Version: 1.3

Datum: 29. April 2013

Projekt: intern   
Autor: Peter Ritzmann (2i)

Datei: ShaciraQT4-process.docx   
Status: vorläufig

|  |
| --- |
| Shacira GUI Framework mit QT 4 |
| Prozessdokumentation Portierung |

Inhalt

[1 C++ -Code-Anpassungen 3](#_Toc354989841)

[1.1 Wrapper-Klassen 3](#_Toc354989842)

[1.1.1 QHeader / HeaderView 3](#_Toc354989843)

[1.1.2 QListView / ListView 3](#_Toc354989844)

[1.1.3 QWizard / Wizard 3](#_Toc354989845)

[1.2 Sonstige-Klassen 4](#_Toc354989846)

[1.2.1 QRangeControl 4](#_Toc354989847)

[1.2.2 QValueVector<T> 4](#_Toc354989848)

[1.2.3 QPtrList<T> 4](#_Toc354989849)

[2 Formulare und Trägerklassen 5](#_Toc354989850)

[2.1 Vereinfachte Ableitung von Formularklassen 6](#_Toc354989851)

[3 Schritte im Portierungsprozess (mosaic) 7](#_Toc354989852)

[3.1 1. Schritt Konvertierung von Projektdateien 8](#_Toc354989853)

[3.2 Property sheets 8](#_Toc354989854)

[3.3 Zusätzliche include Pfade 8](#_Toc354989855)

[3.4 weitere Änderungen 9](#_Toc354989856)

[3.5 Konvertieren und Anpassender Quellcode-Dateien(.h,.cpp.,pp) 9](#_Toc354989857)

[3.6 Konvertieren und Anpassen der Formulare (.ui) 9](#_Toc354989858)

[3.7 Rest-Arbeiten 9](#_Toc354989859)

[3.7.1 Suche nach fehlerbehafteten const char \* Zeigern 9](#_Toc354989860)

[3.7.2 find(…) - Ersetzungen 10](#_Toc354989861)

[3.7.3 Form-Header-Dateien 10](#_Toc354989862)

[3.7.4 QWidget - Methoden 10](#_Toc354989863)

[3.7.5 Sonstiges 11](#_Toc354989864)

# C++ -Code-Anpassungen

## Wrapper-Klassen

Wrapper Klassen sind Klassen, die eine Qt4-Klasse kapseln aber zusätzlich noch Teile des Qt3 API anbieten. Diese Klassen wurden erstellt, damit nicht die Qt3-Kompatibilitäts—Funktionalität genutzt werden muss. Trotzdem kann Quellcode der diese Klassen nutzt (zunächst) unverändert bleiben. Anpassung der Semantik wird zentral in der gekapselten Klasse vorgenommen bis die Umsetzung stabil ist. Nach dieser Phase könnte diese Wrapper-Klasse entfernt werden.

Wrapper Klassen können nur im C++-Code verwendet werden, es gibt keine Widgets, die im Designer in Formularen verwendet werden können.

### QHeader / HeaderView

Die Klasse QHeader wird durch die Wrapper-Klasse HeaderView, die von QHeaderView abgeleitet wird, ersetzt.

Die Verwendung der HeaderView-Klasse erfolgt über einen static cast, der das Qt3-API der HeaderView-Klasse zugänglich macht. Diese Art der Verwendung funktioniert nur solange in der HeaderView-Klasse keine Member-Variablen definiert und genutzt werden.

### QListView / ListView

QListView wird durch die Wrapper-Klasse ListView ersetzt, ebenso QListViewItem durch ListViewItem und QListViewItemIterator durch ListviewItemIterator. Alle diese Wrapper-Klassen sind von den entsprechenden Klassen von QTreeWidget abgeleitet also QtreeWidget, QTreeWidgetItem und QTreeWidgetItemIterator. ListView-Klassen werden als Basis-Klassen von unterschiedlichen CPlugin-Widgets und von anwendungsspezifischen Plugin-Widgets verwendet (CListView, MCoreProgramListView, usw.).

Ein wesentlicher Unterschied zu Qt3 besteht bei den ListViews darin, dass die virtuelle Methode

*virtual void QListViewItem::paintCell ( QPainter \* p, const QColorGroup & cg, int column, int width, int align )*

für QListViewItem Klassen fehlt.

Diese Methode wird von der ListViewItem-Klasse durch

*virtual void ListViewItem::paintCell(QPainter \* painter, const QStyleOptionViewItem & option, const QModelIndex & index);*

ersetzt.

Die Ersetzung der Signatur (Prototyp und Implementierung) bei abgeleiteten ListViewItem-Klassen wird vom automatisierten Generierungsprozess. Der Implementierungscode muss manuell angepasst werden (zu diesem Zweck kann das Referenz-Projekt Mosaic\_F zu Rate gezogen werden).  
Die Implementierungen, die im wesentliche die Farbe des Zellentexts ändern, kann die ListViewItem-Methode setColor(…) verwendet werden. Alle Parameter der unter Qt3 verfügbaren Methode können auch über die Parameter der neuen ListViewItem-Methode gewonnen werden.

### QWizard / Wizard

QWizard wird durch die Wrapper-Klasse Wizard ersetzt, die von QWizard abgeleitet ist. Die sematischen Unterschiede und die Unterschiede von QWizard sind zwischen Qt3 und Qt4 so gross, dass eine solche Wrapper-Klasse sinnvoll ist. Dafür spricht auch die Existenz einer Q3Wizard-Klasse im Qt3-Support-Code unter Qt4.

Wizard Klassen werden von anwendungsspezifischen Plugin-Widgets in abgeleiteten Formularen verwendet, wobei das Formulardesign immer über eine QWizard-Klasse (Qt4) erfolgt.

Die Wizard-Klasse besitzt im Shacira-Framework für Qt4 eine Sonderstellung. Sie ist sowohl Wrapper-Klasse für QWizard, als auch Trägerklasse (siehe Kapitel 2) für QWizard-Formulare.

Da die Semantik der Methode QWizard::next(…) unter Qt4 nicht mehr der unter Qt3 entspricht, wurde es notwendig in der Wrapper-Klasse eine semantisch zu Qt3 kompatible Methode einzuführen, die dann in abgeleiteten QWizard Klassen entsprechend angepasst werden muss (QWizard::next() wird zu Wizard::nextPage(…)).

## Sonstige-Klassen

### QRangeControl

QRangeControl gibt es nicht mehr. Die Funktionalität wird im Wesentlichen durch Verwendung von QSlider oder QSpinBox als Basisklasse portiert.

### QValueVector<T>

QValueVector entfällt und wird durch entsprechende Vektoren vom Typ QVector<T> ersetzt.

Als Iteratoren werden Standard-Iteratoren verwendet.

### QPtrList<T>

QPtrList entfällt und wird durch entsprechende Listen vom Typ QList<T\*> ersetzt

Als Iteratoren werden Standard-Iteratoren verwendet.

# Formulare und Trägerklassen

Ein wesentlicher Unterschied zu Qt 3 tritt zutage, wenn Formulare, die mit Qt Designer erstellt wurden, instanziiert oder sogar abgeleitet werden müssen. Alle Formulare, die mit Qt Designer erstellt wurden, können nicht direkt instanziiert oder abgeleitet werden, sondern sie benötigen eine „Trägerklasse“. In diese Trägerklasse wird der Formularinhalt über die Methode ::setupUi(BasisKlasse \*) „injiziert“.

Instanziiert oder abgeleitet wird dann die Trägerklasse, die den Formularinhalt enthält.

Der Formularinhalt liegt als Klasse vor, die von uic erzeugt wird. Im Falle des MosaicFrame erzeugt uic die Klasse *Ui\_MosaicFrame* (Formular-Klasse) mit Zeigern auf alle widgets, layouts usw. die Bestandteil des Formulars MosaicFrame sind.

Um diesen Prozess zu vereinfachen wurden im Shacira-Framework Träger-Klassen für die im Framework verwendeten Klassen als C++-Template-Klasse angelegt:

Als template-Parameter wird der Typ (Klasse) der vom Designer generierten Formularklasse eingetragen. Da im Shacira-Framework unterschiedliche Klassen als Formulare verwendet werden, gibt es für jede unter Qt3 verwendete Formularklasse eine entsprechende Träger-Klasse, die die mit dem Designer erstellten Formulare (.ui) mit einer Basisklasse verbinden, die dann abgeleitet werden kann.

* Page: Template-Klasse für CPage
* AppFrame: Template-Klasse für CAppFrame
* StartupForm: Template-Klasse für CStartupForm
* ActionDialog: Template-Klasse für CActionDialog
* Dialog: Template-Klasse für QDialog
* Wizard: Template-Klasse für QWizard

Bei den Träger-Klassen handelt es sich also um Klassen-Templates, die mit der konkreten Formularklasse als template-Parameter verwendet werden um gegebenenfalls in einer abgeleiteten Klasse auf die konkreten Formularinhalte (im wesentlichen widgets und layouts) zugreifen zu können.

Im Beispiel der Klasse MosaicFrame wäre dies:

*Appframe<MosaicFrame>*

Die Wrapper-Klasse instanziiiert im Konstruktor die Formular-Klasse und macht sie in der Folge über die Methode ui() zugänglich;

Da unter Qt3 nicht alle Formulare einer Shacira-Anwendung (eigentlich nur wenige) abgeleitet wurden wäre es nötig, beim Erstellen der Page-Listen den Code so zu ändern, dass aus den Formularen zum Beispiel konkrete CPage, CAppframe usw –Instanzen erzeugt werden können.

Dazu müssen dann aber sämtliche include-Anweisungen abgeändert werden:

#include „ovw\_posting.h“

Wird zum Beispiel ersetzt durch ein include auf die abzuleitende Basisklasse und ein include auf den generierten Formularklassen-Code

#include cpage.h  
#include ui\_ovw\_posting.h

Die Makros, die zur Instanziierung verwendet werden müssen entsprechend angepasst werden.

Um den umfangreichen Änderungsaufwand zu vermeiden, werden im Transformationsprozess für alle ui-Dateien Header-Dateien erzeugt, die das gleiche API anbieten wie unter Qt3 (wobei in diesem Fall eigentlich nur der Konstruktor eine Rolle spielt).

Diese Header-Dateien liegen im selben Verzeichnis wie die Formulardatei (.ui).

Damit werden für die Erzeugung der Page-Listen keine Änderungen am Code unter Qt4 benötig.

Diese Erleichterung gilt aber nur für automatisiert erstellte Formulare. Beim Erstellen neuer Formulare (nach der Portierung) muss dies manuell erfolgen.

## Vereinfachte Ableitung von Formularklassen

Für einfach verwendete Formulare (Formulare, die nicht abgeleitet sondern nur instanziiert werden) werden im Generierungsprozesss für die ui-Dateien für jedes Formular eine Header-Datei generiert, die dem unter Qt3 verwendeten Ableitungsmodell entspricht.

Mit Hilfe dieser Header-Dateien kann der Code, der bisher unter Qt3 zur Ableitung von Formularen verwendet wurde völlig unverändert auch unter Qt4 verwendet werden.

Damit entfällt auch der Aufwand für die Änderung von abgeleiteten Formularen.

# Schritte im Portierungsprozess (mosaic)

Die automatische Konvertierung von Qt 3 Dateien in erfolgt mit Hilfe der beiden Werkzeuge q4\_uitrans und q4\_uiport. Als Frontend wird q4\_uitrans verwendet, q4\_uiport wird von q4\_uitrans gegebenenfalls verwendet. q4\_uiport kann zu Testzwecken direkt verwendet werden.

Für die korrekte Verwendung von uiport müssen folgende Punkte gewährleitet sein:

* Benötigte plugins müssen aktualisiert als release und debug version im designer/plugins verzeichnis vorhanden sein.
* Der passende uic3-Patch muss im QTDIR/bin Verzeichnis vorhanden sein.

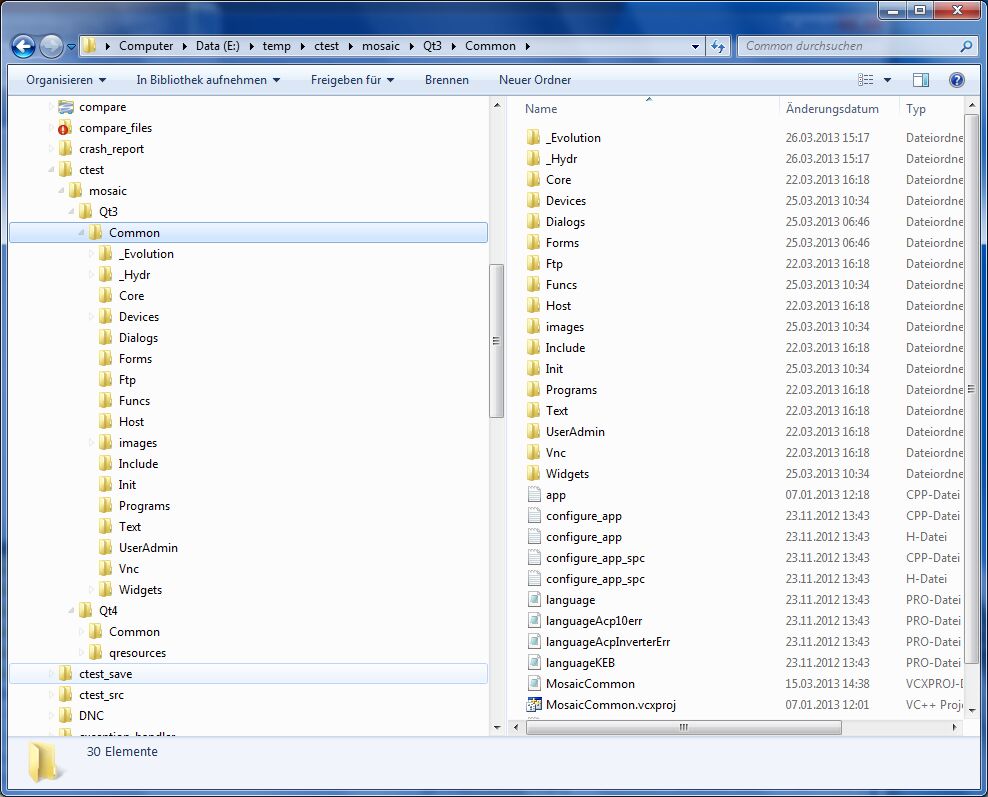
Der uic3-Patch ist im Verzeichnis Shacira/Qt zu finden für die Qt Version 4.8.4 befindet sich dort im Shacira/Qt/bin eine schon erstellte Patch-Version (uic3.exe.484). Für Qt 4.8.4 muss uic3.exe.484 nach QTDIR/bin/uic3.exe kopiert werden. Für andere Qt Versionen muss der Patch unter Verwendung der entsprechenden Patch-Dateien und dem Quellcode dieser Version erstellt werden.

Zur Vorbereitung müssen alle für die Konvertierung relevanten Dateien aus einem gültigen Qt3-Mosaic-Verzeichnis in ein beliebiges Testverzeichnis unter Qt3 kopiert werden. Hierzu in das Qt3-Mosaic-Verzeichnis wechseln, dort mit e:\temp\ctest als Testverzeichnis:

* xcopy Common\\*.ui e:\temp\ctest\mosaic\Qt3\Common /s /v
* xcopy Common\\*.cpp e:\temp\ctest\mosaic\Qt3\Common /s /v
* xcopy Common\\*.h e:\temp\ctest\mosaic\Qt3\Common /s /v
* xcopy Common\\*.pp e:\temp\ctest\mosaic\Qt3\Common /s /v
* xcopy Common\\*.vcxproj e:\temp\ctest\mosaic\Qt3\Common /s /v
* xcopy Common\\*.filters e:\temp\ctest\mosaic\Qt3\Common /s /v

Danach im Testverzeichnis eventuell noch vorhandene irrelevante Verzeichnisse löschen (Generated, Debug, Release, …).

Der Konvertierungsprozess verwaltet das Zielverzeichnis <testverzeichnis>\mosaic\Qt4 in diesem Fall e:\temp\ctest\mosaic\Qt4 eigenständig, dort muss also nichts angelegt oder gelöscht werden, auch das Zielverzeichnis selbst nicht



Im Zielverzeichnis …\Qt4 entsteht durch die Konvertierung eine Verzeichnisstruktur, die identisch mit der des Testverzeichnisses ist, aber die für Qt4 angepassten Dateien enthält (und nur diese, an denen der Prozess Änderungen vorgenommen hat. Dateien, die für Qt4 nicht geändert werden (im Prozess).landen auch nicht im Zielverzeichnis.

## 1. Schritt Konvertierung von Projektdateien

Projektdateien müssen für das vsaddin angepasst werden und die Verweise auf QT3Framework durch QT4Framework-Verweise ersetzt werden.

Die Anpassung an das vsaddin wird in Shacira\Doc\ QTVSADDINBUG-27.htm beschrieben, und wird direkt durch „q4\_uitrans pro“ durchgeführt. Nach „q4\_uitrans pro“

Projektmappe laden, dann jedes einzelne Projekt (MosaicCommon, MosaicMachineType, MosaicWindows, und mosaic\_widget\_plugin) in Qt Addin-Projekte umwandeln:

* Rechte Maustaste auf dem Projektknoten ->“Projekt in Qt Addin-Projekt umwandeln“  
  kann einige Zeit dauern!

Nach der Umwandlung die Qt-Projekteinstellungen ändern:

* Korrekte Qt Version einstellen
* Module: Core Gui und Xml
* Alle Verzeichnisse für generierte Dateien auf .Generated ändern.

## Property sheets

Einstellen der qt 4 property sheets über den Eigenschaften-Manager ,dabei sollten auch die VC6-Upgrade property sheets entfernt werden.

* MosaicCommon: q4\_lib\_release.props, q4\_lib\_debug.props
* MosaicMachineType: q4\_lib\_release.props, q4\_lib\_debug.props
* MosaicWindows: q4\_app\_release.props, q4\_app\_debug.props
* mosaic\_widget\_plugin: q4\_plugin\_release.props, q4\_plugin\_debug.props

Für Mosaic-Anwendungen gibt es die property sheets mosaic und mosaic\_1. Diese property sheets beinhalten lediglich die zusätzlichen include-Pfade für das Projekt. Diese include-Pfade sind zum einen die Include-Pfade ins Shacira-Framework und die teilweise „hierarchisch steuernden“ include-Pfade in die Anwendung.

mosaic.props ist für MosaicCommn, mosaic\_1.props ist für die Maschinenserien (\_Hydr, \_Evolution) Das liegt daran, dass die include-Pfade ins Shacira-Framework in den Mosaic-Anwendungen relativ und nicht über eine Umgebungsvariable angegeben werden.

## Zusätzliche include Pfade

* Alle include-Pfade mit $(QTDIR) können entfernt werden, da sie über die Qt4 property sheets eingestellt werden.
* Für die generierten Header-Dateien  
  .\Generated  
  .\..\Generated  
  .\..\..\Generated
* optional  
  .   
  .\..   
  .\..\..  
  und  
  .\include   
  .\..\include   
  .\..\..\include

## weitere Änderungen

* Bei den Präprozessoreinstellunge muss QT\_NO\_STL entfernt werden, weil sonst QString::toStdString() nicht bekannt ist.

## Konvertieren und Anpassender Quellcode-Dateien(.h,.cpp.,pp)

„q4\_uitrans cpp“

## Konvertieren und Anpassen der Formulare (.ui)

„q4\_uitrans ui“

## Rest-Arbeiten

### Suche nach fehlerbehafteten const char \* Zeigern

Bei der Ersetzung von QString::ascii() durch das Makro CONST\_STRING() kann es zu ungültigenVerwendungen eines const char \* - Zeigers kommen, Diese Stellen werden vom Konvertierungsprozess erkannt und durch ein pragma dokumentiert:

Diese Situation tritt nicht nur in automatisch umgesetzten Code auf.

#pragma message(\_\_LOC\_\_ "this is a suspicious usage of macro CONST\_STRING()")

Sowohl .QString::toAscii() als auch QString::toStdString() arbeiten unter Qt 4 mit temporären Objekten QByteArray bzw. std::string. Zum Zeitpunkt der Verwendung eines über diese Aufrufe erhaltenen const char \* Zeigers ist das Objekt aber unter Umständen zerstört und der Zeiger zeigt ins Nirwana.

Der nachfolgende Code dokumentiert die Situation.

// suspicious string usage

bool compareString(const char \* s\_ptr, const QString & s)

{

if (strcmp(s\_ptr, s.toStdString().c\_str()) == 0) {

return true;

} else {

return false;

}

}

void testStrings()

{

QString s = "heinz";

string stdString = s.toStdString();

const char \* s\_ptr1 = stdString.c\_str();

const char \* s\_ptr2 = s.toStdString().c\_str();

const char \* s\_ptr3 = s.toAscii();

bool rc = false;

rc = compareString(s.toStdString().c\_str(), s); // okay

rc = compareString(s.toAscii(), s); // okay

rc = compareString(s\_ptr1, s); // okay

rc = compareString(s\_ptr2, s); // not okay

rc = compareString(s\_ptr3, s); // not okay

int dummy = 0;

}

### find(…) - Ersetzungen

QString::find(..) –Aufrufe werden in QString::indexOf(..) – Aufrufe umgesetzt. Das passiert gegebenenfalls auch für std::string::find(..) –Aufrufe. Diese Aufrufe müssen wieder in find-Aufrufe zurückgeändert werden.

### Form-Header-Dateien

Siehe auch Kapitel 2

q4\_uiport legt für die Verwendung der neuen Formulare Header-Dateien für den Übergang an. Diese Header-Dateien liegen im selben Verzeichnis wie die ui-Datei:

Common\Forms\tmp\_overview.ui  
+ Common\Forms\tmp\_overview.h

Prinzipiell werden diese Header-Dateien nicht benötigt, Sie werden aber in Dateien wie gui.cpp inkludiert. Damit diese include-Anweisungen keine Fehler generieren, muss auf einige dieser Header-Dateien zugegriffen werden.

### QWidget - Methoden

QWidget unterstützt in Qt 4 ohne QT3\_support einige Funtionen nicht mehr. In Methoden von Klassen, die eines der C-Widgets und damit auch CWidgetBase als Basisklasse haben, ist das kein Problem, weil die Methoden in CWidgetBase nachimplementiert sind.

Für andere Situationen gibt es in qt4port.h entsprechende Methoden mit leicht geänderter signatur. Das QWidget – Objekt wird im ersten Parameter übergeben, die restlichen Parameter sind die Parameter wie unter Qt3 (in der Implementierung mit der Semantik von Qt3):

QWidget \* widget;

widget->method(…)

method(widget, …)

Diese Anpassungen müssen manuell erfolgen und gilt für die folgenden Methoden:

QObject \* child(QObject \* object, const char \* objName, const char \* inheritsClass = 0, bool recursiveSearch = true);

QObject \* child(QObject \* object, const std::string & objName, const char \* inheritsClass = 0, bool recursiveSearch = true);

QObject \* child(QObject \* object, const QString & objName, const char \* inheritsClass = 0, bool recursiveSearch = true);

QObject \* parent(QObject \* object, const char \* objName = NULL, const char \* inheritsClass = NULL, bool recursiveSearch = true);

QObjectList queryList(QObject \* object, const char \* inheritsClass = 0, const char \* objName = 0, bool regexpMatch = true, bool recursiveSearch = true);

### Sonstiges

Die Methode split wird nicht mehr von der Klasse QStringList sondern von QString angeboten:  
QStringList::split(string,…) wird ersetzt durch einen Aufruf string.split(…). Die Parameter sind ausser „string“ die gleichen, die Semanti ist wie unter Qt3.

cPublicDialog entfällt, entsprechende Parameter können ersatzlos entfernt werden.

QWidget::name() -> QWidget->objectName()

QObject::name() -> QObject->objectName()

QPainter::redirected(widget, pixmap\*) -> QPainter::redirected(pixmap\*) der erste Parameter entfällt.

QSharedDoubleBuffer gibt es nicht mehr unter Qt4: entsprechende Methodenaufrufe entfallen ersatzlos.

Der Konstruktor QToolButton(parent, name) wird in QToolButton(parent) mit anschliessendem setObjectName(name) umgewandelt.

Der Konstruktor CComboBox(false, parent, name) wird durch CComboBox(parent, name) ersetzt.

QHboxLayout,QVBoxLayout können nur noch mit dem parent konstruiert werden, die fehlenden Parameter werden dann mit setMargin(..) und setSpacing(..) gesetzt.

Dasselbe gilt auch für widgets mit dem Parameter name, der muss mit setObjectName() gesetzt werden.

QFile::name() wird durch QFile::fileName() ersetzt.

QDir::setMatchAllDirs(true) wird ersetzt durch QDir::setFilter(QDir::AllDirs

QTimer::start(5000, parameter) wird durch QTimer::start(5000) ersetzt, der zweite Parameter kann entfallen.

Die Konstruktio von QByteArray(int) wird ersetzt durch QByteArray(int, ' '), das heißt unter Qt4 muss der Initialisierungswert angegeben werden.

Die Zuweisung QPixmap p = QImage i wird ersetzt durch QPixmap p = QPixmap::fromImage(i);

QString::contains(substring) wird durch QString::count(substring) ersetzt.

QString::replace(int, int, NULL) durch QString::replace(int, int) ersetzt.

Bei abgeleiteten Formularklassen, können die Widgets des Formulars nicht mehr direkt zugegriffen werden. Dort muss über die ui()-Methode der Formular-Klasse auf die formularspezifischen Widgets zugegriffen werden.

Beispiel:

LE\_SetStrEje->setFocus() -> ui().LE\_SetStrEje->setFocus()

Bei abgeleiteten Wizard-Formularklassen:

QWizard::...wird ersetzt durch Wizard::...

Basisklassenname::... wird ersetzt durch Wizard::...

Beispiel (clp\_ecectorwizard\_sub):

QWizard::showPage(actualPage) -> Wizard::showPage(actualPage)

Clp\_EjectorWizard::languageChange() -> Wizard::languageChange()

Bei abgeleiteten Dialog-Formularklassen:

QDialog::...-> Dialog::...

Basisklassenname::... Dialog::...

Beispiel (sys\_dirdialog\_sub):

Sys\_DirDialog::languageChange() -> Dialog::languageChange()

QScrollArea::addChild -> wird ersetzt durch QScrollArea::setWidget. Mehr als ein Widget kann scheinbar nicht in eine QScrollArea eingefügt werden; das ist bisher nicht notwendig muss aber noch geklärt werden)

ui\_hlpviewer.h ersetzen durch ui\_hlp\_viewer.h.

QRegExp::search wird ersetzt durch QRegExp::indexIn.

Bei QApplication::setFont(..) entfällt der zweite Parameter.

Da für QStrings die find Methode im automatisierten portierungsprozess durch indexOf ersetzt wurde (mangels Typ-Informationen) müssen diese Aufrufe bei std:: - Klassen wieder rückgängig gemacht werden.

CTextEdit::text() wird durch CTextEdit::toPlainText() ersetzt.

ASSERT(expression) -> \_ASSERT(expression) ()

QString::simplifyWhiteSpace() wird ersetzt durch QString::simplified().

QDir::currentDirPath() wird ersetzt durch QDir::currentPath()

QFile besitzt keinen public Assignment-Operator mehr:

Zuweisungen der Form QFile x = Qfile(params) warden ersetzt durch QFile x(params).

QWidget::paletteBackgroundColor ist als statische Methode von CWidgetBase mit Übergabeparameter QWidget \* verfügbar. Dasselbe gilt für die set-Methode.

Widgets die von CWidgetBase abgeleitet bieten diese Methoden wie unter Qt3 an (ohne QWidget \* Parameter).

QIconSet wird ersetzt durch QIcon.

QIconSet::Size Parameter entfallen ersatzlos.

QObject Listen:

Die include Anweisung #include <qobjectlist.h> wird fernen und gegebenenfalls verwendete Iteratoren auf Standard-Iteratoren umstellen (::iterator, ::const\_iterator usw.)

QPaintDevice::bitBlt(x, y, doubleBuffer) wird durch QPainter::drawImage(..) ersetzt. Bitblt wird von unter Qt4 nicht mehr angeboten: doubleBuffer ist vom Typ QPixmap:

Qt3:

bitBlt(this, x, y, &doubleBuffer);

Qt4:

QPainter painter(this);

painter.drawImage(x, y, doubleBuffer.toImage());