Australian Synchrotron

QE Framework -Getting Started

Author: Andrew Rhyder

11 November 2012

# Abstract

# Introduction

This document contains details on configuring a run time environment or an application development environment where the QE Framework will be used.

The QE Framework can be used in three ways:

* **Code Free GUI systems** using Qt’s Designer application with the QE Framework plugins to design GUIs, and the QEGui application to present GUIs to users.
* **Code Rich GUI development** using Qt’s Integrated Development Environment with the QE Framework widgets and data objects to design GUI applications.
* **Console application development** using Qt’s Integrated Development Environment with the QE Framework data objects to design console applications that can access EPICS data.

Note, there are many variations to the above, such as using another Integrated Development Environment like Eclipse, or developing new plugin widgets to implement desired functionality, then using those widgets within a code free GUI development.

# How to use this document

You may first like to read ‘Components and prerequisites’ to familiarise yourself with the QE framework components and what may be required to use the QE framework.

Then refer to ‘Setting up your environment’ to determine what is needed to support your specific tasks.

Once clear about what you require, follow the specific instructions referenced.

# Components and prerequisites

The QE framework includes the following components:

* QE library (libQEPlugin.so or QEPlugin.dll)  
  This library contains all classes that implement the QE framework including data objects, widgets (Qt plugins) and supporting classes.
* QEGui  
  A stand alone application that which is used to present a collection of Qt User Interface files as an integrated control centric GUI system.
* Documentation  
  Includes:
  + Getting started guide (this document)
  + User manual
  + QE framework reference

The following components are required to use the QE framework. Some are optional, depending on what you are doing:

* Qt libraries.  
  The C++ based application framework underpinning QE.  
  At the time of writing, the QE Framework is being used with Qt 4.6 to Qt 4.8. Pre Qt 4.6 versions bave been used and are likely to still be OK.
* Qwt library  
  ‘Qt Widgets for Technical Applications’ provides the base widgets used for the plotting and tracing widgets in the QE framework
* CA libraries  
  EPICS Channel Access provides access to EPICS
* Gnu tools (Optional)  
  Qt uses gcc and gmake to build the QE framework.
* Qt designer (Optional)  
  If you are creating Qt User Interface files for a ‘code free’ GUI system, you will also need Qt Designer for laying out the user interface files.
* Qt Creator (Optional)  
  If you are developing applications that use the QE framework, you will also need the Qt Creator IDE.

# Setting up your environment

Depending on what you are going to do with the QE framework, your set up tasks may vary.

## No matter what you are doing...

No matter what you are doing, you will require:

* The Qt 4 libraries. You may well already have Qt 4 as it is used by many other common applications. Refer to ‘Qt libraries’ for details.
* The QE framework. Refer to ‘QE framework’ for details.

## Code free GUI implementation.

Nothing extra required. Use the QEGui application (part of the QE framework package) to present a control system GUI fully defined by a suite of Qt User Interface files.

Refer to ‘???’ regarding QEGui.

## Code free GUI development

If you want to create a control system GUI defined by a suite of Qt User Interface files, you require:

* Qt Designer. This is Qt’s drag and drop form design tool. It is included with the Qt 4 libraries and with the Qt SDK

## Code rich development

If you want to write graphical or non-graphical application code using the QE framework, you will require:

* A development environment such as Qt’s Integrated Development Environment ‘Creator’. Refer to ‘???’ for details.

Note, there are alternatives to Creator, such as the Eclipse IDE or your favourite text editor plus various Qt and Gnu command line tools. For details, see elsewhere.

# Installation tasks

## Qt libraries

Firstly, you may already have the Qt libraries as many common applications depend on them. To check on Linux run the command:

yum info qt-x11

You will also end up with the Qt libraries if you install the Qt SDK.

If you don’t already have the libraries, and you don’t intend on installing the Qt SDK, you can install the Qt libraries with the following command:

yum install qt-x11

Alternatively, the Qt libraries (and instructions to build them) are available for download from <http://qt-project.org/downloads>.

Note, at the time of writing, the QE Framework is being developed in Qt environments from Qt 4.6 to Qt 4.8. Pre Qt 4.6 versions have been used and are likely to still be OK.

Note, the Qt library dependencies should be managed automagically when installing from a Linux distribution repository. The dependencies may not be accommodated when downloading and building Qt manually. One common dependency problem has been older versions of freetype and fontconfig on RedHat EL 5. Installing freetype-2.4.3 and fontconfig-2.8.0 on RedHat EL 5 resolves this dependency issue.

## QE framework

The QE Framework is available at <http://sourceforge.net/projects/epicsqt>

To use the framework, you must download the epicsqt project, build it, and deploy the appropriate QE Framework components.

There are three ways described below to get the framework:

* Download the framework source in a single gzip file
* Use an svn client to extract the source from its svn repository
* Use an svn client to download a single framework makefile which can then be used to download the framework.

There are three ways described below to build the QE Framework:

* Qt Creator
* qmake command
* Framework makefile

There are hundreds of ways to deploy the QT based libraries and applications. Two simple alternatives are described below. Both allow the QE Framework to be used as a library loaded by applications and as a Qt Plugin:

* Manual deployment
* Framework makefile

Note, all the examples for getting and building the QE Framework assume the epicsqt project has been placed in the directory ~/epicsqt.

### Getting the epicsqt project as a gzip file

The latest gzip file is available for download from <http://sourceforge.net/projects/epicsqt>.

The same gzip file, along with earlier versions and links to documentation is available at <http://sourceforge.net/projects/epicsqt/files/>

The contents of the gzip can be extracted using the following command:

tar -zxvf epicsqt-1.1.8-src.tar.gz

### Getting the epicsqt project using svn

The latest source code for the QE Framework can be downloaded using the following svn command:

svn co https://epicsqt.svn.sourceforge.net/svnroot/epicsqt epicsqt

This command will create a directory ‘epicsqt’ containing the QE Framework project.

An svn client can also be used to get earlier version of the source code.

SourceForge also provide a browser based interface to the svn repository at <http://epicsqt.svn.sourceforge.net/viewvc/epicsqt/>

### Getting the build script using svn

A single build script that will download the entire project source from the svn repository can be obtained using the following svn command:

svn export https://epicsqt.svn.sourceforge.net/svnroot/epicsqt/resources/makefile

The following command will download the QE Framework:

make svncheckout

Note, the makefile can also be used to build, clean, and package the QE Framework. Refer to ‘???’ for details.

Note, the makefile is itself part of the epicsqt project files and can be found in the ‘resources’ directory.

### Environment for building the framework

export QWT\_INCLUDE\_PATH=/usr/include/qwt

### Building the framework using Qt Creator

* Open the epicsqt.pro file in Qt Creator. The epicsqt.pro file is in the top level directory of the epicsqt project.
* Ensure shadow building is unchecked in the ‘Projects’ options.
* Build the project.

If you are using a multi core processor, adding a compile option –j*n* (where *n* is the number or concurrent compilations) will speed up compilation considerably. For example, -j4 uses most of the CPU time available on a 4 core CPU.

### Building the framework using qmake

To build the QE Framework using Qt’s qmake, enter the following commands in the epicsqt project directory:

* qmake
* make

### Building the framework using the framework makefile

A single project makefile is available to download the entire project source from the svn repository, build the framework, and package the framework. The makefile can be downloaded from the svn repository (refer to ‘Getting the build script using svn’ for details) and is also itself part of the epicsqt project source and can be found in the ‘resources’ directory.

The following targets can be specified for the project makefile:

make all # (default) download and build the project  
make svncheckout # download the project  
make buildframework # build the project  
make clean # clean the project  
make package # copy all deployable components into a package directory

### Deploying the QE Framework – Manual deployment

There are many options for deploying the QE Framework, but the following describes a simple deployment that allows the QE Framework applications to be located and the QE Framework library to be used as both a standard library and a Qt Designer Plugin.

Extend the path to include the location of the QEGui application:

export PATH=$PATH:/home/*<user>*/epicsqt/applications/QEGuiApp

Ensure applications can find the QE Framework library as a standard library:

export LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:/home/*<user>*/epicsqt/framework/designer

Ensure Qt can find the QE Framework library as a Qt Designer Plugin:

export QT\_PLUGIN\_PATH=:/home/*<user>*/epicsqt/framework

Note, the above command allows Qt to find a ‘designer’ directory containing the QE Framework library libQEPlugin.so. Refer to Qt documentation on the many ways Qt can locate and load Plugins.

### Deploying the QE Framework – using framework makefile

A single project makefile is available to download the entire project source from the svn repository, build the framework, and package the framework. The makefile can be downloaded from the svn repository (refer to ‘Getting the build script using svn’ for details) and is also itself part of the epicsqt project source and can be found in the ‘resources’ directory.

The following command uses the framework makefile to place all required applications and libraries (including CA libraries) into a single ‘package’ directory, and to set up environment variables to allow the QE Framework library to be located by applications and by Qt’s Plugin system:

make package

## Developing Applications in Qt Creator using the QE Framework

???

==============================OLD DOCO BELOW HERE===========================

#=============================

# Qt

#=============================

# Used by Qwt dependant components to locate Qwt include files

export QWT\_INCLUDE\_PATH=/usr/include/qwt

# Used by applications to load the framework library directly. (not using the Qt plugin mechanism)

export LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:/home/rhydera/epicsqt/framework/designer

# Used by Qt plugin mechanism to load the framework.

export QT\_PLUGIN\_PATH=/home/rhydera/epicsqt/framework

# Extend path to pick up designer and creator

#QTDIR=/home/rhydera/qtsdk-2010.05

#QTDIR=/usr/local/Trolltech/Qt-4.8.3

#export PATH=$PATH:$QTDIR/qt/bin:$QTDIR/bin

# Extend path to pick up designer

export PATH=$PATH:/home/rhydera/qt-everywhere-opensource-src-5.0.0-beta2/qtbase/bin

# Extend path to pick up qtcreator

export PATH=$PATH:/home/rhydera/qtcreator-2.6.0/bin

# Extend path to pick up qtcreator

#export PATH=$PATH:$QCAQTDIR/bin

export QE\_ARCHIVE\_LIST=archiver.synchrotron.org.au:80/cgi-bin/ArchiveDataServer1.cgi archiver.synchrotron.org.au:80/cgi-bin/ArchiveDataServer2.cgi

QCAFRAMEWORK

Used by QCa application and components to locate QCa framework

The following script is typical of a script to set up these variables, extend the path to allow command line access to qtcreator and designer, and set up an appropriate EPICS environment.

#=============================

# Qt

#=============================

# Current Qt installation used for QCa development

export QCAQTDIR=~/qtsdk-2010.05

# Used by QCa applications and components to locate QCa framwork

export QCAFRAMEWORK=~/epicsqt/ca\_framework

# Extend path to pick up designer

export PATH=$PATH:$QCAQTDIR/qt/bin

# Extend path to pick up qtcreator

export PATH=$PATH:$QCAQTDIR/bin

#=============================

# EPICS

#=============================

export EPICS\_BASE=/epics/base

export EPICS\_HOST\_ARCH=linux-x86\_64

export EPICS\_TOP=${EPICS\_BASE}

export EPICS\_EXTENSIONS=${EPICS\_BASE}/extensions

export EPICS\_CA\_ADDR\_LIST=127.0.0.1

# Building the run time environment used for code free GUI development

Build the following Qt CA Framework projects:

* epicsqt/ca\_framework/qwt-5.2.1/qwt.pro
* epicsqt/ca\_framework/plugins/QCaDesignerPlugin.pro
* epicsqt/applications/ASguiApp/ASgui.pro

Deploy the following components:

* Qt’s Designer
* Qt’s run time libraries
* Qt CA Framework libQCaPlugin.so plugin library
* Qt CA Framework ASgui application

# Writing applications that link to QCa widgets

When writing applications that link to QCa widgets, the following line should be included in the Qt project file:

LIBS += -L$$(QCAFRAMEWORK)/plugins -lQCaPlugin

(Note, the environment variable QCAFRAMEWORK points to the QCa development framework)

For the above to work the Qca plugin library must be ready for use by the linker. This can be done as follows:

The QCa plugin library (or a link to it) should exist as follows:

/usr/lib/libQCaPlugin.so

The following command should be run to cache the plugin library

ldconfig

# Laying out forms in creator

If the QCa plugins are to be used when laying out forms in creator, the QCa plugin library (or a link to it) should exist as follows:

$QCAQTDIR/bin/designer/libQCaPlugin.so

(Note, the environment variable QCAQTDIR should be set to point to the version of Qt used for QCa development)

# Laying out forms in designer

If the QCa plugins are to be used when laying out forms in designer, the QCa plugin library (or a link to it) should exist as follows:

$QCAQTDIR/qt/plugins/designer/libQCaPlugin.so

(Note, the environment variable QCAQTDIR should be set to point to the version of Qt used for QCa development)

# Writing applications that dynamically load UI files

When writing applications that dynamically load UI files (using QUiLoader), where the UI files include QCa plugins, the QCa plugin library (or a link to it) should exist as follows:

<your-executable-directory>/designer/libQCaPlugin.so

Alternatively, your application can specify where QUiLoader should search for the plugin using QUiLoader::addPluginPath()

# Installation tasks

## Installing Qt Libraries

If you are laying out Qt User Interface SDK, rather than just running, applications Depending on how you will be using the QE framework you may need the the following Depending on how you will be using the framework you

* Qt 4.6, 4.7 or 4.8  
  At the time of writing, the QE Framework is being developed in Qt environments from Qt 4.6 to Qt 4.8. Pre Qt 4.6 versions have been used and are likely to still be OK. Depending on requirements the following Qt components are required:
  + Application development – Qt libraries 4.8.3

or

* + Application runtime – Qt Libraries 4.8.3

both components are available from <http://qt-project.org/downloads>.Note, the Qt Libraries are available via repositories for most major Linux distributions.

The Qt library dependencies should be managed when installing from a Linux distribution repository. The dependencies may not be accommodated when installing Qt using downloaded binaries from Nokia. One common dependency problem has been older versions of freetype and fontconfig on RedHat EL 5. Installing freetype-2.4.3 and fontconfig-2.8.0 on RedHat EL 5 resolves this dependency issue.

## Installing Qt Creator

## Installing Qt Designer

## Installing Qwt