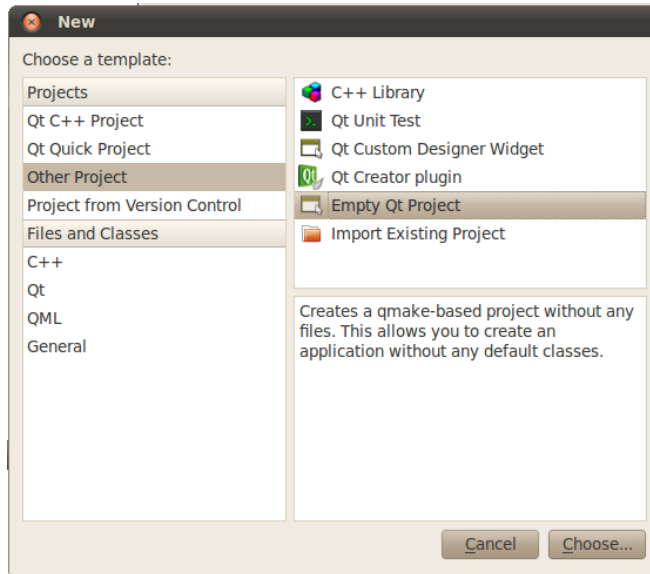


MCM440-Dialer

Project Setup:

Um sowohl GPX- als auch C++Dateien im Qt-Projekt verwenden zu können, habe ich das Projekt als „Empty Qt Project“ angelegt:



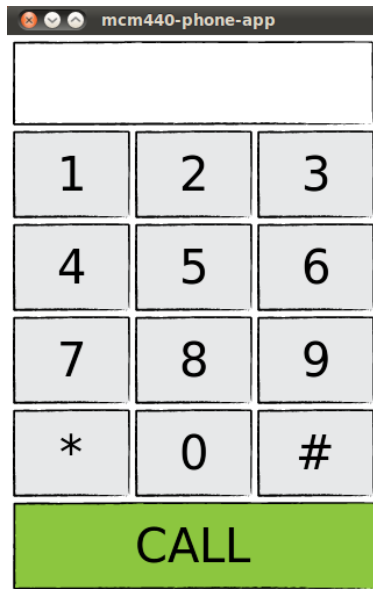
Das einzige, was in diesem Fall zu beachten ist, ist dass die nicht kompilierenden Dateien (qml und sonstige Ressourcen wie Grafiken etc.) sich im gleichen Verzeichnis befinden müssen, wie die Binary-Dateien.

Zur Lösung dieses Problems gibt es folgende 2 Möglichkeiten:

- ▶ Kopieren der entsprechenden Dateien in das Build-Verzeichnis (muss allerdings nach jeder Änderung wiederholt werden)
- ▶ Änderung des Build-Verzeichnisses auf das Projekt-Verzeichnis dh in unserem konkreten Fall den vorgeschlagenen Pfad „.../mcm440-phone-app-build-desktop“ durch den Pfad „.../mcm440-phone-app“ ersetzen

Dialer QML-Ressources:

Der Dialer besteht momentan aus einem Standard-Ziffernblock, einem Display zur Anzeige der getippten Nummer sowie einem Call-Button. Design kommt noch, derzeit sind die Buttons nur skizzenmäßig dargestellt. Außerdem wird noch ein Delete-Button zum entfernen der zuletzt getippten Ziffer hinzugefügt.



Umgesetzt habe ich das in vier QML-Dateien:

- ▶ **Dialer.qml**
Die zentrale Dialer-Datei mit einer Auflösung von 320x480 Pixeln, die das Layout festlegt und die einzelnen Komponenten in sich zusammenfasst. In dieser Datei werden die durchzuführenden Aktionen als Reaktion auf die onClicked-Signals der einzelnen Buttons festgelegt.
- ▶ **DialerDisplay.qml**
Ist die Nummernanzeige am oberen Rand des Dialers bestehend aus einer Image- und einer Text-Komponente.
- ▶ **DialerKey.qml**
Ist eine Zifferntaste des Ziffernblocks bestehend aus einer Image-, einer Text- und einer MouseArea-Komponente. Die Property „text“ legt die hinterlegte Ziffer für die Taste fest.
- ▶ **DialerButton.qml**
Ist der Call-Button am unteren Rand des Dialers bestehend aus einer Image- einer Text- und einer MouseArea-Komponente.

Verbindung von QML und C++

Um Methoden von C++-Objekten von QML-Files aus aufrufen zu können, sind folgende Schritte nötig:

in der Projektdatei mcm400-phone-app.pro:

```
QT += declarative
```

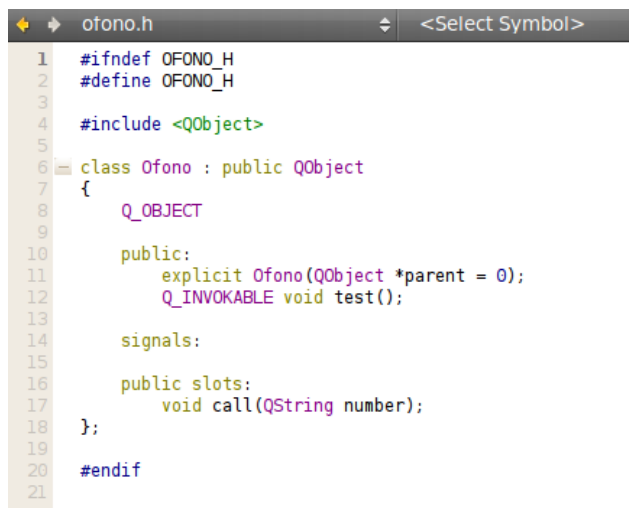
in der main-Methode:

```
QApplication a(argc, argv);
QDeclarativeView view;
view.rootContext()->setContextProperty("OfonoContext", new Ofono());
view.setSource(QUrl::fromLocalFile("Dialer.qml"));
view.show();
```

Wichtigster Teil dabei ist der Aufruf der setContextProperty-Methode. Sie legt fest, welche Objekte in den QML-Ressourcen zur Verfügung stehen. Ein anderes Beispiel könnte etwa sein:

```
setContextProperty(„backgroundColor“, QColor::black);
```

Werden an dieser Stelle eigene Objekttypen (ich habe hier ein Ofono-Objekt verwendet) übergeben, müssen diese von QObject abgeleitet sein und aufzurufende Signals/Slots definieren.



```
1  #ifndef OFONO_H
2  #define OFONO_H
3
4  #include <QObject>
5
6  class Ofono : public QObject
7  {
8      Q_OBJECT
9
10     public:
11         explicit Ofono(QObject *parent = 0);
12         Q_INVOKABLE void test();
13
14     signals:
15
16     public slots:
17         void call(QString number);
18 };
19
20 #endif
21
```

Es gibt zwei Möglichkeiten zur Definition von Slots:

- ▶ im public slots-Abschnitt der Klasse (zB void call(QString number))
- ▶ als Q_INVOKABLE im public-Abschnitt der Klasse (zB void test());

Signals werden im signal-Abschnitt der Klasse definiert.

Die definierten Signals/Slots können dann wie im ContextProperty festgelegt, von QML aus aufgerufen werden.

Hier ein Ausschnitt dazu aus der Dialer.qml-Datei:

```
115
116     DialerButton {
117         id: callbutton
118         x: collororigin
119         y: 402
120         onClicked: {OfonoContext.test();
121                     OfonoContext.call(display.text);}
122     }
123 }
124
```

Die Methoden, die ich in der Ofono-Klasse definiert habe ich vor allem zu Testzwecken erstellt, wie die Signals/Slots der Ofono-Klasse schlussendlich aussehen, hängt davon ab was die D-Bus Komponente damit macht. Wahrscheinlich wäre es auch sauberer, wenn die zu wählende Nummer von der Ofono-Klasse selbst verwaltet wird (und jede Ziffer im onClick einen entsprechenden Setter aufruft), anstatt den Text (so wie jetzt) im text-Property des DialerDisplays zu speichern.

Wie die Schnittstelle schlussendlich konkret aussieht, überlasse ich natürlich euch ;)