height: 480
        title: qsTr("Hello World")

        Item{
                height: 100; width: 100
                anchors.centerIn: parent

                Rectangle{
                        anchors.fill: parent
                        color: "beige"
                }
                Text {
                        text: qsTr("Click to close")
                        anchors.fill: parent
                        verticalAlignment: Text.AlignVCenter
                        horizontalAlignment: Text.AlignHCenter
                }
                MouseArea {
                        anchors.fill: parent
                        onClicked: { Qt.quit(); }
                }
        }
}
```

Window

- Fenster
- Benötigt import
- Spezielle Eigenschaften
  - » title
  - » modalitiy
  - » maximumWidth / maximumHeight
  - » ....

Weitere QML Typen u.a.

- Images
- Controls (natives Look & Feel in Version 1)
- Layouts (Column, ColumnLayout, …)
- ListViews, GridViews,…

```
Item{
        id: myItem

        //Item Objekt mit weiteren Properties.
        property string name: "Max Mustermann"
        property int age: 30
        property bool isRegistered: true

        Text {
                text: myItem.name
        }

        onNameChanged: {
                //Slot
                console.log("Neuer Name ist " + name)
        }
}
```

Properties

- Weitere Eigenschaften können definiert werden
- Binding über ID und Eigenschaft des Objekts
- Changed-Signal automatisch für alle Eigenschaften vorhanden: `on...Changed()`

EINFACHE EINBINDUNG EIGENER QML TYPEN

main.qml

```
import QtQuick 2.9
import QtQuick.Window 2.2

Window {
        visible: true
        width: 640; height: 480
        title: qsTr("Hello World")

        MyButton{
                height: 100; width: 100
                anchors.centerIn: parent
        }
}
```

MyButton.qml

```
import QtQuick 2.9

Item{
        Rectangle{
                anchors.fill: parent
                color: "beige"
        }
        Text {
                id: textEdit
                text: qsTr("Click to close")
                anchors.fill: parent
                verticalAlignment: Text.AlignVCenter
                horizontalAlignment: Text.AlignHCenter
        }
        MouseArea {
                anchors.fill: parent
                onClicked: { Qt.quit(); }
        }
}
```

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>

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

#if defined(Q_OS_WIN)
       QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
#endif

QGuiApplication app(argc, argv);
QQmlApplicationEngine engine;
engine.load(QUrl(QStringLiteral("qrc:/main.qml")));

if (engine.rootObjects().isEmpty())
       return -1;

return app.exec(); }
```

VERBINDUNG C++ UND QML

Voraussetzung:

C++ Klasse muss das Meta-Object System unterstützen

- Ableitung von QObject
- Macro Q_OBJECT

```
class Foo : public QObject
{
        Q_OBJECT
        Q_PROPERTY(int priority READ priority WRITE setPriority NOTIFY priorityChanged)
        Q_PROPERTY(QString name READ WRITE setName NOTIFY nameChanged)

public:
        Foo(QObject *parent = 0);

        void setPriority(int priority);
        int priority() const;
        Q_INVOKABLE void escalatePriority();
        …

signals:
        void priorityChanged(int priority);
```

3 Wege

1. Vorhandenes Objekt dem Root-Kontext als Eigenschaft hinzufügen
2. Klasse im QML Kontext registrieren und Objekt im QML erzeugen
3. Objekt über Q_PROPERTY an QML übergeben

1. Weg - Vorhandenes Objekt dem Root-Kontext als Eigenschaft hinzufügen

main.cpp

```cpp
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include <QQmlContext>
#include "foo.h"

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

        QGuiApplication app(argc, argv);

        Foo bar;
        bar.setName("Hallo Welt");

        QQmlApplicationEngine engine;
        engine.rootContext()->setContextProperty("bar", &bar);
        engine.load(QUrl(QStringLiteral("qrc:/main.qml")));

        return app.exec();
}
```

QML

```
Item {
    Text {
     text: bar.name
     anchors.fill: parent
    }

    MouseArea {
     …
     onClicked: bar.escalatePriority()
    }
}
```

2. Weg - Klasse im QML Kontext registrieren und Objekt im QML erzeugen

main.cpp

```cpp
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include <QQmlContext>
#include "foo.h"

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

        QGuiApplication app(argc, argv);

        qmlRegisterType<Foo>("myNamespace", 1, 0, "Foo");

        QQmlApplicationEngine engine;
        engine.load(QUrl(
                QStringLiteral("qrc:/main.qml")));

        return app.exec();
}
```

QML

```
import myNamespace 1.0

Item {
  Foo {
      id: bar
      name: "Hallo Welt"
  }

  Text {
      text: bar.name
      anchors.fill: parent
  }

  MouseArea {
      …
      onClicked: bar.escalatePriority()
  }
}
```

3. Weg - Objekt über Q_PROPERTY an QML übergeben

MyObject.h

```
class MyObject : public QObject
{
        Q_OBJECT
        Q_PROPERTY(Foo* bar READ bar WRITE setBar
                                    NOTIFY barChanged)

public:
        MyObject(QObject *parent = 0);
…
```

QML

```
Item {
  Text {
      text: myObject.bar.name
      anchors.fill: parent
  }

  MouseArea {
      …
      onClicked:
        myObject.bar.escalatePriority()
  }
}
```

Properties

- Lesen

```
text: IDdesObjekts.PropertyName
```

- Schreiben

```
onClicked: IDdesObjekts.PropertyName = neuer Wert
```

C++ Funktionen

- Nur SLOTS oder Q_INVOKABLE Funktionen

```
onClicked: IDdesObjekts.Function()
```

ACHTUNG

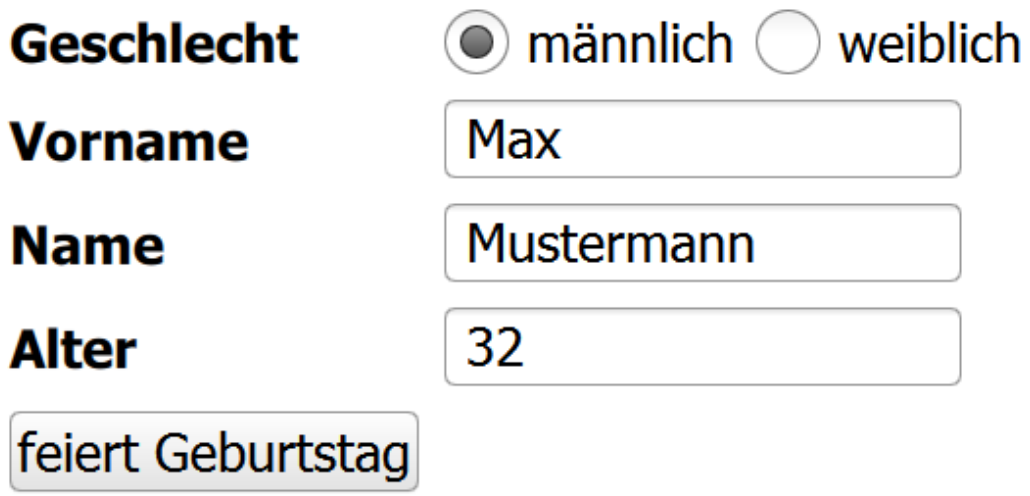
Wenn eine Q_INVOKABLE Funktion ein Objekt zurückgibt, übernimmt der QML Context die Ownership des Objektes.
Dadurch kann es sein, dass das Objekt durch den QML Garbage-Collector gelöscht wird

PRAKTISCHES BEISPIEL

```
class Person {

public:
	Person();

	enum Geschlecht{ Unbekannt, Maennlich, Weiblich };
	std::string vorname = "";
	std::string name = "";
	int alter = 0;
	Geschlecht geschlecht;
};
```



Ziel

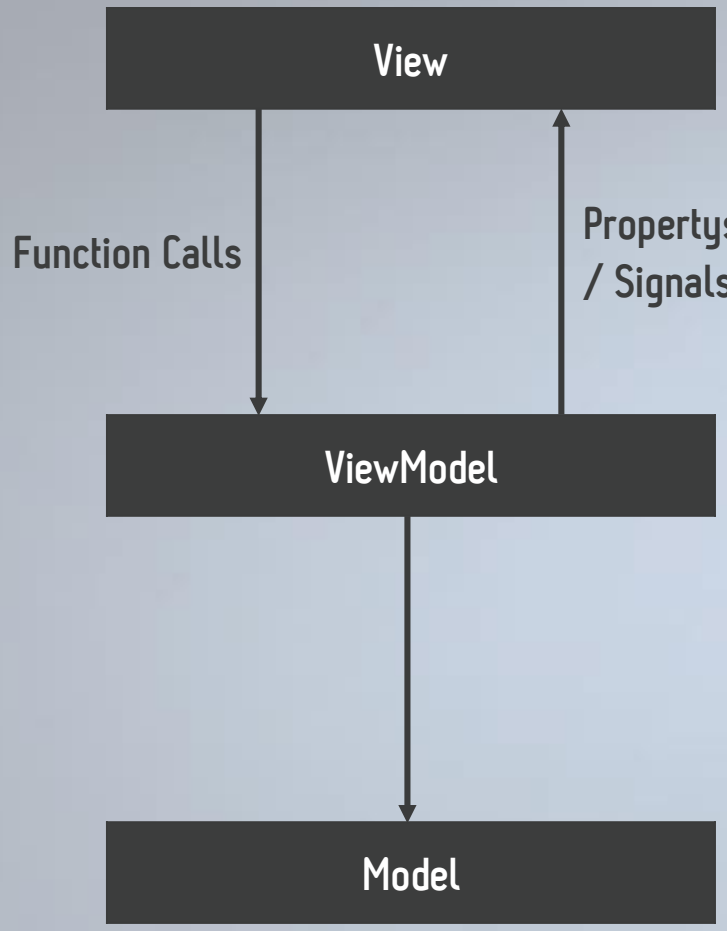
Es ist nicht möglich reine C++
Klassen im QML zu verwenden!

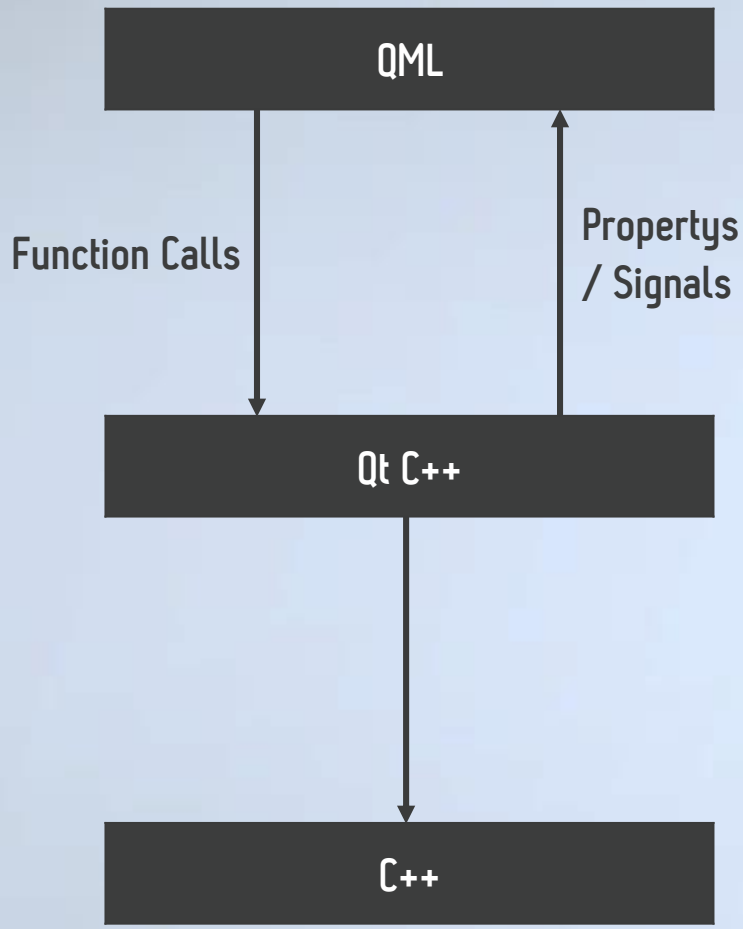
Die Kommunikation zwischen QML und C++
erfolgt über das Meta-Object System.

Einschränkung

Wrapper-Klasse (ViewModel) um die Modelklasse herum

Lösung





MVVM Pattern

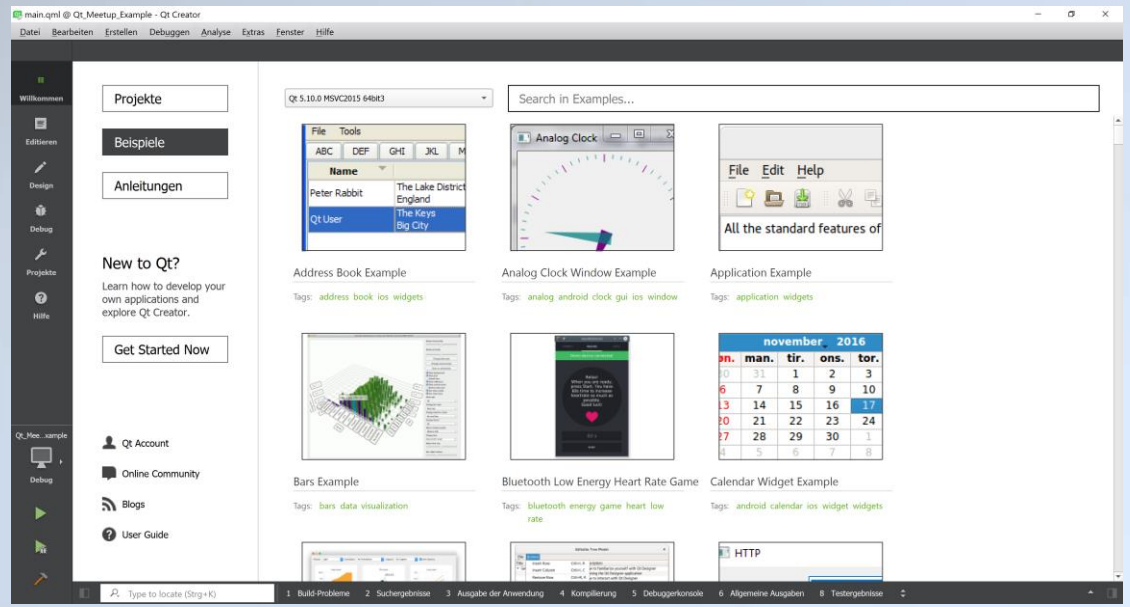
Demo

Demo

TIPPS

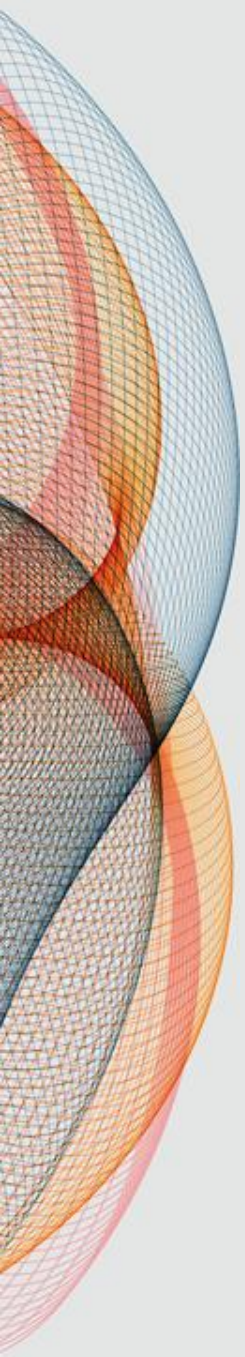
- sehr gute Qt Dokumentation
  http://doc.qt.io/qt-5/

- Examples im Qt Creator



Tipps


FRAGEN?



hello, technology

Michael Bätzner
SOFTWARE ENGINEER

michael.baetzner@uid.com
www.uid.com