Qt BASED GUI SYSTEM FOR EPICS CONTROL SYSTEMS*

A. Rhyder#, R. N. Fernandes, A. Starritt, Australian Synchrotron, Clayton 3168, Australia

Abstract

The Qt-based GUI system developed at the Australian Synchrotron for use on EPICS control systems has recently been enhanced to including support for imaging, plotting, user login, logging and configuration recipes. Plans are also being made to broaden its appeal within the wider EPICS community by expanding the range of development options and adding support for EPICS V4. Current features include graphical and non-graphical application development as well as simple "code-free" GUI design. Additional features will allow developers to let the GUI system handle its own data using Qt-based EPICS-aware classes or, as an alternative, use other control systems data such as PSI's CAFE.

INTRODUCTION

The Qt-based GUI system, known as QE framework or simply QE, is a layered framework based on C++ and Qt for accessing EPICS data using Channel Access (CA). It is used on several beamlines at the Australian Synchrotron. Channel Access is one of the core components of an EPICS system allowing a CA client application to access control system data which may be located on different hosts throughout a network [1]. While CA is the default means to access EPICS data, its use is not trivial. A significant understanding on how this component works is required to read or write data. The complexity of setting up and terminating CA requests leaves room for error. The QE framework handles much of this complexity including initiating and managing a channel. Applications using QE can interact with Channel Access using Qt-based classes and data types. CA updates are delivered using Qt signals and slots mechanism. It provides access to EPICS data at several levels including programmatic reading and writing of data, EPICS-aware widgets such as push buttons, sliders and text widgets for developing GUI applications [2]. When these plugins are used within Qt Designer, GUIs interacting with EPICS can quickly be assembled without the need for any code development by meaning of simple drag & drop operations.

FRAMEWORK OVERVIEW

The QE framework allows access to Channel Access graphically through Qt-based widgets or through Qt friendly data objects. The data objects manage Channel Access connections and provide a simple, comprehensive, object oriented view of the CA data and related attributes. The QE graphical widgets and supporting QE data objects form a hierarchy of classes that is open at all levels to the developer. Appropriate use of these classes is shown in Table 1. Also, the framework includes an application, QEGui, which is available to present control centric Qt user interface files.

Table 1: QE framework classes and their functionality.

Classes	Functionality
Data objects: QEObject QEInteger QEString QEFloating	Provides a convenient object oriented way to access the CA library. Hides CA specific complexity and provides read/write conversions to and from EPICS data types where required. Adds Qt features such as signals and slots to handle data updates. These classes are used programmatically.
Standard widgets: QEComboBox QEForm QEFrame QEGroupBox QELabel QELineEdit QEPushButton QERadioButton QESlider QESpinBox	Graphical objects that allow users to interact with CA data using simple and familiar graphical controls. These classes may be used programmatically or within Qt Designer.
Extended widgets: QEAnalogProgressBar QEBitStatus QEConfiguredLayout QEFileBrowser QEImage QELink QELog QELogin QEPeriodic QEPlot QEPvProperties QERecipe QEScript QEShape QEStripChart QESubstitutedLabel	Graphical objects that allow users to view CA data through a broad range of display models and support sophisticated control system user interface design. These classes may be used programmatically or within Qt Designer.

*Work supported by the Australian Synchrotron
#andrew.rhyder@synchrotron.org.au

ISBN 978-3-95450-124-3

USAGE PARADIGMS

Thanks to its architecture, the QE framework may be used by different people and in different ways. The simplest is as a "code free" development environment where the user builds GUIs without the need to program by means of dragging & dropping EPICS-aware widgets into a graphical user interface. Each widget can be then configured through a set of properties and inter-connected through Qt signals to exchange data with other widgets. All these can be done within Qt Designer in a user-friendly fashion (see Figure 1). A single application QEGui is used to present a set of user interface files as an integrated control system suite.

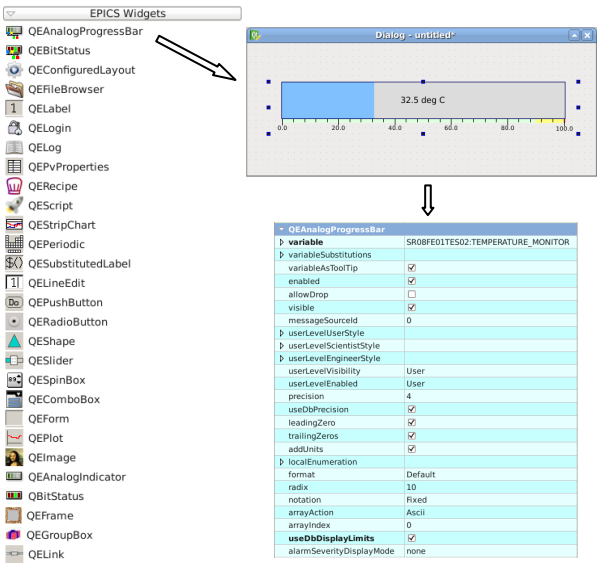

Figure 1: QE framework within Qt Designer.

Another way of building GUIs is to use QE programmatically (see Figure 2). This effectively allows full control of the framework and even extends it to solve specific issues through C++ inheritance mechanism. It also allows the creation of non-graphical applications – e.g. servers – interacting with CA. A complete rewrite of the API reference documentation (generated by Doxygen) is currently being done. It will help the developers to better understand and use QE programmatically.

```
void show(void)
{
    QFrame *myFrame;
    QELabel *myLabel;

    myFrame = new QFrame();

    //creates object called 'myLabel' of type QELabel (belonging to QE framework)
    myLabel = new QELabel();

    //connects 'myLabel' to the PV called 'MACHINE_CURRENT'
    myLabel->setVariableName("MACHINE_CURRENT");

    //adds 'myLabel' to the frame layout and display machine current
    myFrame->layout()->addWidget(myLabel);

    myFrame->show();
}
```

Figure 2: Using the QE framework programmatically.

FUTURE DEVELOPMENTS

The development of the QE framework has been intense in recent months and more developments are expected for the near future. These will produce a framework more stable/mature capable of coping with ever changing requirements. Some key developments were already identified namely:

- Testing. A suite of test units will be implemented to ensure that QE complies with the requirements. It will guarantee that nothing breaks if the development effort intensifies and goes beyond the Australian Synchrotron.
- Guidelines. The specification of guidelines is crucial as the QE framework goes towards an international collaboration development. These guidelines will guarantee that QE maintains its architectural and implementation coherence even if several people are working on it in a scattered fashion. Some of these guidelines will be enforced through automated tests.
- Binding. More and more, the Python programming language is becoming the "lingua franca" of the scientific community. Plans are being made to export the QE framework into this language through bindings. After initial analysis of binding generators, SIP is the chosen one to accomplish such task.
- Layering. Further abstraction/separation between data and graphical (widgets) classes. This will allow other control systems data to be supported by the framework without modifying the graphical layer.
- EPICS V4. New concepts were introduced in the latest version of EPICS such as structured data or a new channel access protocol called pvAccess [3]. QE will be updated to support these.

CONCLUSION

QE is a framework which enables the creation of Qt-based GUIs interacting with EPICS data in a "code-free" fashion or programmatically via C++. This allows GUIs to be built by people without programming skills or by people that need more control. Future developments will allow the use of the framework in Python by the scientific community. Special concern will be made to support EPICS version 4 and its new features.

REFERENCES

[1] P. Stanley, "Channel Access Client Tutorial", Los Alamos National Laboratory, November 1997; http://lansce.lanl.gov/EPICSdata/ca/client/caX5Ftutor-4.html

[2] A. Rhyder et al., "Qt EPICS Development Framework", PCaPAC 2010, Saskatoon, October 2010.

[3] EPICS v4 Working Group, "pvAccess Protocol Specification", September 2012; http://epics-pvdata.sourceforge.net/pvAccess_Protocol_Specification.html

ISBN 978-3-95450-124-3