|  |
| --- |
| Shacira GUI Framework mit QT 4 |
| Analyse |

Version: 1.1

Datum: 3. Juli 2012

Projekt: intern   
Autor: Peter Ritzmann (2i)

Datei: ShaciraQT4.docx   
Status: vorläufig

[1 Anforderungen 3](#_Toc328985757)

[2 Analyse 4](#_Toc328985758)

[2.1 Qt 4 API Methoden, Typen und Klassen 4](#_Toc328985759)

[2.2 Qt 3 Widget properties. 4](#_Toc328985760)

[2.3 Konzeptuelle Änderungen mit Qt 4 4](#_Toc328985761)

[2.4 Überflüssige Mechanismen 4](#_Toc328985762)

[3 Umsetzungskonzept 5](#_Toc328985763)

[3.1 Qt 4 API Methoden, Typen und Klassen 5](#_Toc328985764)

[3.2 Qt 3 Widget properties. 5](#_Toc328985765)

[3.3 Konzeptuelle Änderungen mit Qt 4 5](#_Toc328985766)

[4 Änderungsaufwand für Mosaic 6](#_Toc328985767)

[4.1 Qt 4 API Methoden, Typen und Klassen 6](#_Toc328985768)

[4.2 Qt 3 Widget properties. 6](#_Toc328985769)

[4.3 Konzeptuelle Änderungen mit Qt 4 6](#_Toc328985770)

[5 Komponenten 7](#_Toc328985771)

[5.1 q3\_cwidget\_plugin Plugin 7](#_Toc328985772)

[5.2 q3\_keypad\_plugin Plugin 7](#_Toc328985773)

[5.3 q3\_kbdlayout\_plugin Plugin 7](#_Toc328985774)

[5.4 q3\_framework Bibliothek 7](#_Toc328985775)

[5.5 uiport Konvertierungswerkzeug 7](#_Toc328985776)

[6 Planung / geschätzter Aufwand 8](#_Toc328985777)

# Anforderungen

Das Shacira GUI Framework ist aktuell mit der Oberflächenbibliothek Qt Version 3 realisiert. Diese Implementierung soll auf Qt Version 4 neu implementiert werden.

Die Neuimplementierung unter Qt 4 ist mit weitreichenden Konsequenzen verbunden, da Qt 4 sich im API und teilweise auch in den Konzepten stark von Qt 3 unterscheidet.

Eine Möglichkeit, den für die Implementierung unter Qt 4 notwendigen Entwicklungsprozess kontrollierbar zu gestalten ist zunächst die Umsetzung auf Qt 4 in den entsprechenden Shacira Bibliotheken und Plugins. Diese Umsetzung soll neben der Portierung der eigentlichen Funktionalität der Bibliotheken auch noch um Code erweitert werden, der die Anpassung der Qt-Standard Widgets implementiert. Qt 4 besitzt für diese Zwecke zwar eine Qt3Support – Layer genannte Softwareschicht. Dieser Software-Teil in Qt 4 ist aber nur temporär zu sehen und wird in späteren Versionen (Qt 5) möglicherweise entfallen.

Ziel bei der Portierung von Qt3 auf Qt4 ist, dass das API für Anwendungscode soweit wie möglich unverändert bleibt, um dort den Integrationsaufwand niedrig zu halten.

Dasselbe gilt für die Organisation des Codes (Dateinamen, Verzeichnisstruktur usw.) und für den Einsatz von Werkzeugen (Präprozessor für Shacira Widget properties).

# Analyse

## Qt 4 API Methoden, Typen und Klassen

Mit der Präprozessor-Definition QT3\_SUPPORT bietet Qt 4 ein zu Qt 3 kompatibles API an. Dieser Teil des API deckt jedoch nicht alle Bereiche der Qt 3 Klassen ab. Diese Kompatibilitäts-Schicht von Qt4 wird im Shacira-GUI-Framework für Qt 4 nicht verwendet.

Teilweise sind das nur Änderungen im Namensraum, teilweise bietet Qt4 eine Anpassung über die Compiler-Konstante QT3\_SUPPORT.

Die Verwendung dieser Klassen wird in der Qt 4 Dokumentation nur als temporäre Lösung im Portierungsprozess empfohlen. Vom Einsatz dieser Klassen, Methoden und Typen in einem produktiven System wird abgeraten. Hinzu kommt das diese Kompatibilitätsschicht in der nächsten Qt Haupt-Version entfällt.

* QIcon::Size
* WFlags
* setWFlags

## Qt 3 Widget properties.

Einige Qt 3 spezifische Widget properties werden Qt 4 gar nicht oder mit veränderter Semantik verwendet werden.

QWidget

* paletteBackgroundColor
* paletteForegroundColor
* paletteBackgroundPixmap

QFrame, QGroupBox

* frameShape
* frameShadow

QTextedit, QLineEdit

* alignment

QToolButton und QPushButton

* iconSet
* usesBigPixmap

## Konzeptuelle Änderungen mit Qt 4

### Manipulation der Grafikausgabe

Grafik-Manipulationen dürfen unter Qt 4 nur innerhalb der event-Methode ::paintEvent(..) erfolgen, Alle Manipulationen innerhalb eines Widgets (zum Beispiel unter Verwendung des Objekts QPainter) müssen innerhalb der paintEvent –Methode des Widgets implementiert bzw aufgerufen werden.

### Verwaltung von Bildinformationen

Die Verwaltung von Bildinformationen erfolgt unter Qt4 generell über Qt Resource-Dateien (.qrc). Das Speichern von Bildinformationen innerhalb einer ui-Datei ist nicht mehr möglich. Die in den Qt 3 ui-Dateien gespeicherten Bildinformationen werden mit einem zusätzlichen Werkzeug extrahiert und in einer Qt 4 spezifischen globalen Resource abgelegt. Diese Resource kann in der Weiterentwicklung der Qt 4 Anwendung (mosaic) erweitert und verändert werden.

## Überflüssige Mechanismen

### View-Mechanismus

Der View-Mechanismus der CWidgets, der unter Qt 3 die Grafikausgabe thread safe gemacht hat, ist mit den in 2.2.1 beschriebenen Änderungen überflüssig und kann entfallen weil damit jede Grafikausgabe per se thread safe ist.

# Umsetzungskonzept

## Qt 4 API Methoden, Typen und Klassen

Alle Widget Klassen, die unter Qt 4 nicht mehr unterstützt werden, werden auf die vorhandenen entsprechenden CWidgets oder notwendigenfalls auf neu zu implementierende CWidgets abgebildet.

Die APIs der CWidgets müssen so erweitert werden, dass sie die nicht vorhandene Kompatibilität zu Qt 3 realisieren.

Dort, wo es möglich ist werden Anpassungen durch eine projektweite Ersetzung der Namen realisiert

* QIcon::Size
* WFlags -> Qt::WFlags
* setWFlags -> setWindowFlags

Änderungen, die den Quellcode und nicht die ui-Dateien betreffen werden mit einem entsprechend konfigurierbaren Textersetzungssystem über das ganze Projekt ersetzt.

## Qt 3 Widget properties.

Die genannten properties werden in den abgebildeten CWidgets implementiert. Die Implementierung der korrekten Funktionalität unter Qt 4 für diese properties eröffnet prinzipiell die Möglichkeit der Verwendung dieser properties in der Weiterentwicklung der Anwendung. Die weitere Verwndung dieser ist jedoch strikt zu vermeiden, am Ende eines Übergangsprozesses sollten diese properties nicht mehr verwendet werden.

Jedes CWidget wird um Code erweitert, der die Anpassung für das benötigte property-Interface zur Verfügung stellt.

Der zusätzliche Code der in den CWidgets implementiert wird an abgeleitete Widgets vererbt. Er steht allen Widgets zur Verfügung, die von einem CWidget abgeleitet sind und müssen damit in anwendungsspezifischen Plugins für von einem CWidget abgeleiteten Widget nicht mehr implementiert werden.

Die zusätzlichen Code-Teile aus 3.1 und 3.2 werden über die Compiler-Konstante CWIDGET\_QT3\_SUPPORT ein-/ausschaltbar gemacht. In einer Zielversion der Portierung sollte auf diese Anpassungen vollständig verzichtet werden können.

Anpassungen von property-Namen und –inhalten werden in den ui-Dateien vom uiport-Tool durchgeführt.

## Konzeptuelle Änderungen mit Qt 4

### Manipulation der Grafikausgabe

### Verwaltung von Bildinformationen

Die in den Qt 3 ui-Dateien gespeicherten Bildinformationen werden mit einem zusätzlichen Werkzeug extrahiert und in einer Qt 4 spezifischen globalen Resource abgelegt. Diese Resource kann in der Weiterentwicklung der Qt 4 Anwendung (mosaic) erweitert und verändert werden.

# Änderungsaufwand für Mosaic

Die umfangreichsten Anpassungen sind im Mosaic-Widget-Plugin durchzuführen. Hier wird ein hoher Aufwand vor allem bei den Kernzugprogramm-Widgets und bei den Heizungs-Widgets erwartet. Der restliche ist hinsichtlich Anpassungen unkritisch.

## Qt 4 API Methoden, Typen und Klassen

Projektweite Ersetzungen werden automatisiert durchgeführt

## Qt 3 Widget properties.

## Konzeptuelle Änderungen mit Qt 4

### Manipulation der Grafikausgabe

Die Änderung der Grafikausgabe fällt im Wesentlichen in den Dateien der Heizungs-Widgets und der Kernprogramm-Widgets an.

## Mosaic Wizards

Die QWizard Klassen sind in den beiden Qt-Varianten inkompatibel. Hier muss eine CWizard Klasse implementiert werden, die die in Qt 3 nicht mehr unterstützten Teile des API implementiert.

Eine umfangreichere Analyse zur Umsetzung der Mosaic-Wizards war in diesem Zeitrahmen nicht möglich.

## Mosaic Übersetzungen

Die Qt 4 Übersetzungsmechanismen sind gegenüber Qt 3 im wesentlichen unverändert.

# Komponenten

## q3\_cwidget\_plugin Plugin

Es wird eine Qt4-Plugin DLL q4\_cwidget\_plugin.dll erstellt, die funktional identisch ist mit q3\_cwidget\_plugin.dll.

## q3\_keypad\_plugin Plugin

Es wird eine Qt4-Plugin DLL q4\_keypad\_plugin.dll erstellt, die funktional identisch ist mit q3\_keypad\_plugin.dll.

## q3\_kbdlayout\_plugin Plugin

Es wird eine Qt4-Plugin DLL q4\_kbdlayout\_plugin.dll erstellt, die funktional identisch ist mit q3\_kbdlayout\_plugin.dll.

## q3\_framework Bibliothek

Es wird eine Qt4-Bibliothek q4\_framework.lib erstellt, die funktional identisch ist mit q3\_framework.lib.

## uiport Konvertierungswerkzeug

Qt 3 Forms im Qt3 ui Format können mit uic3 in Forms im Qt4 ui-Format konvertiert werden. Uic3 wird von uiport als erster Schritt in der ui-Code-Konvertierung verwendet. In einem nachgeordneten Schritt wird der entstandene dann noch einmal verändert, um die genannten Zielsaetzungen zu erfüllen (uic3 generiert Q3… Klassen, diese Klassen werden dann zum Beispiel durch C…-Klassen ersetzt).

Als Konsequenz entsteht reiner Qt 4-ui-Code ohne Rückgriff auf das Qt3Support-Layer.

Neben der Anpassung der ui-Dateien der ui-Dateien erzeugt uiport die notwendigen Resourcen für die Verwaltung von Bilddaten.

# Planung / geschätzter Aufwand

|  |  |
| --- | --- |
| Aufgabe | Geschätzter Aufwand |
| Implementierung q4\_cwidget\_plugin Plugin | 15 PT |
| Implementierung q4\_kbdlayout\_plugin Plugin | 4 PT |
| Implementierung q4\_keypad\_plugin Plugin | 4 PT |
| Implementierung q4\_framework Bibliothek | 20 PT |
| Implementierung uiport | 3 PT |
| Änderungen im Widget Präprozessor | 1 |
|  |  |
| Dokumentation für die Umsetzung der anwendungsspezifischen Widgets (Mosaic-Widgets) | 5 PT |
| Initiale Umsetzung des Mosaic-Widget-Plugins | 5 |
| Test der Umsetzung und der Plugins anhand aller Mosaic Pages im Designer | 10 PT |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |