



QE Framework - QEgui and User Interface Design

Andrew Rhyder

19 November 2013

Copyright (C) 2013 Andrew Rhyder.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Contents

Introduction	6
License.....	6
Overview	6
Qt Designer	6
QEGui	7
QE widgets	8
QEGui	9
Command format:.....	9
File location rules.....	10
Saving and restoring configurations	11
Opening GUIs	11
Built in forms.....	12
Editing GUIs.....	13
Menu bar and tool button customisation.....	14
Customisation file format	15
Default customisations	20
Window customisation from a QE button widgets.....	20
Separators in menus	20
Built-in functions.....	20
Repeating sections of a set of window customisations.....	21
Tricks and tips (FAQ)	22
GUI titles	22
User levels.....	23
Logging	25
Finding files	27
Sub form file names	27
Sub form resizing	28
Ensuring QERadioButton and QECheckBox is checked if it matches the current data value	28
What top level form to use	28
GUI based on a QScrollArea won't scroll in QEGui	29
How does a user interact with an updating QE widget	29
Widgets disappear when escape is pressed!	29

A QE widget displays the correct alarm state only when a form is first opened.....	29
A QEPlot widget is not displaying updates	30
Droppable widgets as scratch pads and customisable GUIs.....	30
QE widgets	30
Common QE Widget properties.....	30
variableName and variableSubstitutions.....	31
variableAsTooltip	32
subscribe	32
enabled	32
allowDrop.....	33
visible	33
messageSourceId	33
userLevelUserStyle, userLevelScientistStyle, userLevelEngineerStyle.....	33
userLevelVisibility	33
userLevelEnabled	33
displayAlarmState	34
String formatting properties.....	34
precision.....	34
useDbPrecision.....	34
leadingZero	34
trailingZeros	34
addUnits	34
localEnumeration.....	34
format	36
radix	36
notation.....	36
arrayAction.....	36
arrayIndex	36
QEAnalogIndicator and QEAnalogProgressBar.....	37
QEArchiveStatus.....	37
QBitStatus and QEBitStatus	38
QEComboBox	39
QEConfiguredLayout	39

QEFileBrowser.....	40
QEForm	40
QEFormGrid	42
QEFrame and QEGroupBox.....	42
QEImage.....	42
QELabel	50
QELineEdit.....	52
QELink	54
QELog	55
QELogin	57
QNumericEdit.....	59
QEPeriodic.....	61
QEPlot	65
QEPlotter.....	67
Expressions.....	69
Scaling and Presentation Control.....	69
QEPushButton, QERadioButton and QECheckBox.....	69
QEPvLoadSave.....	78
Tool Bar	79
Context Menu	81
Drop	82
XML File Format	82
Future Enhancements.....	83
QEPvProperties	83
Selecting a PV name.....	85
Selecting Displayed Field Names	86
QERecipe	87
QEScratchPad.....	87
QEScript.....	87
QEShape	87
QESimpleShape.....	93
QESlider.....	95
QESpinBox.....	96

QEStripChart	97
QESubstitutedLabel.....	97
QERadioGroup	98
Appendix A.....	100
GNU Free Documentation Licence.....	100

Introduction

This document describes how to use the QE Framework to develop ‘code free’ Control GUI systems. It explains how features of the QE Framework widgets can be exploited, and how the QE Framework widgets interact with each other and with the QEGui application typically used to present the user interface.

While widget properties are referenced, a definitive list of the available properties is available in document [QEReferenceManual.pdf](#).

This document is not intended to be a general style guide, or a guide on using Qt’s user interface development tool, Designer. Style issues should be resolved using facility based style guidelines, EPICS community standards, and general user interface style guides. Consult Qt documentation regarding Designer.

License

The QE Framework is distributed under the GNU General Public License version 3, distributed with the framework in the file COPYING. It may also be obtained from here: <http://www.gnu.org/licenses/gpl-3.0-standalone.html>

Overview

In a typical configuration, Qt’s Designer is used to produce a set of Qt user interface files (.ui files) that implement an integrated GUI system. The QE Framework application QEGui is then used to present the set of .ui files to users. The set of .ui files may include custom and generic template forms, and forms can include nested sub forms. Other applications can also be integrated.

Qt Designer

Designer is used to create Qt User Interfaces containing Qt Plugin widgets. The QE Framework contains a set of Qt Plugin widgets that enable the design of Control System GUIs as shown in Figure 1. These are used, along with standard Qt widgets and other third party widgets. QE widgets display real time data while in designer if supplied with a variable name and appropriate default macro substitutions.

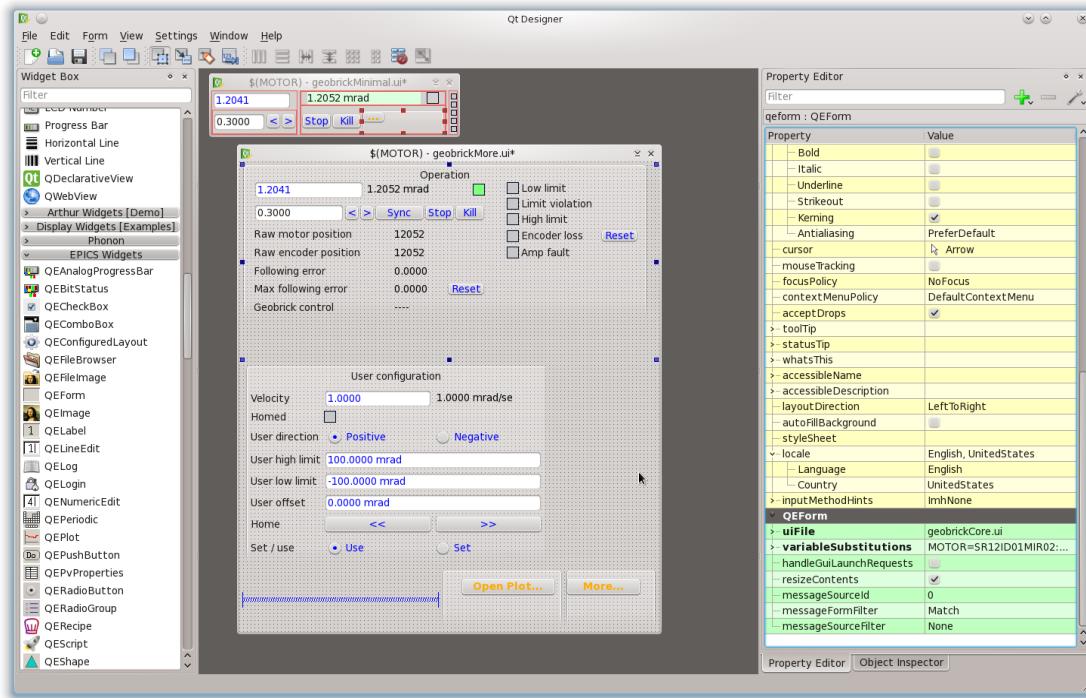


Figure 1 Designer being used to construct GUIs for use by the QEGui application.

QEGui

QEGui is an application used to display Qt User Interface files (.ui files). Almost all of the functionality of a Control System GUI based on the QE Framework is implemented by the widgets in the user interface files. QEGui simply presents these user interface files in new windows, or new tabs, and provides support such as a window menu and application wide logging.

Note: while the application's name is QEGui, the generated executable name is lower case, i.e. qegui for Linux environments, and qegui.exe for Microsoft Windows environments.

Simple but effective integration with Qt Designer is achieved with the option of launching Designer from the QEGui 'Edit' Menu. The user interface being viewed can then be modified, with the changes being automatically reloaded by QEGui.

Refer to 'QEGui' (page 9) for documentation on using QEGui.

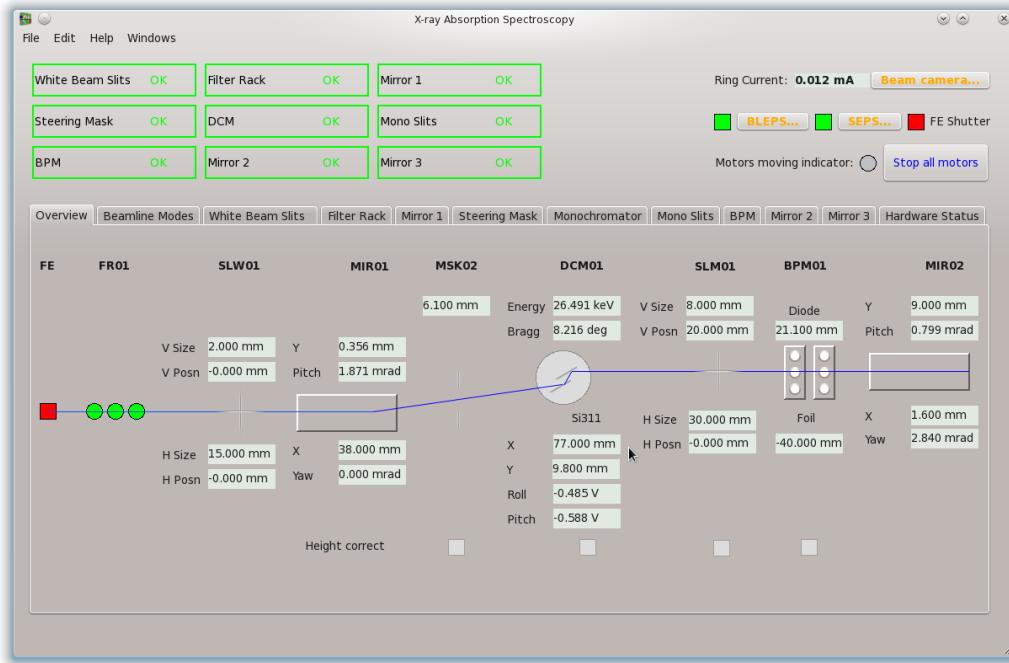


Figure 2 QEgui GUI display application example

QE widgets

QE widgets are self contained. The application loading a user interface file – typically QEgui – does not have to be aware the user interface file even contains QE widgets. The Qt library locates the appropriate Plugin libraries that implement the widgets it finds in a user interface file.

While QE widgets need no support from the application which is loading the user interface containing them, some QE widgets are capable of interacting with the application, and other widgets. For example, a QEPushButton widget can request that whatever application has loaded it open another user interface in a new window.

QE widgets fall into two categories:

- Standard widgets. These widgets are based on a standard Qt widget and generally allow the widget to write and read data to a control system. For example, QELabel is based on QLabel and displays data updates as text.
- Control System Specific widgets. These widgets are not readily identifiable as a single standard Qt widget and implement functionality specific to Control systems. For example, QEPlot displays waveforms.

QEGui

Command format:

```
qogui [-a scale] [-s] [-e] [-b] [-r [configuration]] [-h] [-v] [-m
macros] [-p path-list] [filename] [filename] [filename...]
```

Command switches and parameters are as follows:

- **-s Single application.**

QEGui will attempt to pass all parameters to an existing instance of QEGui. When one instance of QEGui managing all QEGui windows, all windows will appear in the window menu. A typical use is when a QEGui window is started by a button in EDM.

An existing instance of QEGui will only be used if it uses the same macro substitutions (see -m switch)

- **-e Enable edit menu option.**

When the edit menu is enabled Designer can be launched from QEGui, typically to edit the current GUI.

- **-a scale Adjust Scale.**

Adjust the GUI's scaling. This option takes a single value which is the percentage scaling to be applied to each GUI. The value may be either an integer or a floating point number. If specified its value will be constrained to the range 40 to 400.

- **-b Disable the menu bar.**

- **-r [configuration name] Restore Configuration.**

Ignore any filenames provided and restore the named configuration. If no name is provided 'Default' is assumed. Note, multiple configurations may be saved in the same configuration file.

- **-c configuration file] Configuration file.**

Use the specified configuration file when saving and restoring configurations. If no file is specified 'QEguiConfig.xml' in the current working directory is assumed.

- **-p path-list Search paths.**

When opening a file, this list of paths may be used when searching for the file. Refer to 'File location rules' (page 6) for the rules QEGui uses when searching for a file.

The search path format is platform specific and should be in the following forms:

Linux: /home/mydir:/tmp:/home/yourdir

Windows: 'C:\Documents and Settings\All Users; C:\temp;C:\epicsqt'

(Quotes required if spaces are included in the paths)

The search path may end with '...' in which case all sub directories under the path are searched.

For example, assuming /temp/aaa and /temp/bbb exist, -p /temp/... will cause files to be looked for in /temp/aaa and /temp/bbb.

- **-h Display help text explaining these options and exit.**

- **-v Display version information and exit.**

- **[-m macros] Macro substitutions applied to GUIs.**

Macro substitutions are in the form:

keyword=substitution, keyword=substitution, ... and should be enclosed in

quotes if there are any spaces.

Typically substitutions are used to specify specific variable names when loading generic template forms. Substitutions are not limited to template forms, and some QEWidgets use macro substitutions for purposes other than variable names.

- **-w Window customisation file.**

This file contains named sets of window menu bar and tool bar customisations.

Named customisations will be read from this file. If this option is not provided an attempt will be made to use QEGuiCustomisation.xml in the current working directory. A customisation file is optional.

- **-n Default window customisation name.**

This name should be the name of one of the sets of window customisations read from the window customisation file.

- **-t Application title.**

This title will be used instead of the default application title of 'QEGui'

- **filename** GUI filenames to open. Each filename is a separate parameter
If no filenames are supplied, the 'File Open' dialog is presented. Refer to 'File location rules' (page 6) for the rules QEGui uses when searching for a file.

Switches may be separate or grouped. For example ‘-e -m’ or ‘-em’.

Switches that precede a parameter (-p, -m) may be grouped. Associated parameters are then expected in the order the switches were specified. For example:

```
qegui -e -p /home  
qegui -epm /home PUMP=02
```

File location rules

If a user interface file path is absolute, QEGui will simply attempt to open it as is. If the file path is not absolute, QEGui looks for it in the following locations in order:

1. If the filename is for a sub-form, look in the directory of the parent form.
2. Look in the directories specified by the -p switch.
3. Look in the directories specified by the QE_UI_PATH environment variable. Note, these paths may end in ‘...’ in which case look in all the sub directories.
4. Look in the current directory.

Prior to opening a user interface file the current directory is changed to the directory containing the user interface file. This is required since Qt’s Designer saves file references, such as push button icons, with paths relative to the user interface file. After loading the user interface file the current directory is reset.

Paths specified with the -p switch or in the QE_UI_PATH environment variable can end in ‘...’ in which case all the sub directories of the path are searched. For example, assuming /temp/aaa and /temp/bbb exist, -p /temp/... will cause files to be looked for in /temp/aaa and /temp/bbb.

In the QEForm widget, macro substitutions from the slightly misnamed variableSubstitutions property are applied to the user interface file name prior to using the above rules to locate the file. For example, if a QEForm widget ‘uiFile’ property is ‘\$(TYPE)/motorOverview.ui’ and the ‘variableSubstitutions’ property is ‘TYPE=pmac’, then the file to be located will be pmac/motorOverview.ui.

QEGui uses file location rules defined by the QE framework. Refer to [Finding files \(page 27\)](#) for more details.

Saving and restoring configurations

The current layout of GUIs, and many aspects of widgets within the GUIs such as scroll bar positions can be saved and restored. From the ‘File’ menu a user can perform the following save and restore functions:

- Save configuration.
Saves the current configuration with a user specified name. The name of the last configuration read is offered as the default name. The user may also specify that the configuration is to be used when QEGui is started with the –r parameter.
- Restore configuration.
Loads a configuration with a user specified name. The name of the last configuration read or written is offered as the default name.
- Manage configurations.
One or more configurations can be selected and deleted.

By default, all configurations are stored in a file called QEGuiConfig.xml in the current working directory. The QEGui ‘-c’ switch can be used to select a different configuration file.

Opening GUIs

New GUIs can be opened as follows:

- ‘File->New Window’ menu option. Creates a new window and presents the GUI file selection dialog. If the user selects a GUI file (a .ui file) the GUI is opened in the new window.
- ‘File->New Tab’ menu option. Creates a new tab in the current window and presents the GUI file selection dialog. If the user selects a GUI file (a .ui file) the GUI is opened in the new tab.
- From an already opened form in a tabbed window, the form may be reopened as a new form by selecting the “Reopen tab as new window” entry from the tab’s context menu.
Note: this action removes the tab/form from the current form.
- ‘File->Open’ menu option. Presents the GUI file selection dialog. If the user selects a GUI file (a .ui file) the current GUI (if any) is closed and the selected GUI is opened in its place.
- All QE framework buttons (QEPushButton, QERadioButton or QECheckBox) can open new GUIs. Refer to ‘QEPushButton, QERadioButton and QECheckBox’ (page 69) for details.
- ‘File->Recent...’menu option. Create a new window opening a recent GUI file (a .ui file). The path list and macro substitutions that were current when the file first added to the recent file list are used.

Built in forms

QEGui provides several built in forms some of which are shown in Figure 3. These forms can be started from the ‘File’ menu or by right clicking on most QE widgets and include the following:

- PV Properties
- Strip chart
- User Level
- Message Log
- Plotter
- Scratch Pad

Generally, QE widgets can be dragged to these forms.

These forms are implemented using standard QE widgets that are available to a GUI designer. Figure 4 shows a custom GUI using QE widgets that are also used in QEGui’s built in forms.

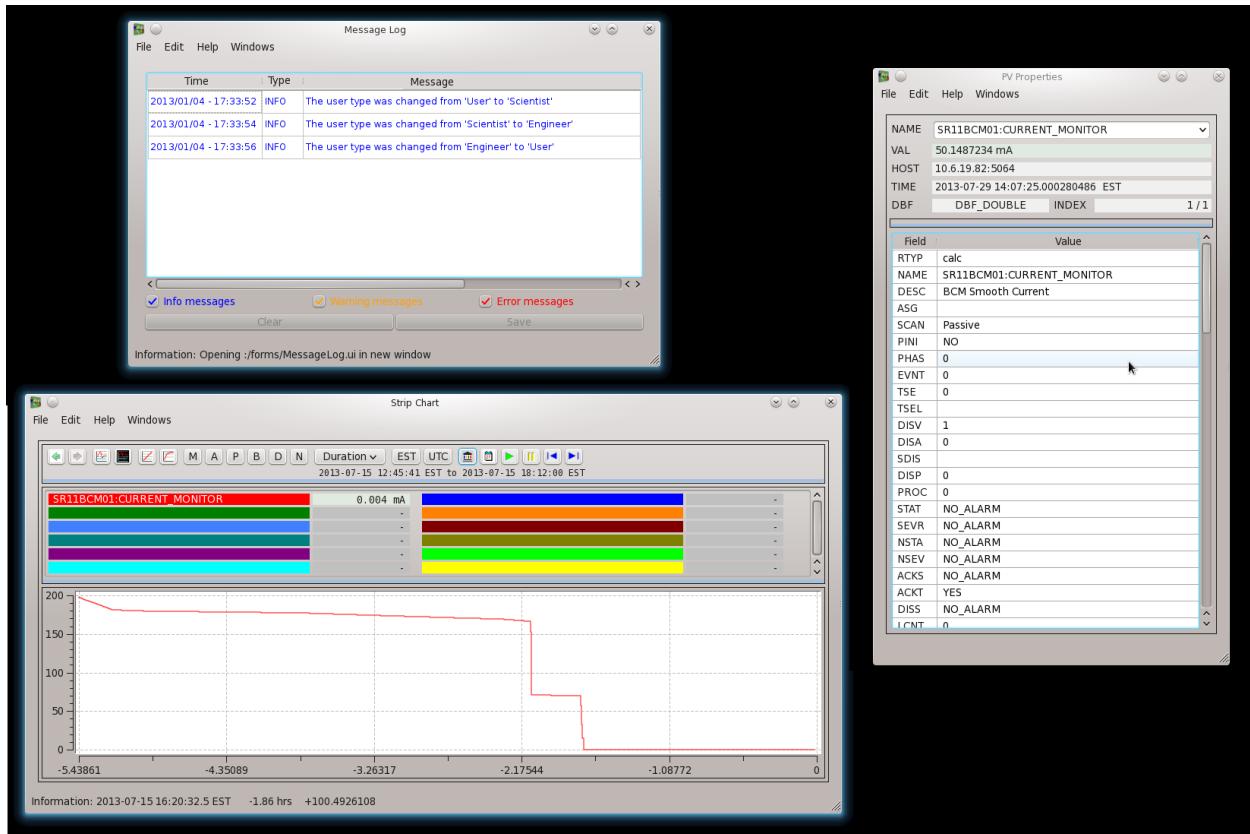


Figure 3 Some of QEGui built in forms

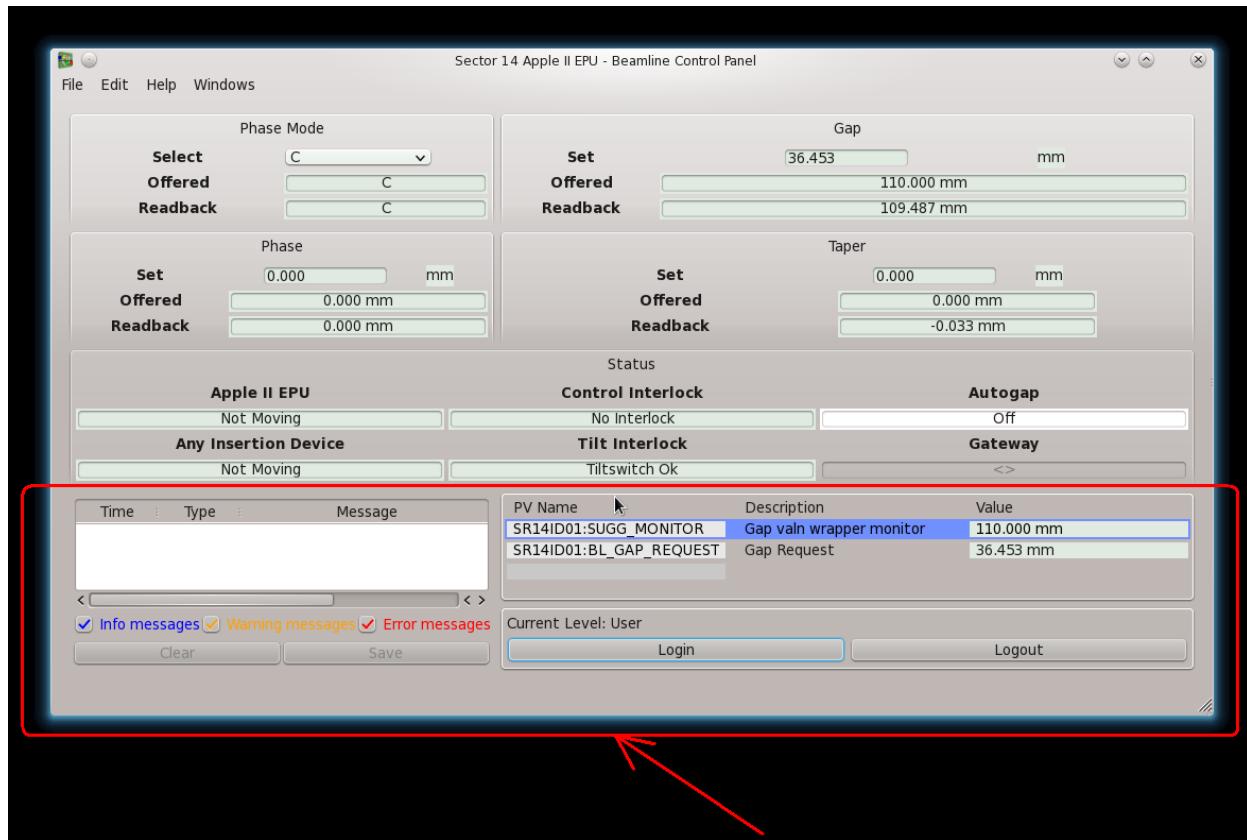


Figure 4 QE widgets used in QEGui's built in forms can be used in any GUI

Editing GUIs

If the 'Edit' menu has been enabled with the '-e' start-up parameter, then the following options may be selected from the 'Edit' menu:

- **Designer...**
Start Qt's designer. This is just a convenient way to start designer.
- **Open CurrentForm in Designer...**
Open the current GUI in Qt's designer. When saved, or when designer is closed, the current GUI will refresh to reflect any changes. This is a simple but powerful integration of QEGui and designer. A user looking at a GUI in QEGui can select this option, modify the GUI, close designer and see the changes with no further action required.
- **Refresh Current Form**
This is a diagnostic option to restart an individual GUI.
- **Set Passwords...**
Display the user level passwords and allow them to be modified. Refer to 'User levels' (page 23) for details on user levels. User level passwords will be saved when QEGui closes. This option is available if the 'Edit' menu is enabled and the 'Edit' menu is intended to only be enabled when a GUI system is being designed. If this model changes, for example if some GUI files are read only and the user is free to edit and create others using the 'Edit' menu then another mechanism for controlling passwords may be required. Note, the QEGui user level passwords are not intended

to be highly secure and are not intended to provide protection from malicious activity. As well as starting QEGui with the –e parameter the user can also view passwords in the QEGui settings file.

Menu bar and tool button customisation

Many graphical applications present a menu bar and tool bar to the user, and QEGui is no exception. The default QEGui menu bar and tool bar, however, focuses on the QEGui application itself and not necessarily on the tasks QEGui is being used to achieve. The QEGui menus and toolbar buttons can be customised to suit the requirements of the GUI solution. In Figure 5 two instances of the QEGui application are shown. Both have the same GUI displayed. The example on the left, however, has customised menus. The customisation includes some entirely new menus and a selection of the default menus. The example on the right includes the default QEGui menu bar.

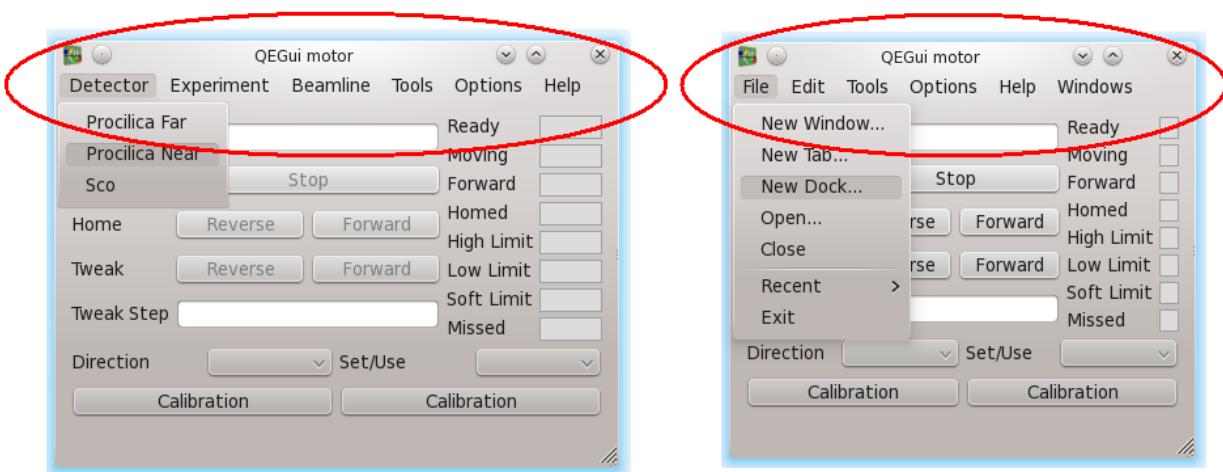


Figure 5 Menu bar customisation

No action is required if customisation is not required. QEGui will start by default with a default menu bar.

Menu bar items and tool bar buttons can be added by the customisation mechanism which can carry out the following functions:

- Open a GUI (a .ui file). The options available are the same as when opening a GUI from a QE button widget, including, open in the current window, in a new window, or in a dock.
- Request the application to perform an action. For example, a menu bar item can be added to ask the application to exit.
- Request a QE widget to perform an action. For example, a menu bar item can be added to ask a QEImage widget to pause image display.
- Run a program. For example, start a web browser.
- Define a placeholder menu that the application can locate and use. For example, the QEGui application looks for a ‘Recent’ placeholder menu. If present QEGui places items in this menu to allow the user to open recently opened GUI files

Customisation is carried out in two general steps:

1) **Define a customisation file containing one or more named sets of customisations.**

A set of customisations defines what buttons appear in what toolbars, what menus are required in the menu bar, what items are in the menus, and what the buttons and menu items do.

The customisation file is loaded when QEGui starts. QEGui loads any customisation file specified on the command line with the –w parameter. If none is specified, QEGui attempts to load QEGuiCustomisation.xml in the current directory. See ‘Command format:’ (page 9) for details.

2) **Request a set of customisations by name when opening QEGui windows.**

This occurs when QEGui starts, or when opening new GUIs. New GUIs can be opened from a menu item or button defined by the customisation itself, or by a QEPushButton widget in a GUI.

The QEGui –n command line parameter is used to specify the default customisation set name.

See ‘Command format:’ (page 9) for details.

Customisation file format

The customisation file contains an XML definition of one or more sets of customisations.

The following XML outlines the syntax. For comprehensive information on the element tags and attributes, refer to the table following the XML.

```
<QEWindowCustomisation>

    <CustomisationIncludeFile>includefile.xml</CustomisationIncludeFile>
    <Customisation Name="name">
        <IncludeCustomisation Name="name" />
        <Menu>
            <Item>
                <Window>
                    <UiFile Name="file.ui" />
                    <Title>Window Title</Title>
                    <MacroSubstitutions>NAM1=v1,NAM2=v2,...</MacroSubstitutions>
                    <CustomisationName>name<CustomisationName>
                    <CreationOption>[Open|NewTab|NewWindow]</CreationOption>
                </Window>
                <Separator/>
            </Item>

            <Item>
                <Dock>
                    <UiFile Name="file.ui" />
                    <MacroSubstitutions>NAM1=v1,NAM2=v2,...</MacroSubstitutions>
                    <CreationOption>[LeftDock|RightDock|TopDock|BottomDock|Floating Dock]</CreationOption>
                    <Hidden/>
                </Dock>
            </Item>
        </Menu>
    </Customisation>
</QEWindowCustomisation>
```

```

<Item>
  <BuiltIn Name="name">
    <WidgetName>name<WidgetName>
  </BuiltIn>
</Item>

<Item>
  <Program Name="name">
    <Arguments>arguments</Arguments>
  </Program>
</Item>
  ...
<Menu>
  ...
</Menu>

</Menu>

<Placeholder Name="name" />

<ToolBar>
  Not defined yet...
</ToolBar>

</Customisation>

<Customisation Name="name">
  ...
</Customisation>

  ...
</QEWindowCustomisation>
  
```

The following table defines the XML elements and tags that may be used to define a customisation file.

Tag name	Element description	Attributes	Child element tags (*Mandatory)
QEWindowCustomisation	A single element with this tag is expected in each customisation file.		CustomisationIncludeFile Customisation

Tag name	Element description	Attributes	Child element tags (*Mandatory)
Customisation	A named set of window customisations.	Name: Used to identify a set of customisations.	Menu Placeholder Toolbar IncludeCustomisation
CustomisationIncludeFile	Name of XML customisation file to include.	Name: Customisation file name	
IncludeCustomisation	Named set of window customisations to add to the current set being defined	Name: Name of customisation set to include.	
Menu	A menu, or sub menu in the window menu bar.	Name: Menu name	MenuItem
Item	A menu item on a menu in the window menu bar.	Name: Menu item name	Window Dock BuiltIn Program Separator
Separator	Element presence signals a menu item is to be preceded by a separator. Element contents ignored		
Window	A GUI to open (in a main window)	UiFile* Title MacroSubstitutions CustomisationName CreationOption	
Dock	A GUI to open (in a dock of the current main window)	UiFile* MacroSubstitutions CreationOption Hidden	
UiFile	GUI file name (.ui file)		
Title	Window title. This overrides any title extracted from the .ui file being presented.		
MacroSubstitutions	Macro substitutions to be passed on to all GUI components.		
CustomisationName	Name of customisation set to be applied to the window		
Hidden	Element presence signals dock is to be created hidden. Element contents ignored		

Tag name	Element description	Attributes	Child element tags (*Mandatory)
CreationOption	<p>Defines how new GUI is presented. If opening the GUI in a window the options are:</p> <ul style="list-style-type: none"> • Open • NewTab • NewWindow <p>If opening the GUI in a dock the options are:</p> <ul style="list-style-type: none"> • LeftDock • RightDock • TopDock • BottomDock • FloatingDock 		
BuiltIn	Name of function built in to the application, or built into a QE widget. If a WidgetName element is not provided, the function is expected to be built in to the application.	Name: Function name	WidgetName
WidgetName	Name of the QE widget being displayed in a GUI that will receive a request for a built in function.		
Program	Command line command.	Name: Program name. For example, firefox	Arguments
Arguments	Command line arguments		
Placeholder	Defines locations that the application can locate by name. For example, QEGui will search for a placeholder called 'Windows' and if found add a 'windows' menu.	Name: placeholder identifier.	
Toolbar	Main window toolbar. Not implemented yet. Will include elements to define toolbar location, name, and buttons		

The following brief example customisation file includes several sets of customisations:

```
<QEWindowCustomisation>
```

```
<Customisation Name="EXIT_ONLY">

    <Menu Name="File">
        <Item Name="Exit" >
            <BuiltIn Name="Exit" />
        </Item>
    </Menu>

</Customisation>

<Customisation Name="MAIN">

    <Menu Name="Detector">

        <Item Name="Procilica Far">
            <Window>
                <Title>Procilica Far</Title>
                <UiFile>detectorOverview.ui</UiFile>
                <MacroSubstitutions>DET=01</MacroSubstitutions>
                <CustomisationName>IMAGING</CustomisationName>
            </Window>
        </Item>

        <Item Name="Procilica Near">
            <Window>
                <Title>Procilica Near</Title>
                <UiFile>detectorOverview.ui</UiFile>
                <MacroSubstitutions>DET=02</MacroSubstitutions>
                <CustomisationName>IMAGING</CustomisationName>
            </Window>
        </Item>

    </Menu>

</Customisation>

<Customisation Name="IMAGING">

    <Menu Name="Imaging">

        <Item Name="Analysis ">
            <Dock>
                <Title>Analysis</Title>
                <UiFile>imageAnalysis.ui</UiFile>
            </Dock>
        </Item>

    </Menu>

</Customisation>
```

```
</Item>

<Item Name="Statistics">
    <Dock>
        <UiFile>imageStatistics.ui</UiFile>
        <Hidden/>
    </Dock>
</Item>

</Menu>

</Customisation>

</QEWindowCustomisation>
```

Default customisations

The default menus in QEGui are implemented using the same customisation mechanism described in this document. On start-up, QEGui loads an internal customisation file which defines a window customisation set named `QEGui_Default`. The set of customisations named `QEGui_Default` is applied if no other named set is specified.

Because the default menus are defined using the customisation mechanism, the customisation set named `QEGui_Default` can be redefined if required.

If a customisation set named `QEGui_Default` is defined in any customisation file loaded, this will override the internal customisation set loaded when QEGui starts.

If you want to create a variation of the default set you might like to start with the original which can be found in the QEGui source code file `QEGuiCustomisationDefault.xml`.

All default menus can be removed by redefining the set as an empty set:

```
<Customisation Name="QEGui_Default"/>
```

Window customisation from a QE button widgets

A GUI can be started from QEPushButton, QECheckBox, or QERadioButton widgets. The QE button widgets contains properties to define the GUI to load (the .ui file), how it is to be presented (as a new window, a tab, a dock, etc) and what customisation set is to be applied to the window containing it. The `customisationName` property is used to specify the window customisation set to apply to the window containing the GUI started by the QE button widget. The customisation does not apply when launching a GUI as a dock.

Sepators in menus

Note, separators in menus are considered an attribute of the menu item following the separator.

Built-in functions

An application, such as the QEGui application, and individual QE widgets can provide a built-in functions that can be specified in a customisation file.

The QEGui application provides the following built-in functions:

- PV Properties...
- Strip Chart...
- Scratch Pad...
- Message Log...
- Plotter...
- PV Load/Save...
- Archive Status...
- New Window...
- New Tab...
- New Dock...
- Open...
- Close
- Save Configuration...
- Restore Configuration...
- Manage Configuration...
- User Level...
- Exit
- Open Designer...
- Open Current Form In Designer...
- Refresh Current Form
- Set Passwords...
- About...

Refer to QE widgets in this document to see what built in functions each provides.

The following example item element defines a menu item which will ask the application to exit:

```
<Item Name="Exit">
    <BuiltIn Name="Exit" />
</Item>
```

The following example item element defines a menu item which will ask a QEImage widget named BeamImage to pause image display:

```
<Item Name="Pause">
    <BuiltIn Name="Pause" >
        <WidgetName>BeamImage</WidgetName>
    </BuiltIn>
</Item>
```

Repeating sections of a set of window customisations

Identical menus may be repeated in multiple sets of customisations. For example, two sets of customisations may be defined for various GUIs, but both require the same 'Tools' menu. Even though

only two customisation names are required when starting the GUIs, a third may be defined for inclusion in the others as follows:

```
<QEWindowCustomisation>

    <Customisation Name="TOOLS">
        <Menu Name="Tools">
            ...
        </Menu>
    </Customisation>

    <Customisation Name="MAIN">
        <Menu Name="...">
            ...
        </Menu>
        <Menu Name="...">
            ...
        </Menu>
        <IncludeCustomisation Name="TOOLS"/>
    </Customisation>

    <Customisation Name="SUB">
        <Menu Name="...">
            ...
        </Menu>
        <Menu Name="...">
            ...
        </Menu>
        <IncludeCustomisation Name="TOOLS"/>
    </Customisation>

</QEWindowCustomisation>
```

Tricks and tips (FAQ)

GUI titles

The QEGui application reads the `windowTitle` property of the top level widget in a user interface file. It then applies any macro substitutions to the name and uses it as the GUI title. Figure 6 shows a `windowTitle` property that includes macros being edited in Designer, with the same user interface being displayed by QEGui with the appropriate macro substitution.

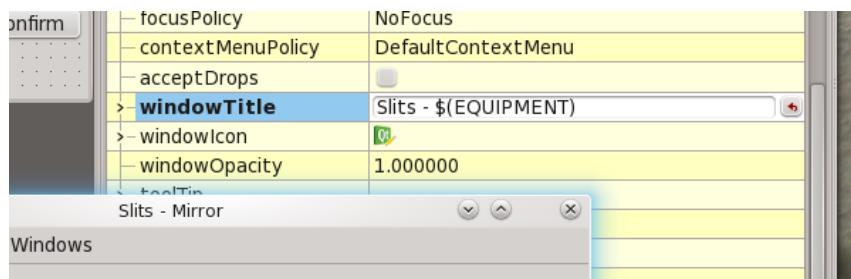


Figure 6 windowTitle Property in designer with actual translated window title on form in foreground

User levels

The QE framework manages three application wide user levels. These are independent of the operating system user accounts.

Within an application using the QE framework (such as the QEGui application), one of three user levels can be set. The three user levels are:

- **User**
- **Scientist**
- **Engineer**

User levels allow the most appropriate view of the system to be presented to different user groups. In Figure 7 for example, while in User mode operational information (beam current) is large. In Scientist mode a ‘maintenance’ panel appears but a maintenance control is not enabled. In Engineer mode, the maintenance control is enabled.

To avoid the annoyance of widgets disappearing while you are trying to design a GUI, widgets will not become ‘not visible’ due to user level while being edited in Qt Designer. This also applies to Designer’s ‘preview’ mode. To check if a widget’s visibility is changes correctly according to user level, open the GUI using the QEGui application.

While the user level can be set and read programmatically using the ContainerProfile class, it is intended to be set using the QELogin widget and acted on by other QE widgets. The QELogin widget imposes a hierarchy to the user levels, requesting passwords when increasing user levels but allowing the user level to be reduced without authority. Refer to ‘QELogin’ (page 57) for details of how passwords are set.

The user levels are used to control individual QE widget behaviour. Most commonly, user level is used to determine if a QE widget is visible or enabled for a given user level through the ‘userLevelVisibility’ and ‘userLevelEnabled’ properties respectively. The ‘userLevelUserStyle’, ‘userLevelScientistStyle’ and ‘userLevelEngineerStyle’ properties, however, allow any style string to be applied for each user level. While user level based style strings allow many simple and convenient user interface changes beyond visibility and enabled state, they can also allow obscure and bizarre behaviour changes. For example, a style string may simply set a QEPushButton background to red in user mode, alternatively a style string could be used to move a QEPushButton to a different location on a form.

The syntax for all Style Sheet strings used by this class is the standard Qt Style Sheet syntax. For example, 'background-color: red'. Refer to Qt Style Sheets Reference for full details. The style sheet syntax includes a 'qproperty' keyword allowing any property to be altered using the style string. For example, 'qproperty-geometry:rect(10 10 100 100);' would move a widget to position 10,10 and give it a size of 100,100.

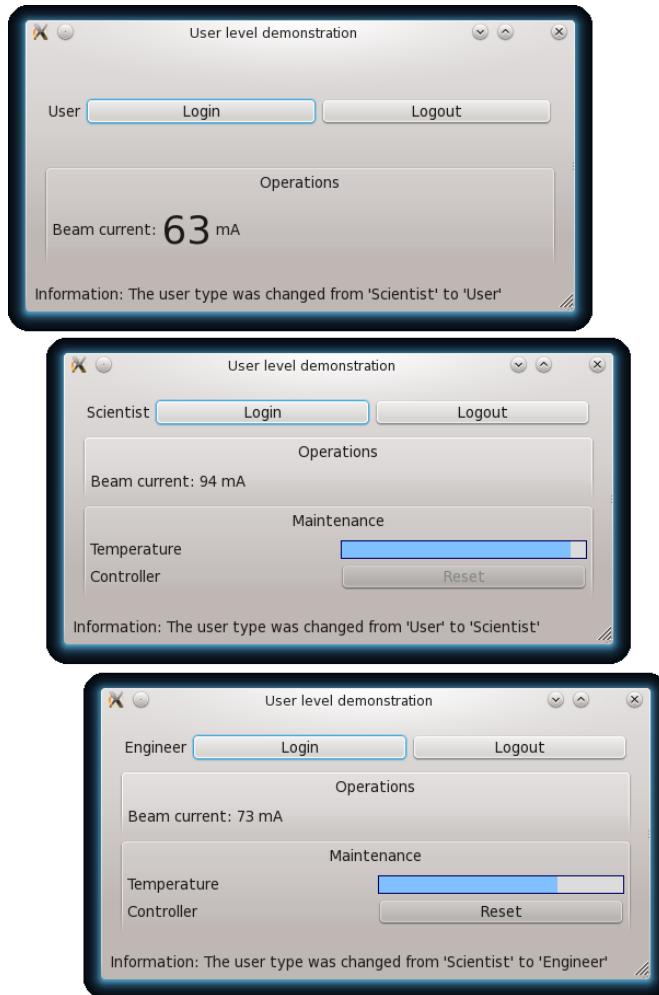


Figure 7 User level example

Logging

Several QE widgets generate log messages. These can be caught and displayed by a QELog widget, or a user application. This section describes the overall QE framework message logging system. Refer to ‘QELog’ (page 55) for a description of the QELog widget.

Log messages have three attributes:

1. the message text itself;
2. its severity (information, warning or error); and
3. the message kind, which defines the class or type of message. It may be set to one, one or both of:
 - a. event – used of significant system events. These can be displayed by the QELog widget as described below; and/or
 - b. status – used for transient status information, such the time/value coordinates associated with the cursor when moving over the plot area of the QEStripChart widget. When running within in QEGui, this class of message are displayed on the form’s status bar.

Simplest use:

The simplest use of this system is to drop a QELog widget onto a form. That’s it. Any log messages generated by any QE widgets within the application (for example, the QEGui application) will be caught and displayed provided that the message kind specifies the event attribute. Figure 8 shows a form containing a QELogin widget and a QELog widget. When the user logs in using the QELogin widget, messages generated by the QELogin widget are automatically logged by the QELog widget.

The messages generated by the QELogin widget are denoted as both status and event, and so also shown on the status bar at the bottom of the form.



Figure 8 Simple logging example

Complex use:

By default, QELog widgets catch and display any message, but messages can be filtered to display only messages from a specific sets of QE widgets or a to display messages originating from QE widgets within the same QEForm containing the QELog widget.

A form may contain QEForm widgets acting as sub forms. A QELog widget in the same form as a QEForm widget can catch and display messages from widgets in the QEForm if the QEForm is set up to catch and re-broadcast these messages. QEForm widgets can catch and filter messages exactly like QELog widgets, but selected messages are not displayed, rather they are simply re-broadcast as originating from themselves. When a QELog widget is selecting messages only from QE widgets in the same form it is in it will catch these re-broadcast messages

The messageFormFilter, messageSourceFilter, and messageSourceId properties are used to manage message filtering as follows:

Any QE widget that generates messages has a messageSourceId property. QELog and QEForm widgets with the messageSourceId property set to the same value can then use the messageSourceFilter property to filter messages based on the message source ID as follows:

- **Any** A message will always be accepted. (messages source ID is irrelevant)
- **Match** A message will be accepted if it comes from a QEWidget with a matching message source ID.
- **None** The message will not be matched based on the message source ID. (It may still be accepted based on the message form ID.)

All generated messages are also given a message form ID. The message form ID is supplied by the QEForm the QE widget is located in (or zero if not contained within a QEForm widget). QELog and QEForm widgets with a matching message form ID can then use the messageFormFilter property to filter messages based on the message form ID as follows:

- **Any** A message will always be accepted.
- **Match** A message will be accepted if it comes from a QE widget on the same form.
- **None** The message will not be matched based on the form the message comes from. (It may still be accepted based on the message source ID.)

Note: The internal archive access manager object also generates log messages. As this object is not a widget, the value of its messageSourceId is not modifiable as a property and has been hard-coded as 9,001. The range 9,001 to 10,000 is reserved for internal framework use, and while one is not prohibited from allocating these numbers to widgets within, it is not recommended.

Figure 9 shows a complex logging example. The main form contains two sub forms and a QELog widget. The right hand sub form looks after its own messages. It has a QELog widget with filtering set to catch any messages generated on the same form. The left hand sub form does not display its own messages, but the form is set up to re-broadcast any messages generated by QE widgets it contains, so the QELog widget on the main form can be set up to catch and display these messages. Note, the QEGui application itself also uses a UserMessage class to catch and present the same messages on its status bar.

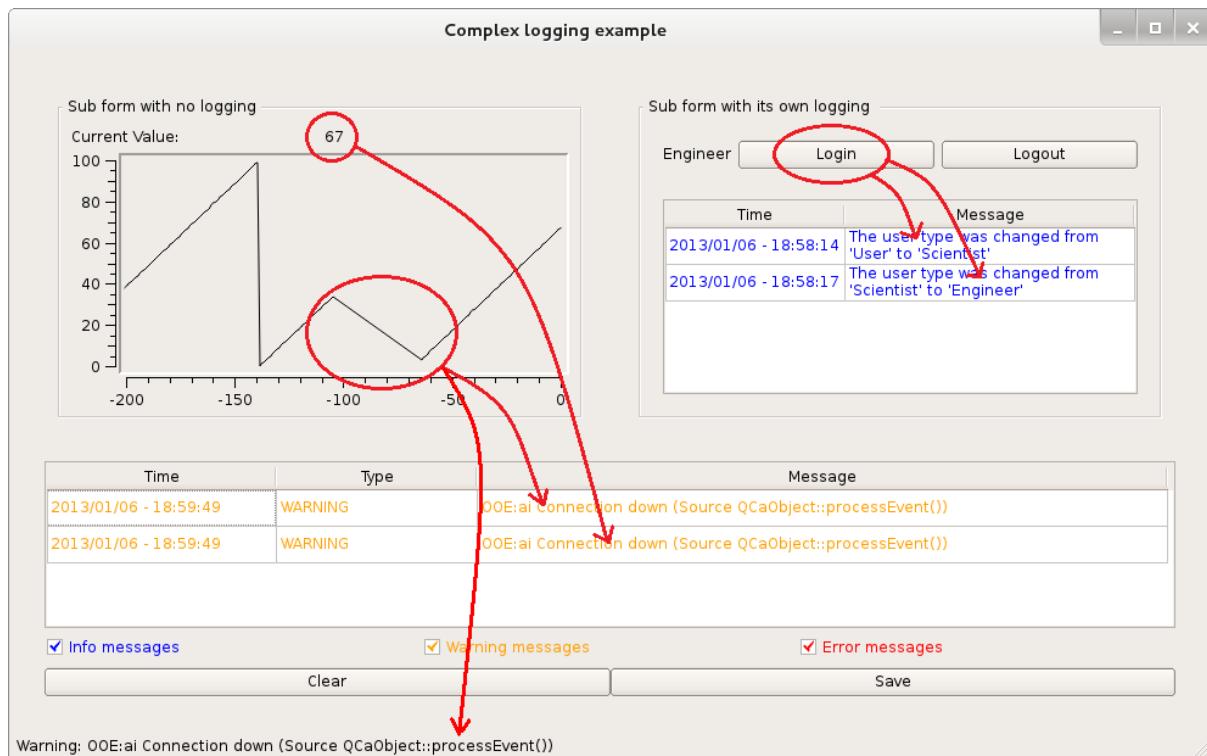


Figure 9 Complex logging example

Note, Application developers can catch messages from any QE widgets in the same way the QELog and QEForm widgets do, by implementing a class based on the UserMessage class. See the UserMessage class documentation for details.

Finding files

The QE widgets uses a consistent set of rules when locates files. File names can be absolute, relative to the path of the QEform in which the QE widget is located, relative to the any path in the path list published in the ContainerProfile class or in the QE_UI_PATH environment variable, or relative to the current path.

See QEWidget:: findQEFile() in QEWidget.cpp for details on how the rules are implemented.

In the GEQui application, the `-p` switch is used to specify a path list which is published in the ContainerProfile class. In the QEForm widget, macro substitutions can be used to modify file names. Refer to 'File location rules' (page 6) details on how the GEQui application and QEForm widgets search for a user interface file given absolute and relative file paths.

Sub form file names

Absolute names simplify locating forms, but make a set of related GUI forms and sub forms less portable. The following rules will help make a set of forms and sub forms more portable.

- No path should be specified for sub forms in the same directory as the parent form.
- A relative path should be given for sub forms in a directory under the parent form directory.

- Paths to directories containing generic sub forms can be added to the -p switch.

Refer to ‘File location rules’ (page 6) details on how QEGui searches for a user interface file given absolute and relative file paths.

Sub form resizing

QEForm widgets are used to embed sub forms in a user interface by loading a Qt user interface (.ui) file into itself at run time. Each QEForm widget has a set of properties (inherited from QWidget) that define how resizing is managed including geometry, size policy, maximum, minimum, and base sizes, size increments and margins. The top level widget in the .ui file loaded by a QEForm also contains these properties. A conflict may exist if the size related properties of the QEForm are not the same as the size related properties of the top level widget in the .ui file the QEForm is loading.

This conflict can be resolved with the resizeContents property of the QEForm. If resizeContents is true, the size related properties of the top level widget in the .ui file are adjusted to match the QEForm. If resizeContents is false, the size related properties of the QEForm are adjusted to match the top level widget in the .ui file.

Refer to QEForm (page 97) for complete details about the QEForm widget

Ensuring QERadioButton and QECheckBox is checked if it matches the current data value

When a data update matches the checkText property, the Radio Button or Check Box will be checked.

If the ‘format’ property is set to ‘Default’ (which happens to be the default!), and the data has enumeration strings then the checkText property must match any enumeration string.

This can cause confusion if the values written are numerical – the click text (the value written) can end up different to the clickCheck text. Also, if the enumeration strings are dynamic, it is not possible to specify at GUI design time what enumeration strings to match.

To solve this problem, set the ‘format’ property to ‘Integer’ and set the ‘checkText’ property to the appropriate integer value. Remember, the checkText property is a text field that will be matched against the data formatted as text, so the checkText property must match the integer formatting. For example, a checkText property of ‘ 2’ (includes spaces) will not match ‘2’ (no spaces)

What top level form to use

If you are using Qt’s Designer to lay out user interfaces as part of an application you are developing, then the top level form you start with will depend on your application, but if you are creating a user interface file for use in QEGui the following guidelines apply:

QEGui can load a user interface file with any sort of top level widget, but the most appropriate is likely to be one of the simpler containers such as QWidget as QEGui can manage most aspects more complex containers such as are designed to manage, such as scrolling.

You select the top level widget when you create a new user interface in Designer. It is recommended that you choose QWidget, but if there is functionality you require provided by other widgets, then feel free to use any other widget. For example if you are designing a sub-form with a border you may chose a QFrame as the top level widget. If you have some specific scrolling requirements you may choose a QScrollArea widget

QEGui opens all user interface files using a QEForm widget. If the user interface file it is opening does not have a layout, the top level widget in the user interface file is resized to match the QEForm.

If it does have a layout, then the QEForm will also have given itself a layout to ensure layout requests are propagated and the top level widget is not resized.

GUI based on a QScrollArea won't scroll in QEGui

In designer, uncheck the widgetResizable property in the QEScrollArea. This behaviour is a function of the QScrollArea widget and can be observed by opening the qui in Designer's preview mode.

QEGui can open a .ui file with a QScrollArea as the top level widget. QEGui notices the top level widget in the .ui file is a scroll area and will not impose its own scrolling. If the widgetResizable property is checked, the widget resizes as the QScrollArea is resized to fit so the scroll bars never appear.

How does a user interact with an updating QE widget

Most QE widgets that a user can use to write to an EPICS database can also be set to subscribe to the variable it controls and display its current value. In fact this is generally the default. Updating of the widget is stopped, however, whenever a widget has focus. This means updates will not cause the widget to change what it is displaying while the user is interacting with it.

If a separate readback value is preferred, the control widget's 'subscribe' property can be cleared and a read only widget such as a QELabel can be added beside the control widget.

Widgets disappear when escape is pressed!

A QDialog widget has been used as the top level form. Use a QWidget instead. A QDialog will work as a QEGui form, but a feature of a QDialog is that the escape key causes it to close. The main task of the QEGui application is to load .ui files. Apart from a small amount of introspection to determine if the loaded form will be managing its own resizing QEGui does not know or care what the top level widget in the .ui file is. You may have used a QDialog widget as the top level form simply because this was the default Qt's designer offered. Refer to 'What top level form to use' (page 28) for selecting the best top level widget.

A QE widget displays the correct alarm state only when a form is first opened

Most QE widgets display a variable's alarm state by default. The alarm state represents the state of the variable's value (.VAL) and is unrelated to most other fields. Channel Access will provide the current alarm state with any field but will only supply updates for the .VAL field when the alarm state changes. For example, if a QELabel widget displays MY_MOTOR.VAL and another displays MY_MOTOR.DIR, both will display the current alarm state when first created, but only the QELabel displaying MY_MOTOR.VAL will display a change in the alarm state because only that widget will receive a CA update.

To avoid this problem, uncheck ‘displayAlarmState’ for QE widgets that are displaying a field unrelated to the alarm state.

A QEPlot widget is not displaying updates

If the time on the machine running the QEPlot widget is ahead of the time on the machine generating the data by more than the time span, the QEPlot widget will not display the data. For example, if the QEPlot widget is displaying the last minute’s data and an update arrives for data two minutes ago (from a machine with the time set two minutes behind), the QEPlot widget will discard the update along with any other values earlier than the time span being presented.

Droppable widgets as scratch pads and customisable GUIs

Most QE widgets have a ‘droppable’ property. If this is set, variable names may be dropped from other QE widgets and the widget will connect to the new variable. Using this, a GUI designer can add a customisable area of a GUI or add a specialised scratch pad area. For example a corner of a GUI may be reserved for dropping variables of interest. Since most QE widgets are droppable, some of the scratch pad area may include control widgets such as QELlineEdit or QESlider.

Note, The QEGui application provides a built in scratch pad form (refer to ‘Built in forms’ (page 12) for details). The QEScratchPad widget used within this built in forms is also available to be added into any GUI. It provides a tabular area for dropping variable names and adds a description column.

QE widgets

QE widgets enable the design of control system user interfaces.

This document describes what the widgets are designed to do, what features they have and how they should be used. For a comprehensive list of properties, refer to the widget class documentation in [QE_ReferenceManual.pdf](#)

EPICS enabled standard Qt widgets:

Many QE widgets are simply standard Qt widgets that can generally read and write to EPICS variables. For example, a QELLabel widget is basically a QLabel widget with a variable name property. When a variable name is supplied, text representing the variable is displayed in the label.

The QE Framework also manages variable status using colour, provides properties to control formatting, etc

Control System widgets

Other QE widgets implement a specific requirement of a Control System. For example QEPlot presents waveforms. These widgets are still based on standard low level Qt widgets so still benefit from common Qt widget properties for managing common properties such as geometry.

Common QE Widget properties

Properties of base Qt widgets are not documented here – refer to Qt documentation for details.

variableName and variableSubstitutions

All EPICS aware widgets have one or more variable name properties. The variable names may contain macro substitutions that will be translated when a user interface is opened. The same variable name macro substitutions are used by many widgets for translating macros in other text based properties as well. For example, QEPushbutton uses the macro substitutions in the GUIFile property.

Generally the macro substitutions will be supplied from QEGui application command line parameters, and from parent forms when a user interface is acting as a sub form. The widget itself may have default macro substitutions defined in the ‘variableSubstitutions’ property. Default macro substitutions are very useful when designing user interface forms as they allow live data to be viewed when designing generic user interfaces. For example, a QELabel in a generic sub form may be given the variable name SEC\${SECTOR}:PMP\${PUMP} and default substitutions of ‘SECTOR=12 PUMP=03’. When used as a sub form valid macro substitutions will be supplied that override the default substitutions. At design time, however, the QELabel will connect to and display data for SEC12:PMP03. Note, default substitutions can be dangerous if they are never overridden.

The following example describes a scenario where macro substitutions required for a valid variable name are defined at several levels, and in one case multiple levels.

Figure 10 shows a form containing a QELabel. The variable name includes macros SECTOR, DEVICE and MONITOR. Default substitutions are provided for MONITOR. This is not adequate to derive a complete variable name.

Figure 11 shows a form using the form from Figure 10 as a sub form. Additional macro definitions for SECTOR and DEVICE are provided with the sub-form file name. When the sub form is loaded, the QELabel in the sub form can now derive a complete variable name (SR01BCM01:CURRENT_MONITOR). While complete, this is not actually functional – the correct sector is SR11.

Figure 12 shows the form from Figure 11 opened by the QEGui application with the following parameters:

```
qegui -m "SECTOR=11" example.ui
```

The MONITOR macro has been overwritten, so the QELabel in the sub form now derives the correct variable name SR11BCM01:CURRENT_MONITOR.

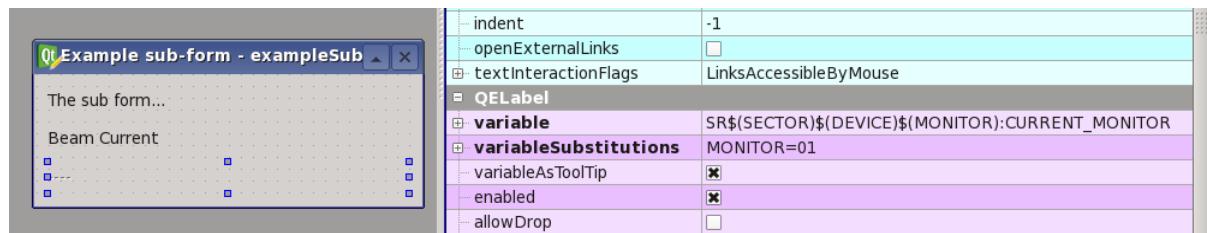


Figure 10 Sub form with macro substitution for part of the variable name

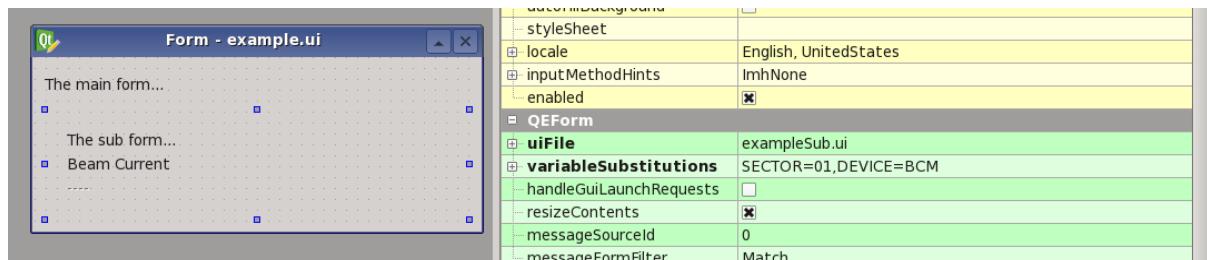


Figure 11 Main form containing sub form with all macro substitutions satisfied (but one is incorrect)

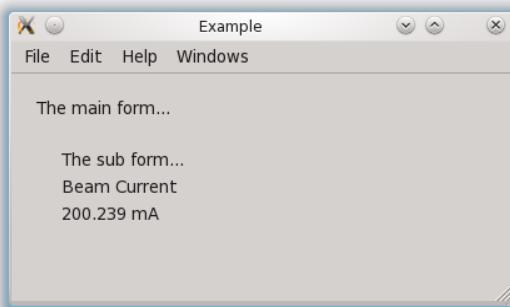
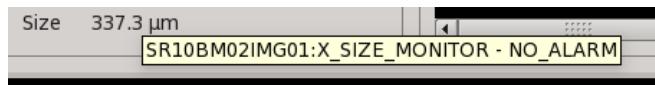


Figure 12 QEGui displaying form and sub forms with all macro substitutions satisfied correctly

variableAsTooltip

If checked, the ToolTip is generated dynamically from the variable name or names and status.



subscribe

If checked, the widget will subscribe for data updates and display them. This is true by default for display QE widgets as QLabel. For control widgets it may be false by default. For example it is false by default for QPushButton since it is more common to have static text in the button label, but it can be set to true if the button text should be a readback value, or if the button icon is to be updated by a readback value.

enabled

Set the preferred 'enabled' state. Default is true.

The standard Qt 'enabled' property is set false by many QE widgets to indicate if the data displayed is invalid (disabled). When the data displayed is valid, the QE widget will reset standard Qt 'enabled' property to the value of this 'enabled' property. Users wanting to enable or disable a QE widget for other purposes should use this property. This property will be used to set the standard Qt 'enabled' property except when data is invalid.

allowDrop

Allow drag/drops operations to this widget. Default is false. Any dropped text will be used as a new variable name.

visible

Display the widget. Default is true. Setting this property false is useful if widget is only used to provide a signal - for example, when supplying data to a QELink widget.

Note, when false the widget will still be visible in Qt Designer.

messageSourceId

Set the ID used by the message filtering system. Default is zero.

Widgets or applications that use messages from the framework have the option of filtering on this ID.

For example, by using a unique message source ID a QELog widget may be set up to only log messages from a select set of widgets.

Refer to Logging (page 25) for further details.

userLevelUserStyle, userLevelScientistStyle, userLevelEngineerStyle

Style Sheet strings to be applied when the widget is displayed in 'User', 'Scientist', or 'Engineer' mode. Default is an empty string.

The syntax is the standard Qt Style Sheet syntax. For example, 'background-color: red'

This style strings will be safely merged with any existing style string supplied by the application environment for this widget, or any style string generated for the presentation of data.

Refer to User levels (page 23) for details regarding user levels.

userLevelVisibility

Lowest user level at which the widget is visible. Default is 'User'.

Used when designing GUIs that display more detail according to the user mode.

The user mode is set application wide through the QELogin widget, or programmatically through setUserLevel().

Widgets that are always visible should be visible at 'User'.

Widgets that are only used by scientists managing the facility should be visible at 'Scientist'.

Widgets that are only used by engineers maintaining the facility should be visible at 'Engineer'.

Refer to User levels (page 23) for details regarding user levels.

userLevelEnabled

Lowest user level at which the widget is enabled. Default is 'User'.

Used when designing GUIs that allows access to more detail according to the user mode.

The user mode is set application wide through the QELogin widget, or programmatically through `setUserLevel()`

Widgets that are always accessible should be visible at 'User'.

Widgets that are only accessible to scientists managing the facility should be visible at 'Scientist'.

Widgets that are only accessible to engineers maintaining the facility should be visible at 'Engineer'.

Refer to User levels (page 23) for details regarding user levels.

displayAlarmState

If true (default) the widget will indicate the alarm state of any variable data is displaying. Typically the background colour is set to indicate the alarm state.

Note, this property is included in the set of standard properties as it applies to most widgets. It will do nothing for widgets that don't display data.

String formatting properties

Many QE widgets present data as text, or interpret text and write data accordingly. Examples are QELabel and QELineEdit.

Common formatting properties are used for all these widgets where possible. Not all are relevant for all data types.

precision

Precision used when formatting floating point numbers. The default is 4.

This is only used if the 'useDbPrecision' property is false.

useDbPrecision

If true (default), format floating point numbers using the precision supplied with the data.

If false, the 'precision' property is used.

leadingZero

If true (default), always add a leading zero when formatting numbers.

trailingZeros

If true (default), always remove any trailing zeros when formatting numbers.

addUnits

If true (default), add engineering units supplied with the data.

localEnumeration

An enumeration list used to data values. Used only when the 'format' property set to 'local enumeration'.

The data value is converted to an integer which is used to select a string from this list.

Format is:

```
[ [<|<=|=|!=|>|=]value1|*] : string1 , [ [<|<=|=|!=|>|=]value2|*] :  
string2 , [ [<|<=|=|!=|>|=]value3|*] : string3 , ...
```

Where:

- < Less than
- <= Less than or equal
- = Equal (default if no operator specified)
- >= Greater than or equal
- > Greater than
- * Always match (used to specify default text)

Rules are:

- Values may be numeric or textual
- Values do not have to be in any order, but first match wins
- Values may be quoted
- Strings may be quoted
- Consecutive values do not have to be present.
- Operator is assumed to be equality if not present.
- White space is ignored except within quoted strings.
- \n may be included in a string to indicate a line break

Examples:

- 0:Off,1:On
- 0 :"Pump Running", 1 :"Pump not running"
- 0:"", 1:"Warning!\nAlarm"
- <2:"Value is less than two", =2:"Value is equal to two", >2:"Value is greater than 2"
- 3:"Beamline Available", *:""
- "Pump Off":"OH NO!, the pump is OFF!","Pump On":"It's OK, the pump is on"

The data value is converted to a string if no enumeration for that value is available.

For example, if the local enumeration is '0:off,1:on', and a value of 10 is processed, the text generated is '10'.

If a blank string is required, this should be explicit. for example, '0:off,1:on,10:'''

A range of numbers can be covered by a pair of values as in the following example:

- >=4:"Between 4 and 8",<=8:"Between 4 and 8"

The QELabel widget will parse the results of the local enumeration searching for embedded style hints. Any text within <angle brackets>.will be applied to the widget's style rather than displayed as text as shown in the following table:

Text supplied to QELabel for display	How QELabel displays them
<background-color: red>Engineering Mode	Engineering Mode
<color: red>not selected	not selected

The format of the style text is standard Qt style syntax.

format

This property indicates how non textual data is to be converted to text:

- Default Format as best appropriate for the data type.
- Floating Format as a floating point number
- Integer Format as an integer
- UnsignedInteger Format as an unsigned integer
- Time Format as a time, source value interpreted as seconds.
- LocalEnumeration Format as a selection from the 'localEnumeration' property

radix

Base used for when formatting integers. Default is 10 (duh!)

notation

Notation to use when formatting data as a floating point number. Default is Fixed. Options are:

- Fixed Standard floating point. For example: 123456.789
- Scientific Scientific representation. For example: 1.23456789e6
- Automatic Automatic choice of standard or scientific notation

arrayAction

This property defines how array data is formatted as text. Default is ASCII. Options are:

- **Append** Interpret each element in the array as an unsigned integer and append string representations of each element from the array with a space in between each. For example, an array of three numbers 10, 11 and 12 will be formatted as '10 11 12'.
- **Ascii** Interpret each element from the array as a character in a string. Translate all non printing characters to '?' except for trailing zeros (ignore them). For example an array of three characters 'a' 'b' 'c' will be formatted as 'abc'.
- **Index** Interpret the element selected by setArrayIndex() as an unsigned integer. For example, if arrayIndex property is 1, an array of three numbers 10, 11 and 12 will be formatted as '11'.

arrayIndex

Index used to select a single item of data for formatting from an array of data. Default is 0.

Only used when the arrayAction property is INDEX. Refer to the ‘arrayAction’ property for more details.

QEAnalogIndicator and QEAnalogProgressBar

The QEAnalogIndicator widget is used to simulate an analog indicator such as a bar indicator or dial. It is not EPICS aware.

The QEAnalogProgressBar is based on the QEAnalogIndicator and is EPICS aware.

Features include:

- Logarithmic or linear scale
- Optional units
- Same widget used for multiple analog indicators including dial and bar.
- Based on QEAnalogIndicator which is available for non EPICS aware uses.
- Alarm Limits are represented on the scale if required

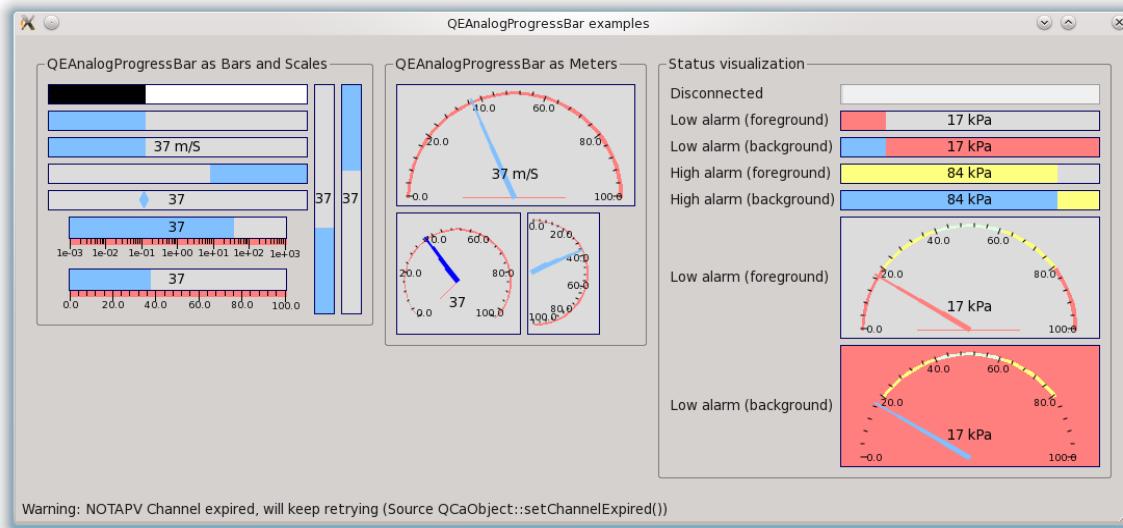


Figure 13 QEAnalogProgressBar examples

QEArchiveStatus

The QEArchiveStatus widget is a non EPICS aware widget that provides status regarding the selected archive hosts together with process variable information retrieved from each Channel Access archive. It inherits directly from QEFrame (refer to QEFrame and QEGroupBox), and as such it provides user level enabled and user level visibility control to the frame, but note it is not a container, i.e. other widgets may not be dropped into a QEArchiveStatus object from within designer.

Archive Status Summary						
Host:Port	End Point	Status	Available	Read	Num PVs	
cr01arc01:80	/cgi-bin/ArchiveDataServer.cgi	Unknown	0	0	0	
cr01arc02:80	/cgi-bin/ArchiveDataServer.cgi	Unknown	0	0	0	

Figure 14 QEArchiveStatus example

QBitStatus and QEBitStatus

The QBitStatus widget is used to present a selected set of bits from a data word. It is not EPICS aware. The QEBitStatus widget is based on QBitStatus and is EPICS aware.

Bits are presented as an array of rectangles or circles with presentation properties to control shape, size, orientation, spacing and colour. Other properties allow bit by bit selection of what values display as ‘on’ and ‘off’ and if bits are rendered when ‘on’ or ‘off’.

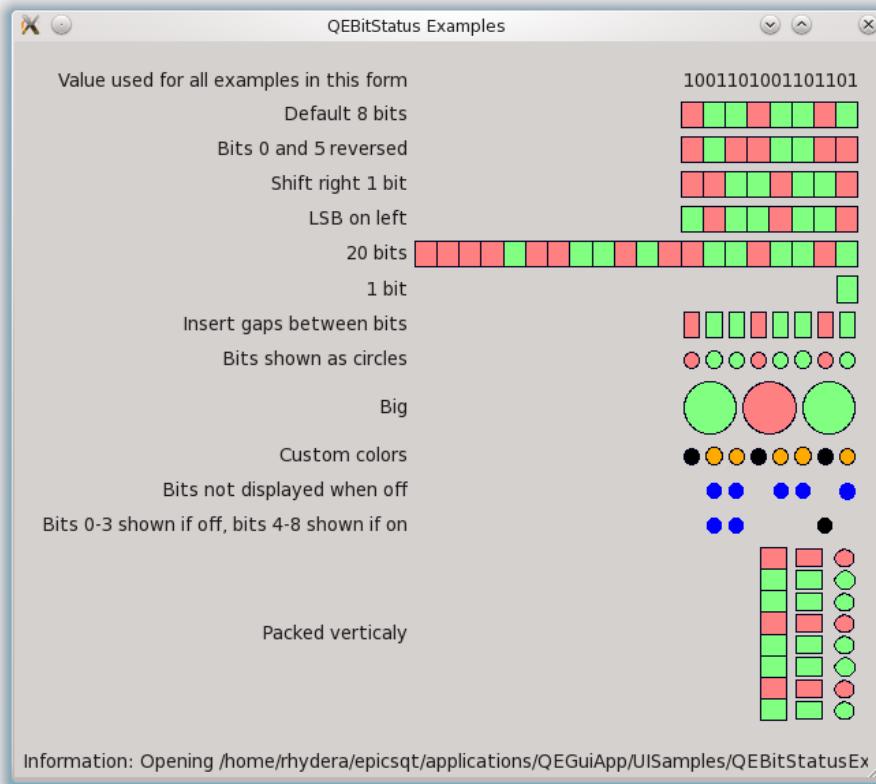


Figure 15 QEBitStatus widget examples

QEComboBox

The QEComboBox widget provides the ability to display and modify the value of a single PV using a combo box. This widget is derived from QComboBox. The example in Figure 16 shows QEComboBox widgets connected to an mbbi record. This widget is primarily intended for presenting a variable with enumeration strings defined for each value. Typically, the enumeration strings are defined in the database and will be used by the QEComboBox if the ‘useDbEnumerations’ property is set (the default). If the ‘useDbEnumerations’ property is not set, then the strings used by the combo box for each variable value must be set up in the QEComboBox at design time. This is done by modifying the localEnumeration property (see String formatting properties, localEnumeration for details).

Warning: while using Qt’s designer you can right click over a QEComboBox and select ‘Add Items’ to add the combo box strings. However at run time, the combo box string will be reset when the widget receives its first update (to either the database enumeration values or the localEnumeration property values).

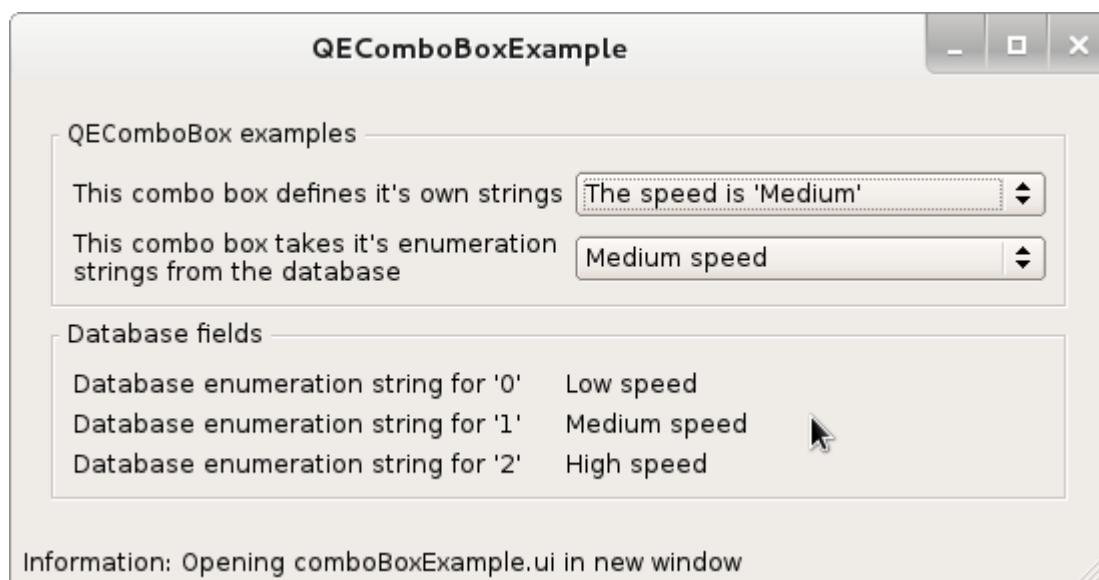


Figure 16 QEComboBox example showing local and database defined enumeration strings

QEConfiguredLayout

The QEConfiguredLayout presents a tabular layout of QE widgets, including button, combo box, label and line edit widgets based on an xml definition stored within the widget, or in a file that can be shared by multiple widgets. It provides similar functionality to a sub form without the need to design and maintain a suitable tabular sub form. The XML defining the layout contains the definition for the rows and columns. Since a change to the row definition affects all columns and a change to a column definition affects all rows, the layout of widgets in a QEConfiguredLayout is always consistent.

<example of xml to be included>

<image of QEConfiguredLayout example to be included>

QEFileBrowser

The QEFileBrowser widget is currently under development. <General description to be added>

QEForm

The QEForm widget is used to present a Qt user interface (.ui) file. While an application can programmatically achieve this by opening a .ui file with a QFile class and loading the contents using the QUiLoader, the QEForm widget adds the following functionality:

- The QEForm uses consistent rules for locating the file common to all QE widgets that access files. Refer to [Finding files \(page 27\)](#) for details.
- The contents of a QEForm is dynamic and can be changed by changing the ‘uiFile’ property.
- The .ui file used to generate the contents of a QEForm is monitored and re-loaded if it changes.
- The QEForm can be used as a sub form. Forms can share common sub forms. Sub forms can be nested.
- The QEForm uses macro substitutions. This means a form can contain multiple instances of the same sub form, each with a different set of macro substitutions. For example, a form displaying a set of slits could use an identical sub form for each motor.

QEForms help manage messages emitted by QE widgets. Messages can be filtered and displayed based on the QEform they reside in. Refer to [Logging \(page 25\)](#) for details.

- The .ui file loaded by a QEForm widget will have a top level widget with size and layout policies that may differ to those of the QEForm. To minimise any confusion, the QEForm widget ensures the top level widget loaded and itself share the same size and layout policies. By default the QEForm widget sets the top level widget loaded to match itself, but this behaviour can be reversed. The ‘resizeContents’ property controls this behaviour. If true, the top level widget loaded is set to match the QEForm. If false, the QEForm is set to match the top level widget loaded.
- QEPushButton, QERadioButton and QECheckBox widgets look in the ContainerProfile class to see if a slot they can use to create new GUI windows is available. Applications like QEGui publish a slot to open new GUIs using this mechanism. If the ‘handleGuiLaunchRequests’ property is true, the QEForm widget publishes its own slot for launching new GUIs and so all QE widgets within it will use the QEForm’s mechanism for launching new GUIs.

The following properties are specific to the QEForm widget:

- uiFile
File name of the user interface file to be presented. Refer to [Finding files \(page 27\)](#) for details on how this file is located.
- handleGuiLaunch
If set the QEForm will supply the slot used by any QE widgets it creates to launch new GUIs.
(Typically it is QE buttons that will use this slot.)
Generally this should be left unset when used within QEGui, allowing the QEGui application to supply the slot used to launch new GUI windows.

- `resizeContents`

If set, the QEForm will resize the top level widget of the .ui file it opens (and set other size and border related properties) to match itself. This is useful if the QEForm is used as a sub form within a main form (possibly another QEForm) and you want to control the size of the QEForm being used as a sub form.

If clear, the QEForm will resize itself (and set other size and border related properties) to match the top level widget of the .ui file it opens. This is useful if the QEForm is used as a sub form within a main form (possibly another QEForm) and you want the main form to resize to match the size of the QEForm being used as a sub form, or you want the sub form border decorations (such as frame shape and shadow) to be displayed.

In Figure 17, the QEGui application is displaying a user interface (.ui) file. QEGui uses QEForms to present .ui files. In the example given, the .ui file itself includes three QEForm widgets, each referencing the same sub form, but with different macro substitutions, resulting in a different title and the display of data from different variables. In this example the top level widget in the sub form is a QFrame with a border. To ensure the border is displayed, the QEForm widgets in the main form have their ‘`resizeContents`’ property set to false so the contents (the top level QFrame in the sub form) copies its border properties to the QEFrame, rather than the other way around.

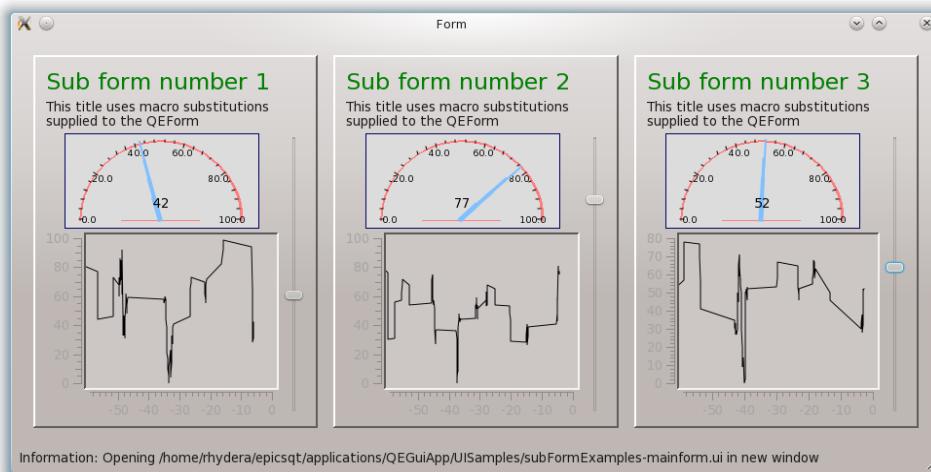


Figure 17 QEForm examples

A QEForm ‘`uiFile`’ property can include macro substitutions allowing a selection of file names based on macros supplied by a higher level form. For example, a GUI may open a QEForm to display motor details and supply the macro ‘`TYPE=pmac`’. A deeply nested sub form may be used to display motor details specific to the motor type and have a ‘`uiFile`’ property of ‘`$(TYPE)_specific.ui`’. A set of .ui files including `pmac_specific.ui` can be provided to allow type specific motor details to be displayed.

QEFormGrid

This widget allows a grid of forms to be loaded. To be documented

QEFrame and QEGroupBox

The QEFrame and QEGroupBox widgets provide a minimalist extension to the QFrame and QGroupBox widgets respectively. It provides user level enabled and user level visibility control to the frame or group box but more significantly to all the widgets enclosed within the QEFrame or QEGroupBox container also. The User Level example in Figure 7 (page 24) shows a QEGroupBox only visible in ‘Engineer’ mode.

QEImage

The QEImage widget is used to present an EPICS waveform (typically from areaDetector) as an image. It provides local analysis tools, such as displaying pixel profiles of slices through the image, and interacts with central analysis tools, such as areaDetector’s Region of Interest plugin.

Images can be zoomed, panned, and scrolled. Images views can be captured to a local file.

The image has functionality to support sample positioning.

An image input for the QEImage is defined using the following properties:

- `imageVariable` An EPICS waveform record (typically from areaDetector)
- `widthVariable` An EPICS record (typically from areaDetector)
- `heightVariable` An EPICS record (typically from areaDetector)
- `formatOption` The expected pixel format (unrelated to the data type of the waveform record, except that the pixel format must fit in the waveform record data type)

QEImage user interaction is as follows: (note the full context menu will not be available unless the ‘fullContextMenu’ property is set.)

- To pause image updating, press  if the Button Bar is displayed, or select ‘Pause’ from the context menu. To resume image updating, press  if the Button Bar is displayed, or select ‘Play’ from the context menu.
- To save the current image to a local file, press  if the Button Bar is displayed, or select ‘Save...’ from the context menu.
- To move the target position into the beam, mark the target and beam positions and press  on the Button Bar. To mark the target and beam, select ‘Mark Target’ and ‘Mark Beam’ from the select menu (available on the button bar and in the context menu) and mark the target and beam positions on the image with the mouse. When 

Page:42 of 108

perform the required move in each dimension. Each calc record subtracts the target position from the beam position, applies a scaling factor to convert pixels to distance, adds this to the current position of the target and writes the result to the target positioned.

Target and Beam markers can be seen selected in Figure 18.

The EPICS variables written to when marking the beam and target are defined by the following properties:

- targetXVariable
- targetYVariable
- beamXVariable
- beamYVariable
- targetTriggerVariable
- To zoom, either:
 - Select the required zoom percentage from the ‘Zoom’ menu on the button bar or in the context menu.
 - Select ‘Fit’ from the ‘Zoom’ menu on the button bar or in the context menu to zoom to a percentage that will fit the image in the current window. The image will be resized if the window size changes.
 - Choose ‘Select Area 1’ (Region 1) from the Mode menu on the button bar or from the context menu, select an area within the image, then select ‘Selected Area’ from the ‘Zoom’ menu on the button bar or in the context menu.

The image may be zoomed and set to an initial scroll position by default using the following properties:

- resizeOption
- zoom
- initialHosScrollPos
- initialVertScrollPos
- To rotate an image by 90 degrees clockwise or anticlockwise, or 180 degrees, select the appropriate option from the Flip/Rotate menu. Refer to Figure 21 for an example of rotated images.

The image may be rotated by default using the following property:

- rotation
- To flip image vertically or horizontally, select the appropriate options from the Flip/Rotate menu. Refer to Figure 21 for an example of flipped images.

The image may be flipped by default using the following properties:

- verticalFlip
- horizontalFlip
- To apply contrast reversal to an image (present a negative view), select the ‘Contract Reversal’ from the context menu. Refer to Figure 21 for an example of contrast reversal.

The image contrast may be reversed by default using the following property:

- contrastReversal

- To display a timestamp in the top left corner of the image, select ‘Show Time’ from the context menu.

The timestamp may be shown by default using the following property:

- showTime

- To present a profile of pixel values on a vertical ‘Horizontal Slice Profile’, ‘Vertical Slice Profile’, or ‘Line Profile’ from the Mode menu and mark a vertical slice, a horizontal slice, or mark an arbitrary line on the image with the mouse. After the markup is drawn, the mouse can be used to drag the markup to a new location or, in the case of the arbitrary line, can also be used to drag either end of the line to a new location. The mark-ups can be cleared by right clicking over the outline and selecting ‘Clear’

Figure 18 shows an image with Vertical, Horizontal and arbitrary profiles selected.

The profile thickness can be changed from a single line by grabbing the square handle in the middle of the line and moving the line boundary as required. The line boundary lines (dashed) can be grabbed anywhere and dragged to change the line thickness. When dragged back to the centre line the dashed boundary lines disappear, the thickness reverts to a single pixel, and the square handle used to set the thickness reappears in the centre of the line. The line thickness can also be returned to single line thickness from the line’s context menu.

The profile plots are simple indicative plots of the profile data. For more detailed analysis, the profile data presented in the plot can be copied by selecting ‘Copy Plot Data’ from the plot context menu. This can then be pasted into another program such as Excel. Note the data displayed and copied is generated from the most recent image update using the full original image data. It is unaffected by the current zoom level. The current zoom level will affect how accurately the lines can be positioned.

- To set the area in up to 4 areaDetector Region of Interest plugins, select ‘Select Area 1’, ‘Select Area 2’, ‘Select Area 3’ or ‘Select Area 4’ from the Mode menu on the button bar or in the context menu, and mark the area in the image using the mouse. When marked, the four EPICS areaDetector variables representing the Region of Interest area position and size will be updated. Figure 20 shows an example of this.

After the area mark-ups are drawn, the mouse can be used to drag the markups to a new location to drag individual sides or corners to a new location. The area can be cleared by right clicking over the outline and selecting ‘Clear’

The four EPICS areaDetector variables for each area are defined by the following properties:

- regionOfInterest1XVariable
- regionOfInterest1YVariable
- regionOfInterest1WVariable (width)
- regionOfInterest1HVariable (height)
- regionOfInterest2XVariable
- regionOfInterest2YVariable
- regionOfInterest2WVariable (width)
- regionOfInterest2HVariable (height)
- regionOfInterest3XVariable
- regionOfInterest3YVariable

- regionOfInterest3WVariable (width)
- regionOfInterest3HVariable (height)
- regionOfInterest4XVariable
- regionOfInterest4YVariable
- regionOfInterest4WVariable (width)
- regionOfInterest4HVariable (height)
- Image clipping can be achieved by defining clipping variables with the following properties:
 - clippingLowVariable
 - clipingHighVariable
 - clipingOnOffVariable
- To simplify the user interfaces, some options can be disabled by default using the following properties:
 - enableVertSliceSelection
 - enableHozSliceSelection
 - enableProfileSliceSelection
 - enableAreaSliceSelection (for all area and region selection)
 - enableTargetSliceSelection (for beam and target selection)
- Markup colors can be altered using the following properties:
 - vertSliceColor
 - hozSliceColor
 - profilecolor
 - areaColor
 - beamColor
 - targetColor
 - timeColor
- Displays of information about the image, such as line profile plots, are presented within the image by default. The following property can be set to request the application displaying the QEImage widget host these displays in docks. The QEGui application will honour these requests.
 - externalControls
- The context menu may be a full context menu containing options to manipulate the image or a simpler context menu containing just the standard context menu options. The following property determines which context menu is available.
 - fullContextMenu
- Local brightness and contrast controls can be enabled by setting the enableBrightnessContrast property, or by checking ‘Display local brightness and contrast controls’ from the widget’s context menu. Brightness and contrast can then be set by moving the brightness and contrast sliders as required. Local brightness and contrast are independent of areaDetector brightness and contrast settings. If ‘auto brightness and contrast’ is checked then selecting any area or region of interest will cause the brightness and contrast to be adjusted so match the range of pixel in the selected area. The reset button above the brightness and contrast sliders can be pressed to reset the controls to ‘normal’.

- If the image is not being displayed correctly, the QEImage context menu option ‘About Image...’ can be used to display a message box documenting how the QEImage widget is interpreting the image data.

The QEImage widget can act on requests from the window customisation mechanism. Refer to ‘Menu bar and tool button customisation’ (page 14) for details on the window customisation mechanism.

The following request names are valid:

- Save...
- Pause
- Move target position into beam
- About image...
- Zoom
- Flip/Rotate
- Mode
- Options...

Examples of elements in a window customisation menu definition file are:

- Button to save the image presented in a QEImage named BeamImage:

```
<Item Name="Save...">
    <BuiltIn Name="Save..." >
        <WidgetName>BeamImage</WidgetName>
    </BuiltIn>
</Item>
```

- Button to allow the QEImage named BeamImage to be paused:

```
<Item Name="Pause">
    <BuiltIn Name="Pause" >
        <WidgetName>BeamImage</WidgetName>
    </BuiltIn>
</Item>
```

- Menu to zoom the QEImage:

```
<Item Name="Zoom" >
    <BuiltIn Name="Zoom" >
        <WidgetName>BeamImage</WidgetName>
    </BuiltIn>
</Item>
```

- Menu to flip and rotate the QEImage:

```
<Item Name="Flip/Rotate" >
    <BuiltIn Name="Flip/Rotate" >
```

```
<WidgetName>BeamImage</WidgetName>
</BuiltIn>
</Item>
```

- Menu to chose the current user interaction with the image: (for example, when clicking and dragging over the image, is the user panning the image, or selecting a region of interest, etc)

```
<Item Name="Mode" >
    <BuiltIn Name="Mode" >
        <WidgetName>BeamImage</WidgetName>
    </BuiltIn>
</Item>
```

- Button to present the Optinos dialog for the QEImage named BeamImage:

```
<Item Name="Options..." >
    <BuiltIn Name="Options..." >
        <WidgetName>BeamImage</WidgetName>
    </BuiltIn>
</Item>
```

QE Framework – User Interface Design

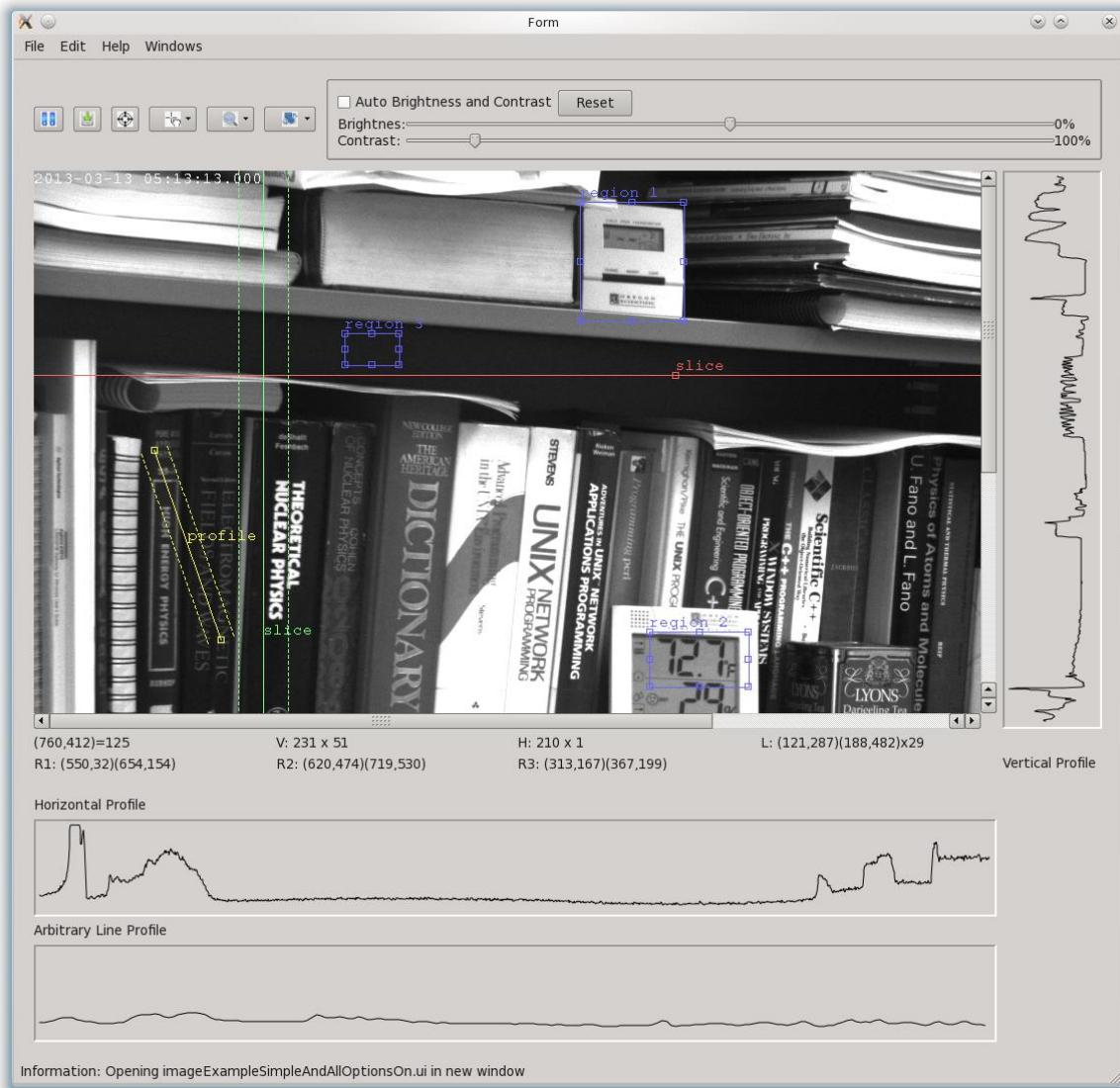


Figure 18 QEImage with most options activated



Figure 19 Minimal use of QEImage

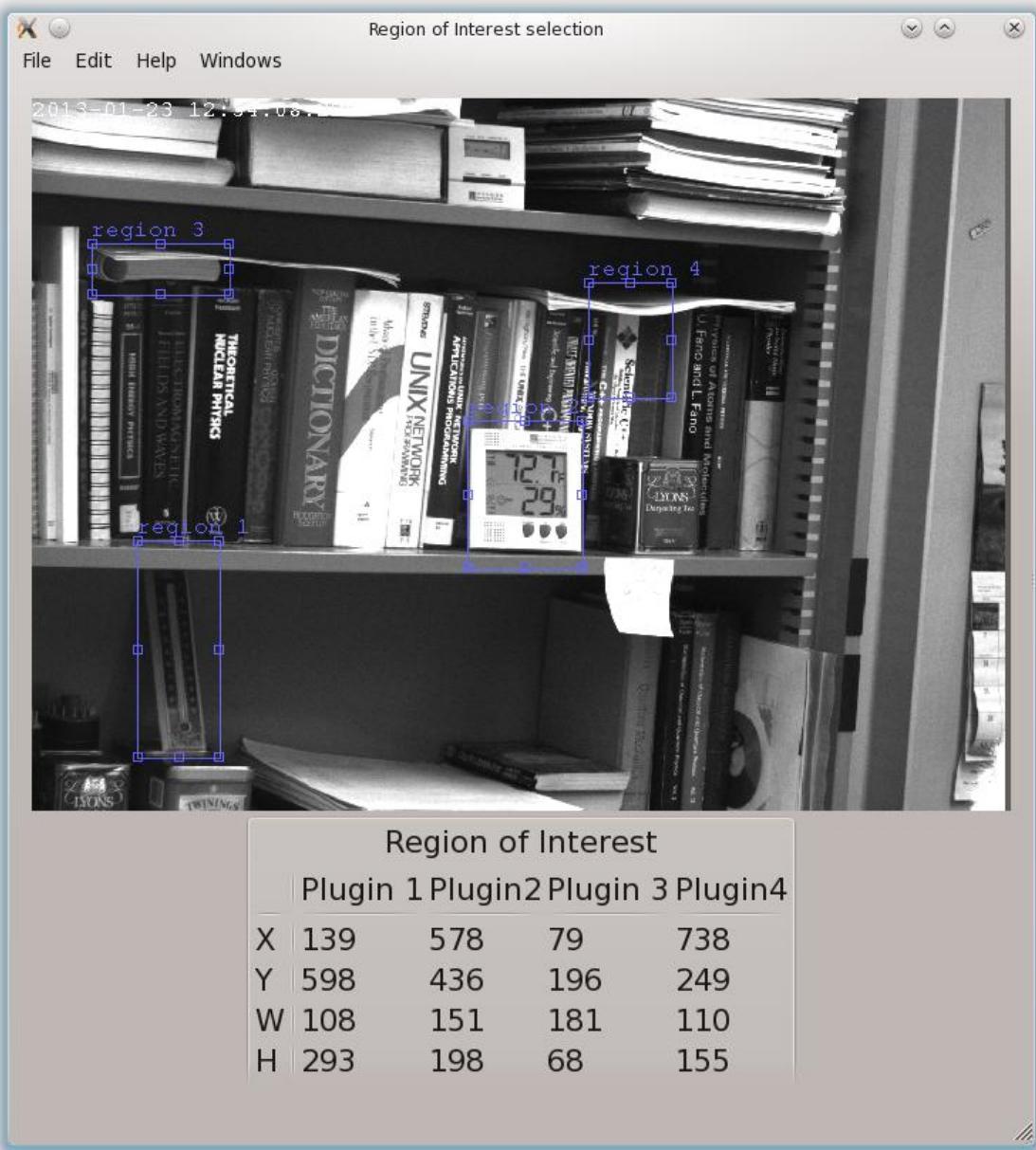


Figure 20 QEImage specifying areaDetector Region of Interest

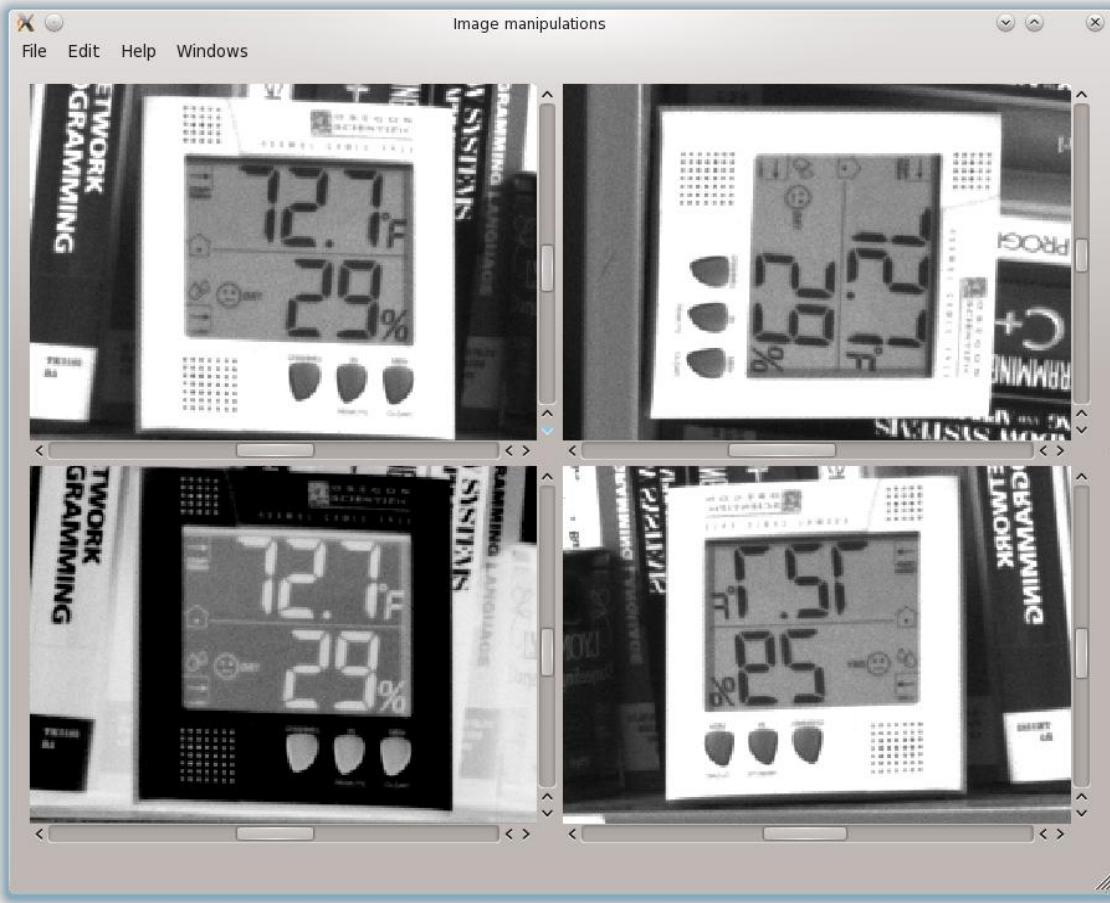


Figure 21 Some QEImage image manipulation options

QELabel

The QELabel widget provides a simple textual display of EPICS data. It is based on the QLabel widget and so shares QLabel properties such as justification.

The QELabel widget provides many options for formatting the EPICS data as text. These formatting options are common to all QE widgets that display EPICS data as text. Most of these options do not presume any specific EPICS data type. Refer to ‘String formatting properties’ (page 34) for details about the standard text formatting. In particular, note how local enumerations can include style hints for QELabel widgets.

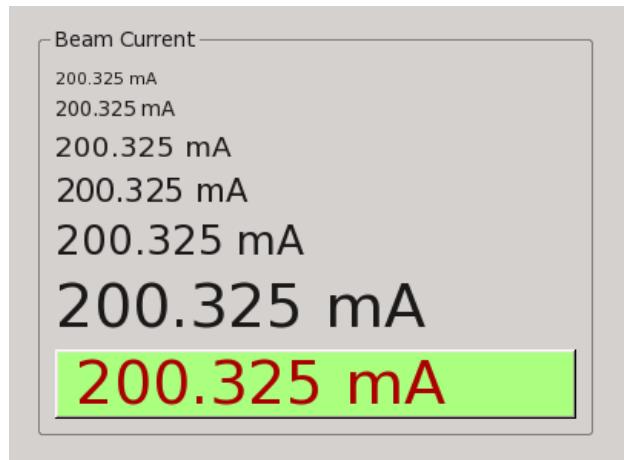


Figure 22 QELabel examples with variations to QLabel properties



Figure 23 QELabel used to display a pump failure

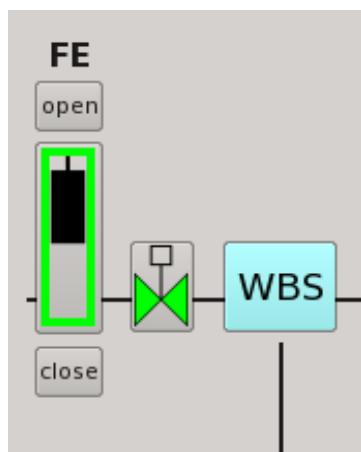


Figure 24 QELabels used icons to represent states

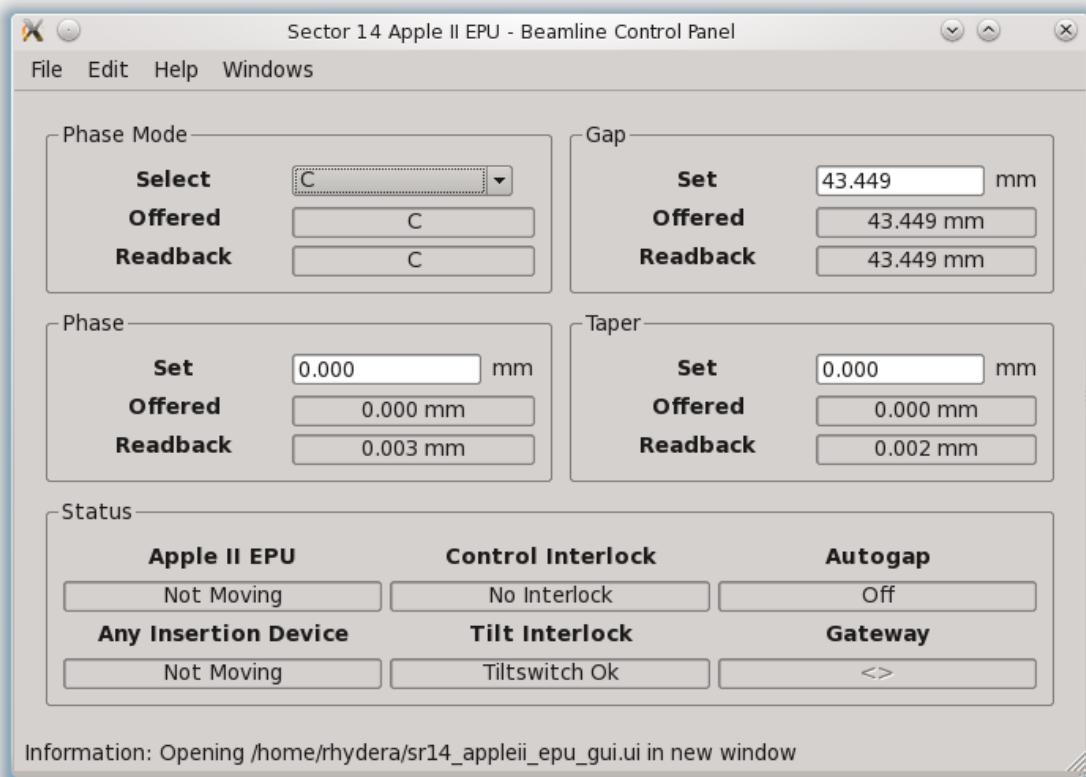


Figure 25 GUI using mostly QELabels to represent numeric and textual data

The text displayed in a QELabel reflects the value of the variable. How that text is presented reflects the state of the variable as follows:

- **Invalid** (not connected) – The QELabel is displayed not-enabled, or ‘greyed out’.
- **In alarm condition** – The QELabel is optionally displayed with an appropriate background colour.

In common with any Qt widget, many aspects of the presentation can be set by the GUI designer, or modified by an imposed ‘style’. It is important that any changes to the presentation of the QELabel is compatible with the display of the variable state.

Display of alarm state is optional – Display of alarm state is on by default. It may be appropriate to turn display of alarm state off if the alarm state is displayed elsewhere, or the alarm state is the actual field being displayed.

QLineEdit

The QLineEdit widget provides the ability to textually modify the value of a single PV. This widget is (indirectly) derived from QLineEdit. The example in Figure 26 shows a QLineEdit widget connected to an ao record. While this widget is primarily intended for writing to string PVs, it can also be used with

numerical PVs as in this example. However in this case, a QENumericEdit or a QESpinBox widget may be more appropriate.

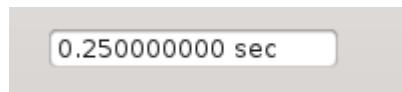


Figure 26 QLineEdit example

The behaviour of the widget is defined by the widget specific properties as shown in Figure 27.

Property	Value
> QObject	
> QWidget	
> QLineEdit	
QEGenericEdit	
> variable	TS02EVR01:TTL01_TIME_DELAY_SP
> variableSubstitutions	
subscribe	<input checked="" type="checkbox"/>
writeOnLoseFocus	<input type="checkbox"/>
writeOnEnter	<input checked="" type="checkbox"/>
writeOnFinish	<input checked="" type="checkbox"/>
confirmWrite	<input type="checkbox"/>
variableAsToolTip	<input checked="" type="checkbox"/>
enabled	<input checked="" type="checkbox"/>
allowDrop	<input type="checkbox"/>
visible	<input checked="" type="checkbox"/>
messageSourceId	0
> userLevelUserStyle	
> userLevelScientistStyle	
> userLevelEngineerStyle	
userLevelVisibility	User
userLevelEnabled	User
displayAlarmState	<input checked="" type="checkbox"/>
QELineEdit	
precision	4
useDbPrecision	<input checked="" type="checkbox"/>
leadingZero	<input checked="" type="checkbox"/>
trailingZeros	<input checked="" type="checkbox"/>
addUnits	<input checked="" type="checkbox"/>
> localEnumeration	
format	Default
radix	10
notation	Fixed
arrayAction	Ascii
arrayIndex	0

Figure 27 QLineEdit properties

As well as the usual PV variable name, substitutions, display format, user level etc. properties, the widget has additional properties to control it mode of operation:

- a) subscribe (default true): determines if the widget subscribes for data updates and displays current data;
- b) writeOnLoseFocus (default false): when true this widget automatically writes any changes when it loses focus;
- c) writeOnEnter (default true): when true writes when the user presses 'enter'. Note, the current value will be written even if the user has not changed it;
- d) writeOnFinish (default true): when true writes any changes when the user finished editing (the QLineEdit 'editingFinished' signal is emitted). No writing occurs if no changes were made; and
- e) confirmWrite (default false): when true this widget will ask for confirmation (using a dialog box) prior to writing data.

QELink

The QELink widget is part of a general mechanism to allow a GUI to be modified by data changes. For example, to disable a GroupBox if a variable is equal to a nominated value.

QELink widgets are only visible while in Designer. After placing them in a GUI the appropriate signals/slots connections and properties are defined to configure the GUI behaviour based on PV values. Then when opened in QEGui (or in any application except Designer) the functionality remains, but the QELink widget itself is hidden.

Typically, a QE widget sends data update signals to a QELink widget which makes a comparison and signals a value to another widget depending on the comparison result. The output signal can be used to set a widget invisible, or enabled, or click a button, or set focus, or raise, or...

In Figure 28, A QELink widget (circled) is configured to receive data update signals from a QELabel displaying beam current. It compares this to 205 (mA) and if greater sends a signal to enable the group box on the right. The signals used and the relevant QELink Properties are shown in the figure. Figure 29 shows this GUI in use by the QEGui display application. The QELink widget is not visible. The 'Shutdown' group box on the right is not enabled as the beam current is less than 205 mA.

The QELink widget can be make visible at all times by setting the 'visible' property.

Traditionally, the type of GUI functionality QELink widgets support has been effected by using EPICS database variables (often CALC records) to determine the state of GUI items. Where the variable is primarily a part of the control system this is appropriate. Where the variable is only present to support the GUI, then this functionality should be embedded in the GUI.

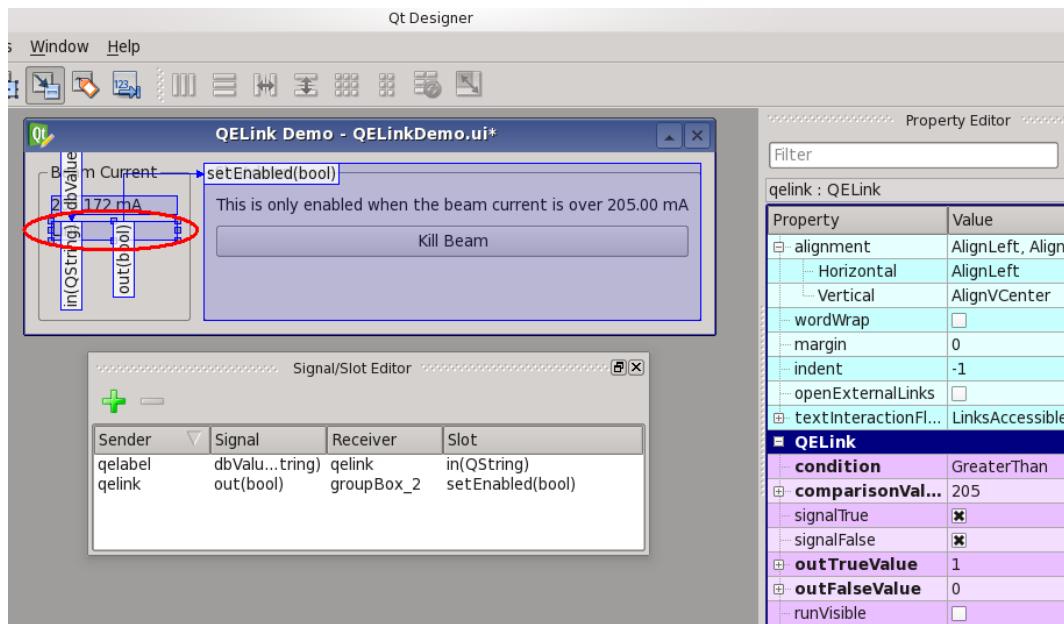


Figure 28 QELink being configured



Figure 29 QELink in use

QELog

The QELog widget provides a destination for messages generated by other QE widgets, or other widgets and applications using the QE framework. Messages may be generated due to user actions such as changing user level, data issues such as an invalid variable name, and application errors.

The QELog widget receives and displays messages from the QE framework message system. Any application or widget can generate or consume these messages. For example, the QEGui application displays QE messages in its status bar.

Refer to ‘Logging’ (page 25) for a more general discussion on how the QELog widget is used as part of the QE framework message logging system.

Time	Type	Message
2013/01/04 - 17:33:52	INFO	The user type was changed from 'User' to 'Scientist'
2013/01/04 - 17:33:54	INFO	The user type was changed from 'Scientist' to 'Engineer'
2013/01/04 - 17:33:56	INFO	The user type was changed from 'Engineer' to 'User'

Info messages
 Warning messages
 Error messages

Clear Save

Figure 30 QELog example

The QELog widget is designed to be dropped on a form and automatically catch messages from QE widgets on the same form, or in sub forms. Alternately, it can be used to filter messages from specific sets of QE widgets and forms.

The logged messages can be saved or cleared by the user. The user can also select the type of messages logged from a message filter. Note, the message filter viewed by the user is used by the user to filter message content. For example, the user can select only information messages. Filter properties are also available to filter messages based on the source of the message, rather than content.

Properties of the QELog widget allow:

- Selective display of message time, type and content.
- Presentation of the 'Clear' and 'Save' buttons and the message filter.
- Message type colour selection.
- Selection of the message filtering based on the source of the message. Note, this is different to the message filter presented to the user which allows the user to filter based on message type.

Each QE widget can be given a message source ID (the messageSourceId property). The GUI designer is free to allocate any ID to any widget. IDs do not need to be unique, so a set of widgets might have the same message source ID if required.

Each QEForm widget also has a unique message form ID allocated by the QE framework.

QELog widgets can be set up to filter messages based on the message source ID (the QE widget or set of widgets it came from) and the QEForm that widget generating the message is in. The filtering is as follows:

- **Form filtering:**
 - **None** - Never match based on the form ID
 - **Match** – Use the message if message came from a widget in the same form as the QELog widget, or from a sub form. Note, Messages are accepted from sub forms because QEForms themselves filter messages and rebroadcast them as their own.
 - **Any** – Always use the message. When this option is selected, message source filtering, below, is irrelevant.
- **Message source filtering:**

- **None** – Never match based on message source ID
- **Match** – Use the message if the message came from a widget with the same message source ID.
- **Any** - Always use the message. When this option is selected, form filtering, above, is irrelevant.

By default a QELog widget form filter is set to ‘Match’ and the message source filter is set to ‘None’. These are the settings required to allow a QELog widget to be dropped onto a form to display all messages from widgets on the form, including those within sub forms.

QELogin

The QELogin widget allows a user to select one of three user levels: ‘user’, ‘scientist’, and ‘engineer’. Most QE widgets can be set to use the current user level to control if the widget is enabled, visible, or if a particular style string is applied. Refer to ‘User levels’ (page 23) for details on how user levels can control access to GUI components. The QELogin widget can be dropped into any QUI form, but provides some features that allow it to be effectively used as the basis for a user level dialog box.



Figure 31 QELogin widget being used to set the user level within the QEGui application

The QEGui application uses a QELogin widget in the ‘File -> User Level’ menu option as shown in Figure 31. Generally, therefore, GUIs presented in QEGui do not need to include a QELogin widget, except perhaps in ‘status only’ mode to indicate the current user level. If not using QEGui, QELogin widgets can be dropped into a GUI form or used programmatically to manage user level.

The QELogin widget emits a ‘login’ signal when a user successfully changes the user level. If the QElogin widget is being used within a dialog box, this signal can be connected to the dialog box ‘accept’ slot to close the dialog box.

If defined the QELogin will use an application wide set of user level passwords which can be set up using the QE framework. The QEGui application uses the QE framework to set passwords. The QEGui application allows these passwords to be set when the ‘Edit’ menu is enabled. If no global passwords

have been set using the QE framework the QELogin widget will use its own ‘user’, ‘scientist’, and ‘engineer’ level password properties. Using the QELogin widget password properties makes sense when the application does not set global passwords through the QE framework, and when there is only one QELogin widget in use. The QEGui application uses a QELogin widget in the ‘File -> User Level’ menu option.

The QELogin widget can be used in a ‘status only’ mode which simply displays the current user level.

When not in ‘status only’ mode the QELogin provides controls for a user to change the user level. The QEWidget widget operates in ‘compact mode’ by default where the ‘Login’ button must be pressed to open a dialog box presenting all the user level selection fields. When not in ‘compact mode’ the QELogin widget presents all the user level selection fields.

Figure 32 shows several versions of the same GUI containing a QELogin widget. The QELogin widget in the first is in ‘status only’ mode, the other two have controls for the user to change the user level with the second in ‘compact mode’ (the default). (Note, the user level is also different in each example causing other elements of the GUI to be displayed or enabled.)

The QELogin widget has a ‘Logout’ button that will return the user level to what it was before logging in to a new level. Note, a previous level history is kept for each QELogin widget, so if multiple QELogin are presented to the user each will have its own history.

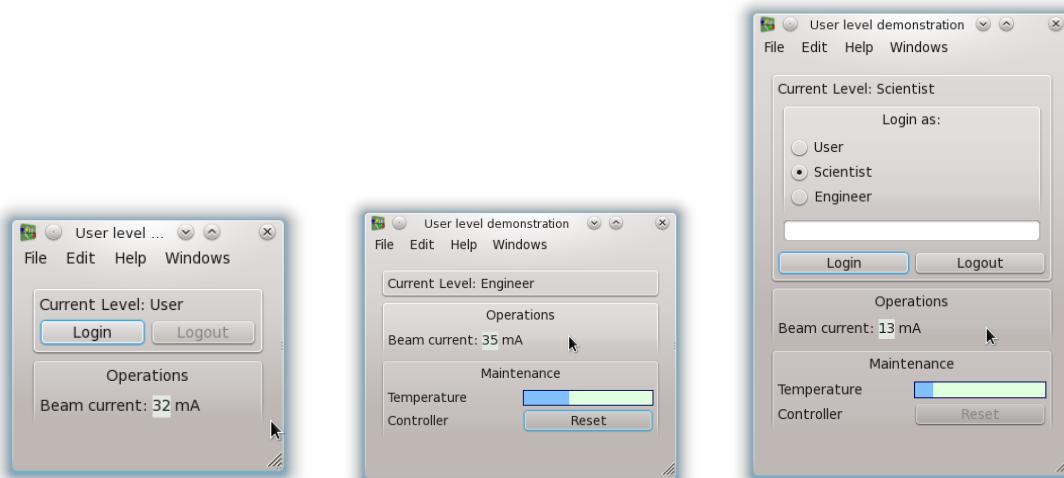


Figure 32 QELogin widgets in various modes and user levels

The QELogin widget is based on a QFrame. In addition the QELogin widget has the following properties:

- **statusOnly**
If set, the current user level only is presented. No controls will be shown to the user.
- **compactStyle**
If set, and not in ‘status only’ mode, the controls will consist of only a ‘Login’ button and a

‘Logout’ button. Pressing the ‘Login’ button will display a dialog box with all the controls required for changing the user level.

- **userPassword, scientistPassword, engineeringPassword**

These passwords, if present, must be entered to change to the appropriate user level. These passwords are ignored if the QE framework has been used by the application to set up application wide passwords. The QEGui application is an example where application wide passwords can be set.

QENumericEdit

The QENumericEdit widget provides the ability to modify the value of a single numeric, either integer or floating point, PV. This widget is (indirectly) derived from QLineEdit. Figure 33 shows examples of the widget in several configurations, and in each case the widget is connected to the same PV.



Figure 33 QENumericEdit examples

The first example shows a QENumericEdit in its default configuration, and in appearance at least, looks very much like its QLineEdit counterpart. The second example shows the appearance with the separator property set to “comma”. The 3rd, 4th and 5th show the same with the radix property set to Hexadecimal, Octal and Binary respectively. The widgets tool tip will be annotated accordingly.

Unlike the QLineEdit, the user may only enter valid radix digits and if a sign is present enter a plus/minus (“+”, “-”). A sign is displayed if and only if the allowed range of values encompasses negative values. The user may also use the left and right key to navigate sideways to select a digit and use the up and down keys to increment or decrement the overall value by an amount corresponding to the unit value of the selected digit. An example sequence is shown below (using an approximate representation of the widget appearance):

0.2589 Amps	
0.2589 Amps	- widget gets focus – the current selected digit is after the first decimal point
0.2589 Amps	- left key – first digit selected – note: the decimal point skipped
0.2589 Amps	- right key three times – third digit after point selected
0.2599 Amps	- up key – increment value by 0.001
0.2609 Amps	- up key – increment value by 0.001, second digit has changed from 5 to 6.

Like the QLineEdit widget, the subscribe, writeOnLoseFocus, writeOnEnter, writeOnFinish and confirmWrite properties modify the behaviour is exactly the same manor. The widget specific properties are shown in Figure 34. These are described below:

- a) autoScale (default true): when true the number of leading zeros, precision, minimum and maximum values will be determined from the PV's associated meta. When false (or when not connected), the precision, leadingZeros, minimum and maximum properties are used;
- b) precision (default 4): specifies the number digits after the decimal point for display and editing;
- c) leadingZeros (default 3): specified the number of digits before the decimal point;
- d) minimum: specifies the minimum value allowed to be entered;
- e) maximum: specifies the maximum value allowed to be entered;
- f) addUnits (default true): the widget displays includes any engineering units;
- g) radix (default Decimal): allows the selection of display/editing radix. Unlike other widgets, this is restricted to just four options: Decimal, Hexadecimal, Octal and Binary.
Note: the widget assumes that the precision/leading zeros , manual or automatic, are appropriate for the selected radix; and
- h) separator (default None): allows the use of a character to break up the string. This may be one of None, Comma, Underscore or space. For Decimal and Octal, this is between every third digits, whereas for Hexadecimal and Binary, this is every 4th digit.

Note: the widget ensures that items (b), (c), (d) and (e) are consistent. For example: the maximum value is always greater than or equal to the minimum value. When in decimal mode, he sum of (b) and (c) is never greater than 15 which is approximately the maximum significance of an IEEE 64 bit float which is used to hold the underlying widget value (and indeed is the “best” significance supported by Channel Access).

Property	Value
> QObject	
> QWidget	
> QLineEdit	
↳ QEGenericEdit	
> variable	TS02EVR01:TTL01_TIME_DELAY_SP
> variableSubstitutions	
subscribe	<input checked="" type="checkbox"/>
writeOnLoseFocus	<input type="checkbox"/>
writeOnEnter	<input checked="" type="checkbox"/>
writeOnFinish	<input checked="" type="checkbox"/>
confirmWrite	<input type="checkbox"/>
variableAsToolTip	<input checked="" type="checkbox"/>
enabled	<input checked="" type="checkbox"/>
allowDrop	<input type="checkbox"/>
visible	<input checked="" type="checkbox"/>
messageSourceId	0
> userLevelUserStyle	
> userLevelScientistStyle	
> userLevelEngineerStyle	
userLevelVisibility	User
userLevelEnabled	User
displayAlarmState	<input checked="" type="checkbox"/>
↳ QENumericEdit	
autoScale	<input checked="" type="checkbox"/>
precision	4
leadingZeros	3
minimum	-999.999900
maximum	999.999900
addUnits	<input checked="" type="checkbox"/>
radix	Decimal
separator	None

Figure 34 QENumericEdit properties

QEPeriodic

The QEPeriodic widget is used to associate variable values with elements and allow a user to read or write values by element selection.

For example, a two axis reference foil stage may be controlled with a QEPeriodic widget. Each element on the reference foil stage can be placed in the beam by setting the position on the two motors controlling the stage. Using the QEPeriodic widget the user can get a direct reading of which element is in the beam, or move an element into the beam by selecting it from a dialog containing a periodic table.

Alternatively, using a QEPeriodic widget a variable holding ionization energy may be set directly by a user selecting an element from a dialog containing a periodic table.

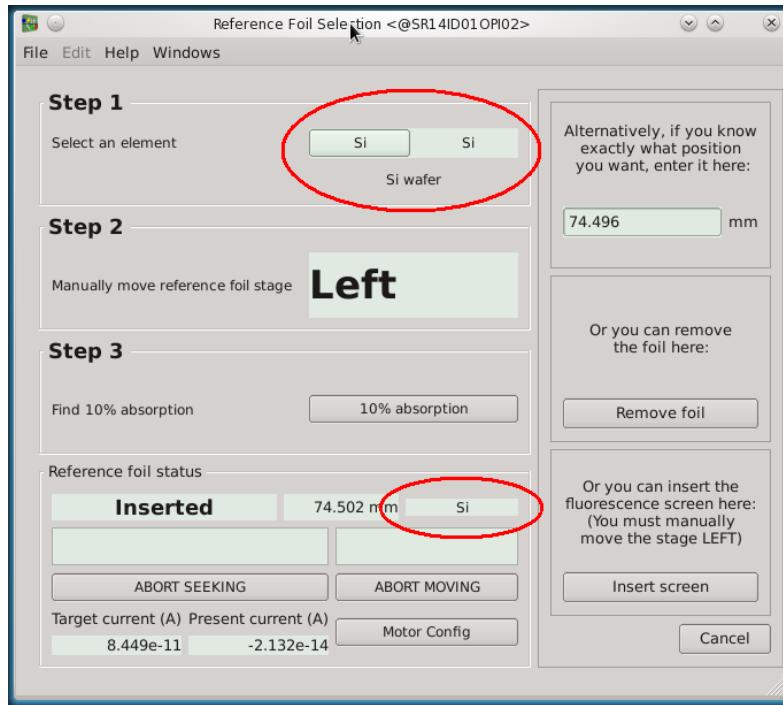


Figure 35 QEPeriodic used for both read-back and control by element.

A property determines if the user is presented with a read-back label, a write button, or both.

- PresentationOptions (Default is buttonAndLabel)

When the read-back label is enabled the widget reads the required variables and presents a label displaying the element associated with the values read. An example of this is shown in Figure 36. The two properties defining the one or two variables use to update the label are:

- readbackLabelVariable1
- readbackLabelVariable2

When the write button is enabled, the widget presents a button displaying the currently selected element. When pressed, a dialog containing a periodic table is displayed allowing the user to select an element. When the user selects an element from the table, the widget writes the associated values. An example of this is shown in Figure 37. The two properties defining the one or two variables written to are:

- writeButtonVariable1
- writeButtonVariable2

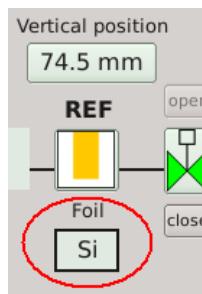


Figure 36 QEPeriodic widget used to represent variables by element in a read only mode.

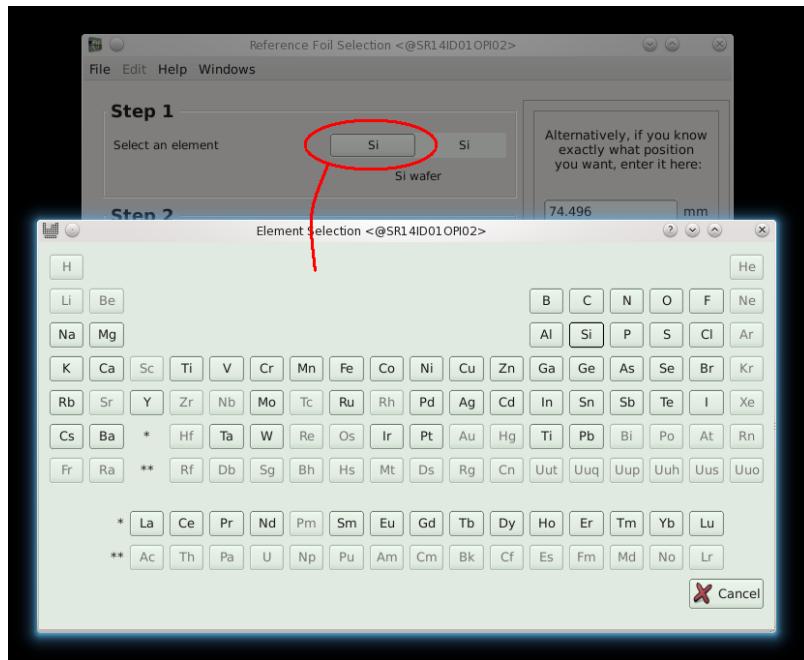


Figure 37 QEPeriodic widget used to manipulate variables by element selection

The QEPeriodic widget associates an element to one or two variable values by comparing the variable values to arbitrary values set up for each element at design time, or to intrinsic attributes of the element such as ionization energy or melting point. The associations available are:

- Number
- Atomic weight
- Melting point
- Boiling point
- Density
- Group
- Ionization energy
- User value1 (defined in the userInfo property)
- User value2 (defined in the userInfo property)

The following properties determine how the widget associates each variable to the each element:

- variableType1 (default is userValue1)
- variableType2 (default is userValue2)

The widget is typically configured at design time to associate one or two arbitrary values with each element of interest. Alternatively, the widget can be configured to associate one or two values with intrinsic attributes of the element. With these associations in place a user can view or write to variables by element reference.

When configured to match an element by comparing the variable values to arbitrary values for each element (which is the default), these arbitrary values must be defined at design time. A single property holds an XML string defining these values:

- userInfo

The form of this XML is shown in the following example:

```
<elements>
    <element number="5" enable="yes" value1="58.498" value2="2" text="BN powder"/>
    <element number="6" enable="yes" value1="45.676" value2="2" text="HOPG"/>
    <element number="7" enable="yes" value1="58.498" value2="2" text="BN powder"/>
    ...
    ...
    ...
</elements>
```

While the userInfo property may be edited directly, it is easier to use the user info editor shown in Figure 38. This editor may be invoked by right clicking on the QEPeriodic widget in ‘Designer’ and selecting ‘Edit User Info...’. Figure 39 details shows what can be configured for each element as well as the variable values to match. Note, if the element is not enabled, the user will not be able to select this element (it will be greyed out in the selection dialog) and it will never match and be displayed in the read-back label)

As well as defining the values associated with the element, some text may also be defined which will be emitted by the dbElementChanged signal when the read-back label updates. This may be, for example, connected to a standard QLabel setText slot as shown in Figure 35.

QE Framework – User Interface Design

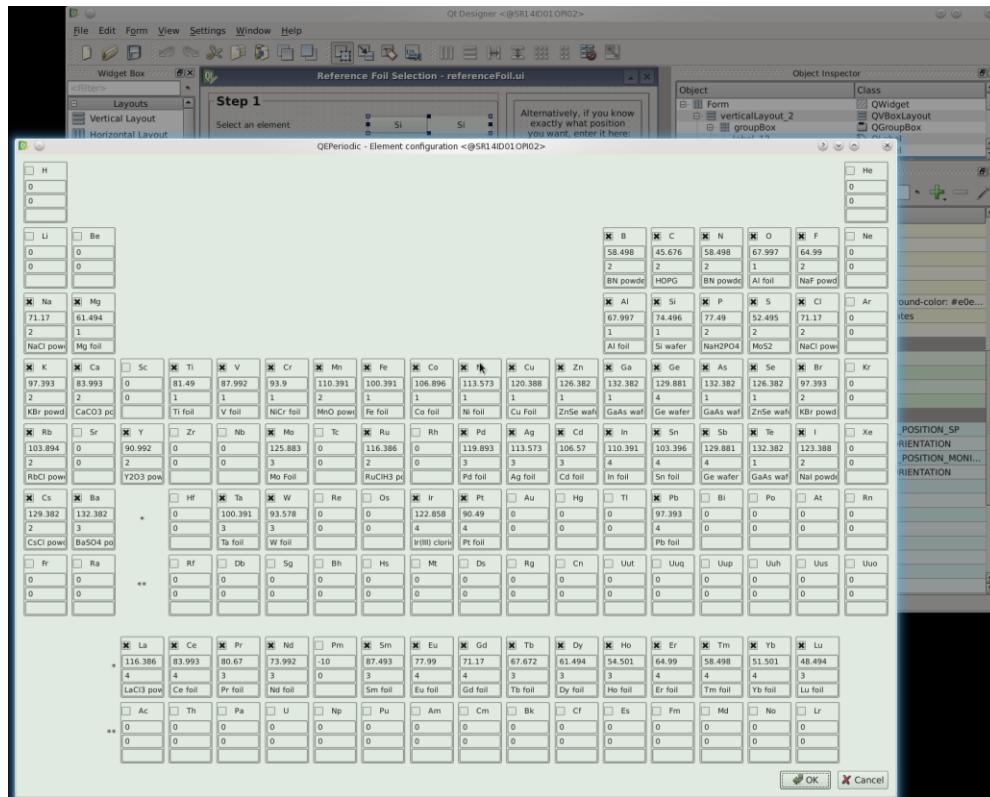


Figure 38 Editing the QEPeriodic userInfo property - the relationship between each element and variable values

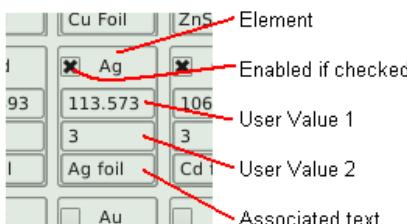


Figure 39 Editing the QEPeriodic userInfo property - what is associated with each element

A tolerance can be specified for each of the associated variables so each element will match a small range of values. The tolerance is set to be marginally larger than the positional error of the system being measured. The tolerance is defined for each variable by the properties:

- variableTolerance1
- variableTolerance2

QEPlot

The QEPlot widget is a basic widget for plotting scalar variables over time, or presenting waveform variables. On receiving an update of a scalar value it will add the value to the scalar values already presented in the plot. On receiving an update of a waveform it will replace the current plot with a plot of the new waveform. This widget is intended for presentation of a small indicator plot. It has limited

scaling ability and no user interaction such as cursors and measurements, or user defined scaling or timescale.

Up to four variables may be plotted. By default plots are auto-scaled with a time span of 60 seconds.



Figure 40 GUI using QEPlot widgets to plot waveforms (left) and scalar values (right)

Individual trace presentation:

Each trace may be given a colour, line style, and a legend using the following properties:

- traceColour1-4
- traceStyle1-4
- traceLegend1-4

Scaling and units:

All four traces must be scaled as a set. The QEPlot widget can auto-scale all traces, or use a fixed range. With the following properties:

- autoScale (default is to auto-scale)
- yMin and yMax

X and Y units may be specified using the following properties: Note, these are presented regardless of the actual data.

- xUnit, yUnit

Plot presentation:

X and Y axis are independently optional, as are major and minor X and Y grids, using the following properties:

- axisEnableX and axisEnableY (default is display axis)
- gridEnableMajorX and gridEnableMajorY (default is no grids)
- gridEnableMinorX and gridEnableMinorY (default is no grids)

The plot background and grid colours may be changed using:

- backgroundColor
- gridMajorColor and gridMinorColor

The entire plot may be given a title:

- title (default is no title)

Scalar attributes:

When displaying scalar values, the QEPlot widget displays all updates with timestamps within the time span specified. The entire plot is redrawn asynchronously to updates. The values on the X axis are seconds before the current time. The relevant properties are:

- timeSpan (default is 59 seconds)
- tickRate (default is 50mS)

Waveform attributes:

When displaying waveforms, the QEPlot widget presents the waveform and sets the range of values on the X axis according to properties specifying an initial value and an incremental value per point in the waveform.

- xStart
- xIncrement

QEPlotter

(Note: this widget is still under development but nonetheless functional)

The QEPlotter widget is a widget for presenting waveform variables. On receiving an update of a waveform it will replace the current plot with a plot of the new waveform. This widget is intended for presentation on a number of waveforms, such as from the sscan record. This widget is a complex widget and used as the basis of one of the QEGui's built-in forms.

Up to 16 'Y' variables may be plotted against an optional 'X' waveform variable.

The 'X' variable and the 'Y' variables are specified by a data object and a size object.

A data object is typically specified by a Process Variable (PV), but can also be an expression similar in form to that used by the calc/calcout records (in fact under the covers, QEPlotter uses the same postfix functions as the calc record).

As expected, PVs are specified as a PV name, e.g. "BR01RF01AMP01:OUT_FWD_PHASE_MONITOR".

Expressions are introduced by an equals character, e.g. "=-LN (B/C)". No sensible PV name begins with "="". See the expressions section below for details on expressions.

A size object may be defined by a PV name, e.g. "SR14ID01:scan1.CPT"; as a constant such as "72"; or left blank. Since all 'Y' variables are plotted against the 'X' variable, the 'Y' size is effectively truncated to match the 'X' size if needs be.

The following tables show the size and values used for the 'X' variable for each combination of size object/data object.

Size Object	Data Object		
	Blank	Data PV name	Calculation
Blank	n/a	No. Data PV elements	n/a
Size PV name	Value of PV	Min (Value of Size PV, No. Data PV elements)	Value of PV
Constant	Fixed Value	Min (Value of Size PV, Fixed Value)	Fixed Value

Size Object	Data Object		
	Blank	Data PV name	Calculation
Blank	n/a	X [s] := PV [s]	n/a
Size PV name	X [s] := s	X [s] := PV [s]	X[s] := calc (s)
Constant	X [s] := s	X [s] := PV [s]	X[s] := calc (s)

Note: the widget attempts to make sensible assumption if/when the size or data object is blank. For example if no data PV is specified and a constant size, say 40, is specified then the 'X' values run from 0 to 39.

The following tables show the size and values used for the 'Y' variable for each combination of size object/data object. This is similar to the above, although there are some differences.

Size Object	Data Object		
	Blank	Data PV name	Calculation
Blank	n/a	No. PV elements	Number of X elements
Size PV name	n/a	Min (Value of PV, No. PV elements)	Value of PV
Constant	n/a	Min (Value of PV, Fixed Value)	Fixed Value

Size Object	Data Object		
	Blank	Data PV name	Calculation
Blank	n/a	Y [s] := PV [s]	Y[s] := calc (s, X[s], A[s], B[s],...)
Size PV name	n/a	Y [s] := PV [s]	Y[s] := calc (s, X[s], A[s], B[s],...)
Constant	n/a	Y [s] := PV [s]	Y[s] := calc (s, X[s], A[s], B[s],...)

Expressions

Each point of the expression waveform is calculated from the corresponding point of each of the input waveforms. On the QEWidget, the 16 'Y' variables are labelled A to P, so in this expression, the B arguments represents the value provided by the 2nd Y variable, and C the value provided by the 3rd Y variable. X refers to the 'X' variable and S refers to the array element number starting from 0.

The QEPlotter also calculates dA/dX, dB/dX, dC/dX etc. and these are available within expressions as A', B', C' etc. For completeness X' and S' are also available.

Readers familiar with the calc/calcout records will recall that these only support 12 inputs (A to L). The QEPlotter widget performs a translation of the 36 possible inputs onto 12 inputs. It can do provides that no expression uses more than 12 arguments, i.e. = C' + S + X is a valid QEPlotter expression, whereas =A + B + C + D + E + F + G + H + I + J + K + L + M is invalid as there are more than 12 elements.

Scaling and Presentation Control

Currently the QEPlotter is dynamically scaled. Future enhancements will included fixed scaling, normalised scaling, black background. These will be documents as these features are added.

QEPushButton, QERadioButton and QECheckBox

General description:

The QEPushButton, QERadioButton and QECheckBox widgets provide the following non exclusive functions:

- Write to a variable
- Read from a variable
- Issue a command to the operating system

- Open a new GUI form.
- Emit a signal

If the properties used to define any or all of these functions are set up, the functions will be carried out.

All QE button like widget types are based on QEGenericButton and on QAbstractButton (through QPushButton, QRadioButton and QCheckBox). QEPushButton, QERadioButton and QECheckBox widgets share most properties and it is mainly the way the buttons are presented that differentiates them.

Generally, QERadioButton and QECheckBox widgets will be shown as checkable, and properties related to the checked state are more likely to be used for QERadioButton and QECheckBox widgets.

Various data values can be written on any or all or the following button actions:

- Press A mouse press with the pointer over the button
- Release A mouse release with the pointer over the button
- Click A press and release while over the button

By default, values are written on a button click. A click will be accompanied with a press and release.

Writing values on Press and Release typically allows a value to be set momentary, while the button is held down. In this case, no data would be written on the click.

Use of enumerated values:

QE buttons can display variable data in the button label and, like many QE widgets, QE buttons may construct a list of enumerated values for this purpose. The same list of enumerations, if present, is used to validate values written by the button.

If a list of enumerated values has been constructed for the variable being written to, then any value written must match a value from the enumeration list. The enumeration list may have originated from the database or be stored locally in the GUI file. The ‘pressText’, ‘releaseText’ and ‘clickText’ properties must all match one of the enumeration values or an error will be displayed when a write is attempted. If an enumeration list was build from the database then the following error will be displayed:

Write failed. String not written was '*your string*'. Value does not match an enumeration value from the database.

If an enumeration list was stored in the GUI file then the following error will be displayed:

Write failed. String not written was '*your string*'. Value does not match a local enumeration value.

Enumeration lists will be present and used to check any string written in the following scenarios:

- The ‘format’ property is set to ‘LocalEnumeration’ and ‘localEnumeration’ property is defined.
- The ‘subscribe’ property is set to true (checked), the ‘format’ property is set to ‘Default’ and enumeration values were successfully read from the database for the variable.

Conversely, enumeration lists will not present and string will be written without validation by the button in the following scenarios:

- The ‘format’ property is set to ‘LocalEnumeration’ but no ‘localEnumeration’ property is defined.
- The ‘subscribe’ property is set to false (unchecked), the ‘format’ property is not set to ‘Default’ or enumeration values were not successfully read from the database for the variable.

In these scenarios any string in the pressText’, ‘releaseText’ and ‘clickText’ properties is written as is and it is up to the database to accept or reject the string.

Signals on user action:

The same value that would be written to a variable is also interpreted as an integer and emitted as a ‘pressed’, ‘released’ or ‘clicked’ signal. This is useful, for example, for selecting a tab in a tab widget or a page in a toolbox widget.

Why QE buttons can open a new GUI form:

While QEPushButton, QERadioButton and QECheckBox widgets can open a new GUI form when set up correctly without any action on the part of the application that created them, this functionality is mainly so the button functionality can be tested from the Designer ‘preview’ window. Applications using QEPushButton, QERadioButton and QECheckBox widgets should provide a slot to create new windows through the ContainerProfile class. The application can then respect the creation options set up with the new button and manage the window better – for example it may wish to add the window to its window menu. The QEGui application provides such a slot through the ContainerProfile class. Refer to the QEGui application and the Container Profile class for more details.

To write to a variable, the following properties are used:

- **variable**
If present, a value will be written to the variable when the button is operated.
The value of this variable can also be used to update the button text or image.
- **variable Substitutions**
Macro substitutions to apply to ‘variable’ and ‘altReadbackVariable’ properties. Note, the variableSubstitutions property is also applied to pressText, releaseText, and clickText properties prior to writing, is applied to the labelText property if present, and is used in any GUI filename and passed on to any new GUI launched by the QE button.
- **password**
Password user will need to enter before any action is taken.
- **confirmAction**
If true, a dialog will be presented asking the user to confirm if the button action should be carried out

- **confirmText**

If confirmAction property is true, this text will be presented to the user in the confirmation dialog. The default text is "Do you want to perform this action?"

- **writeOnPress**

If true, the 'pressText' property is written when the button is pressed. Default is false.

- **writeOnRelease**

If true, the 'releaseText' property is written when the button is released. Default is false

- **writeOnClick**

If true, the 'clickText' property is written when the button is clicked. Default is true

- **pressText**

Value written when user presses button if 'writeOnPress' property is true.

This property is also interpreted as an integer and used in the 'pressed' signal.

Note, the variableSubstitutions property is also applied to this property before writing. For example, if the property contains MY\$(ITEM) and the variable substitutions contains ITEM=CAR, MYCAR will be written.

Note, for variables with enumerated values in the database, the text must match one of the enumerated values. So if a variable is set up to display 'Off' and 'On' instead of 0 or 1, then the press text must be 'Off' or 'On', not 0 or 1.

- **releaseText**

Value written when user releases button if 'writeOnRelease' property is true.

This property is also interpreted as an integer and used in the 'released' signal.

Note, the variableSubstitutions property is also applied to this property before writing. For example, if the property contains MY\$(ITEM) and the variable substitutions contains ITEM=CAR, MYCAR will be written.

Note, for variables with enumerated values in the database, the text must match one of the enumerated values. So if a variable is set up to display 'Off' and 'On' instead of 0 or 1, then the press text must be 'Off' or 'On', not 0 or 1.

- **clickText**

Value written when user clicks button if 'writeOnClick' property is true and the button is unchecked.

This property is also interpreted as an integer and used in the 'clicked' signal when the button is unchecked.

Note, the variableSubstitutions property is also applied to this property before writing. For example, if the property contains MY\$(ITEM) and the variable substitutions contains ITEM=CAR, MYCAR will be written.

Note, for variables with enumerated values in the database, the text must match one of the enumerated values. So if a variable is set up to display 'Off' and 'On' instead of 0 or 1, then the press text must be 'Off' or 'On', not 0 or 1.

The default 'clickText' varies to suit the default 'checkable' property of the QEButton type. For QEPushButton the default 'clickText' is "1" which suits the default 'checkable' property which is 'false'. For QERadioButton and QECheckBox the default is 'clickText' is "0" which suits the

default ‘checkable’ property which is ‘true’. If the ‘checkable’ property is changed the default ‘clickText’ property is likely to be inappropriate.

- **clickCheckedText**

Text used to compare with text written or read to determine if push button should be marked as checked.

Note, must be an exact match following formatting of data updates.

When writing values, the 'pressText', 'ReleaseText', or 'clickedtext' must match this property to cause the button to be checked when the write occurs.

- **Good example:** formatting set to display a data value of '1' as 'On', clickCheckedText is 'On', clickText is 'On'. In this example, the push button will be checked when a data update occurs with a value of 1 or when the button is clicked.
- **Bad example:** formatting set to display a data value of '1' as 'On', clickCheckedText is 'On', clickText is '1'. In this example, the push button will be checked when a data update occurs with a value of 1 but, although a valid value will be written when clicked, the button will not be checked when clicked as '1' is not the same as 'On'.

This property is also interpreted as an integer and used in the ‘clicked’ signal when the button is checked.

Note, the variableSubstitutions property is also applied to this property before writing. For example, if the property contains MY\$(ITEM) and the variable substitutions contains ITEM=CAR, MYCAR will be written.

The default ‘clickCheckText’ varies to suit the default ‘checkable’ property of the QEButton type. For QEPushButton the default ‘clickCheckText’ is “0” which suits the default ‘checkable’ property which is ‘false’. For QERadioButton and QECheckBox the default is ‘clickText’ is “1” which suits the default ‘checkable’ property which is ‘true’. If the ‘checkable’ property is changed the default ‘clickCheckText’ property is likely to be inappropriate.

To read from a variable, the following properties are used:

- **subscribe**

If checked, the button will read and present the current value defined by the ‘variable’ property. If the ‘altReadbackVariable’ property is define, it is used in preference to the ‘variable’ property

- **variable**

If present, a value will be written to the variable when the button is operated.

The value of this variable can also be used to update the button text or image.

- **altReadbackVariable**

If present, the value of this variable will be used to update the button text or image if required.

- **variable Substitutions**

Macro substitutions to apply to ‘variable’ and ‘altReadbackVariable’ properties. Note, the variableSubstitutions property is also applied to pressText, releaseText, and clickText properties prior to writing, is applied to the labelText property if present, and is , and is used in any GUI filename and passed on to any new GUI launched by the QE button.

- **updateOption**

Used to determine if the data is presented textually using the button's 'text' property, or graphically using the button's 'icon' property, both textually and graphically, or if the data updates the buttons checked state.

Options are:

- Text Data updates will update the button text
- Icon Data updates will update the button icon
- TextAndIcon Data updates will update the button text and icon
- State Data updates will update the button state (checked or unchecked)

- **Pixmap0 to pixmap7**

Pixmap to display if updateOption is Icon or TextAndIcon and data value translates to an index between 0 and 7.

- **alignment**

Set the buttons text alignment.

Left justification is particularly useful when displaying quickly changing numeric data updates.

General presentation:

- **labelText**

Button label text (prior to substitution).

Macro substitutions from the variableSubstitutions property will be applied to this text and the result will be set as the button text.

Used when data updates are not being represented in the button text.

For example, a button in a sub form may have a 'labelText' property of 'Turn Pump \$(PUMPNUM) On'.

When the sub form is used twice in a main form with substitutions PUMPNUM=1 and PUMPNUM=2 respectively, the two identical buttons in the sub forms will have the labels 'Turn Pump 1 On' and 'Turn Pump 2 On' respectively.

A system command can be issued on a button click using the following properties:

- **program**

Program to run when the button is clicked.

No attempt to run a program is made if this property is empty.

Substitutions are applied to the program name.

- **arguments**

Arguments for program specified in the 'program' property.

Substitutions are applied to the arguments.

- **programStartupOption**

Option for how program is managed.

- **None:** Start and ignore the program
- **Terminal:** Start a terminal and run the program in the terminal
- **LogOutput:** Start the program and log its output to the QE message system
Content logged to the QE message system can be viewed in the Message Log in the

QEgui application, refer to ‘Logging’ (page 25) for more details on how to view content logged to the QE message system.

Some Windows commands (for example, dir) are not provided by separate applications, but by the command interpreter itself. If you specify these commands as the ‘program’ directly, it won’t work. One solution is to execute the command interpreter itself (cmd on some Windows systems), and ask the interpreter to execute the desired command. For example, the specify the ‘program’ as ‘cmd dir’. Another solution is to run the command from within a terminal (‘programStartOption’ = ‘Terminal’) where a command interpreter is started automatically.

Note, the ‘arguments’ property is only provided for convenience. It is simply appended to the ‘program’ property. An entire command can be specified in the ‘program’ property if required.

Examples:

- Start an internet browser with a specified URL:

```
program: firefox
arguments: www.google.com
programStartupOption: None
```

or

```
program: firefox www.google.com
arguments:
programStartupOption: None
```

- List the contents of the current directory: (windows example)

In this example, the ‘programStartupOption’ property is set to ‘Terminal’ so the directory output can be seen. Also, the ‘program’ argument does not need to start the command interpreter (cmd dir) as a command interpreter is started for the terminal.

```
program: dir
arguments:
programStartupOption: Terminal
```

- List the contents of the current directory: (windows example)

In this example, the ‘programStartupOption’ property is set to ‘LogOutput’ so the directory output can be seen. Also, the ‘program’ argument needs to start the command interpreter (cmd dir) as the dir command is a function built into the command interpreter.

```
program: cmd dir
arguments:
programStartupOption: LogOutput
```

- Start a python script: (windows example)
Output logged in the QE message system.

```
program:           python "C:\some path\script.py"  
arguments:  
programStartupOption: LogOutput
```

- Start a python script: (windows example)
Output in a terminal window.

```
program:           python "C:\some path\script.py"  
arguments:  
programStartupOption: Terminal
```

- Start a python script: (windows example)
Output in a terminal window as above, but the terminal window is created by the Windows ‘cmd start’ command in the ‘program’ property. Note, the ‘start’ command is built into the Windows command interpreter.

```
program:           cmd start python "C:\some path\script.py"  
arguments:  
programStartupOption: None
```

A new GUI can be started on a button click using the following properties:

- **guiFile**

File name of GUI to be presented on button click.

QWidgets use a common set of rules for locating a file. Refer to Finding files (page 27) for details.

- **creationOption**

Creation options when opening a new GUI. Open a new window, open a new tab, or replace the current window.

The creation option is supplied when the button generates a newGui signal.

Application code connected to this signal should honour this request if possible.

When used within the QEGui application, the QEGui application creates a new window, new tab, or replaces the current window as appropriate.

Options are:

- Open Replace the current GUI with the new GUI
- NewTab Open new GUI in a new tab
- NewWindow Open new GUI in a new window
- DockTop Open new GUI in a top dock
- DockBottom Open new GUI in a bottom dock
- DockLeft Open new GUI in a left dock
- DockRight Open new GUI in a right dock
- DockFloating Open new GUI in a floating dock

- **customisationName**

This name will be used to select a set of window customisations including menu items and tool bar buttons.

Applications such as QEGui can load .xml files containing named sets of window customisations.

This property is used to select a set loaded from these files.

The selected set of customisations will be applied to the main window containing the new GUI.
Customisations are not applied if the GUI is opened as a dock.

- **variableSubstitutions**

The variableSubstitutions property is applied to the GUI file name and added to the list of macro substations provided to the new form being opened by the QE button. The macro substitutions present in the variableSubstitutions property **do not** take precedence over any other macro substitutions already defined by any QEForm containing the button, or by the application. Note, the variableSubstitutions property is also used to provide default substitutions for the variable names, is applied to pressText, releaseText, and clickText properties prior to writing, and is applied to the labelText property if present.

- **prioritySubstitutions**

The prioritySubstitutions property is added to the list of macro substations provided to the new form being opened by the QE button. The macro substitutions present in the prioritySubstitutions property **do** take precedence over any other macro substitutions already defined by any QEForm containing the button, or by the application. Unlike the variableSubstitutions property, the prioritySubstitutions property is only added to the list of macro substitutions provided to a new GUI being launched by the QE button.

The prioritySubstitutions property is particularly useful when re-opening the form containing the QE button, but with different macro substitutions. The variableSubstitutions property can't be used for this since the macro substitutions it contains do not take precedence over existing macro substitutions.

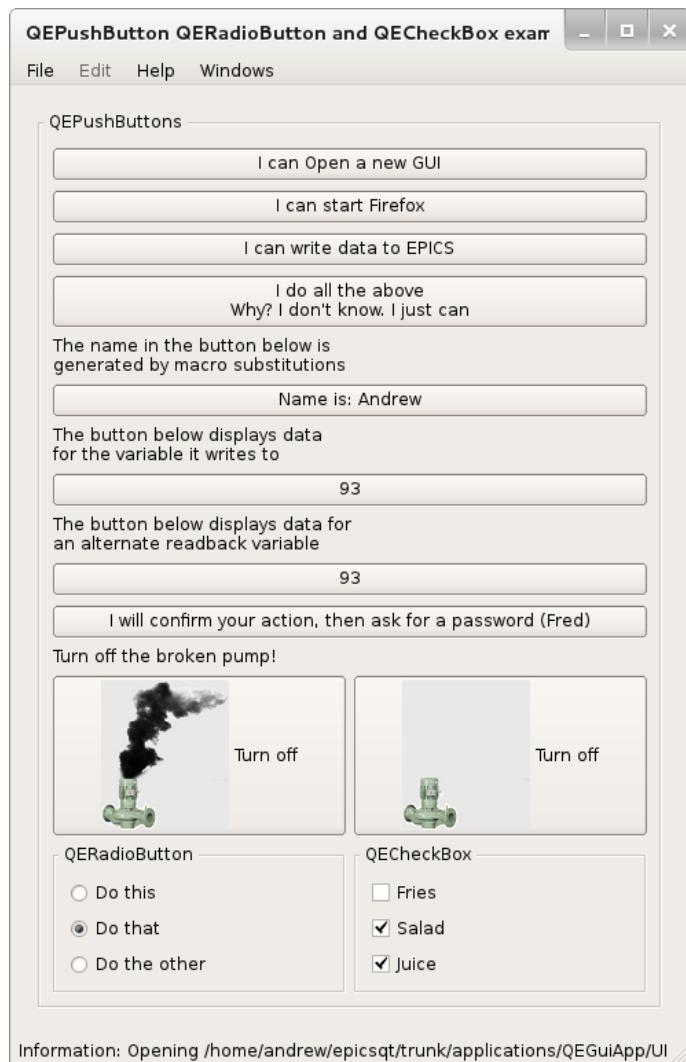


Figure 41 QEPushButton, QERadioButton and QECheckBox examples

QEPvLoadSave

The QEPvLoadSave widget is designed and provided primarily to support the in built-in PV Load/Save form included in the QEGui application. However form designers may include one or more instances of this widget on their own forms if so desired.

The QEPvLoadSave widget allows (or will eventually allow – some features are still under development) a user to define a hierarchical set of variables and apply the following actions to the whole hierarchy or a selected subset:

- Write the values to the ‘system’ – the system being whatever IOCs and other Channel Access servers are currently available to the QEGui application;
- Read the values from the system;

- c) Read the values from the archive for a user nominated time;
- d) Write the values to a file for not volatile storage. The file is an xml file - the format is described below;
- e) Read the values from a file; and
- f) Edit a nominated value.

The QEPvLoadSave widget actually supports two simultaneous and independent hierarchies, and the user is able to merge the whole hierarchy or a selected subset into the other hierarchy. The user may also request the display of the difference between the hierarchies. This is presented graphically to the user to enable him/her to quickly identify differences in the values associated between the PVs common to both sets.

Figure 42 below shows the QEPvLoadSave widget as used within the QEGui built-in form.

Tool Bar

Each hierarchy is provided with a tool bar. The functions provided by each of these buttons are:

- a)  - this button writes all values in the hierarchy to their associated PVs;
- b)  - this button reads all values in the hierarchy from their associated PVs;
- c)  - this button writes the selected sub-hierarchy values to their associated PVs;
- d)  - this button reads the selected sub-hierarchy values from their associated PVs;
- e)  - this button displays a date/time selection dialog. Once the user has selected a data and time, the archiver is accessed and the associated values extracted (*this functionality is TDB*);
- f)  or  - this button copies all values from the hierarchy and merges these into the other hierarchy;
- g)  or  - this button copies all values from the selected sub-hierarchy and merges these into the other hierarchy;
- h) Show second tree check box – this control whether the second tree (hierarchy) is displayed;

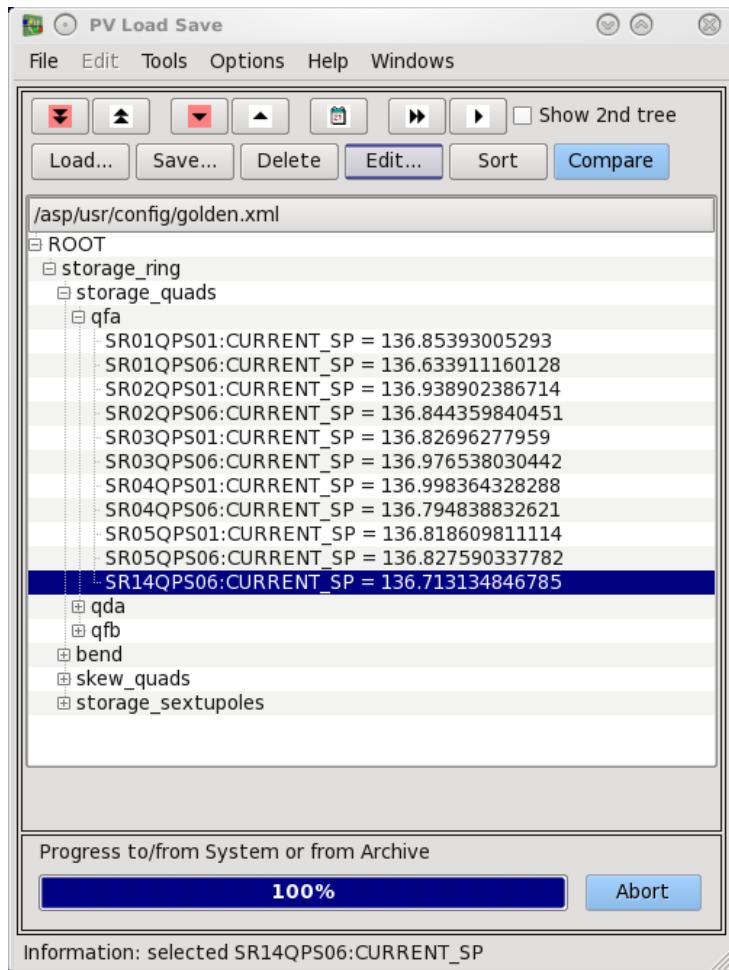


Figure 42 QEPvLoadSave – basic example.

- i) Load... - this button allows the user to navigate the file system to load a PV-Value xml file. (Optionally one may also load an old-style .pcf file as used by the AS Delphi GUI);
- j) Save... - this button allows the user to navigate the file system and select the file to save the current configuration file;
- k) Delete - this button delete the selected sub-hierarchy;
- l) Edit... - this button allows the user to edit the value of the selected PV;
- m) Sort - this button sort by PV name the selected sub-hierarchy (*this functionality is TDB*); and

- n) Compare - this button generates a graphical comparison of the two sets of PVs and their associated values. Only numerical values common to both hierarchies contribute to this graphical display (*this functionality is TDB*).

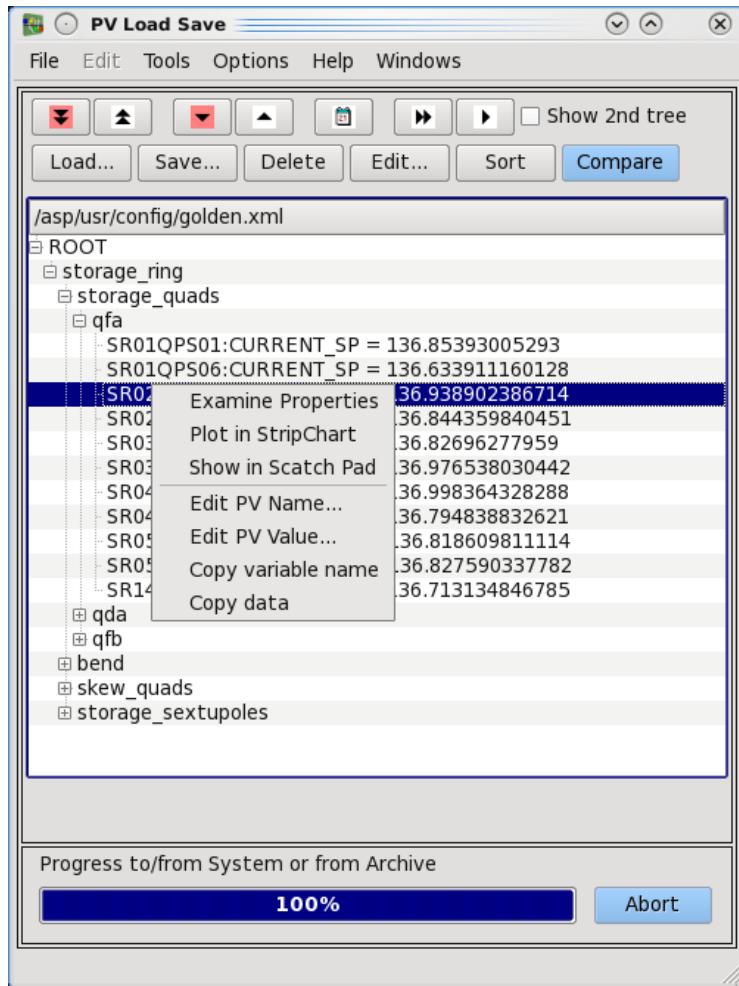


Figure 43 QEPvLoadSave – context menu example.

Context Menu

The QEPvLoadSave widget tree hierarchies provided context menus to allow the following. The content of the context menu depends on the type of item if any selected when the context menu is launched. Figure 43 above shows the context menu presented to the user when a PV node in the hierarchy is selected. Most of these items mirror those available for any EPICS aware framework widget. QEPvLoadSave specific context menu items are:

- a) Create Root (not shown in example). This is only available for an empty hierarchy and creates the root node;
- b) Add Group... (not shown in example). This allows a group to be added to the hierarchy. It is only available if the root node or another group node is selected;

- c) Rename Group... (not shown in example). This allows the group name to be modified. It is only available if a group node is selected;
- d) Add PV... (not shown in example). This allows a PV to be added to the hierarchy. It is only available if the root node or another group node is selected;
- e) Edit PV Name... This allows the PV name to be modified. It is only available if a PV node is selected;
- f) Edit PV Value.... This is allows the user to edit the value of the selected PV – is essentially the same as using the edit button as described above.

Drop

The PV name from another framework widget may be dropped onto any group node on the hierarchy. If the dropped onto node is a group node, the new PV name is added to the end of the group. The the dropped on node is a PV node, the new PV name becomes a sibling of that node, i.e. this is as if it had been dropped on the PV node's parent group node.

Note: currently one cannot drag from or between the hierarchy trees.

XML File Format

The format of the xml file used to store the hierarchy in a file is illustrated by example in Figure 44 below.

```
<QEPvLoadSave Version="1">
  <Group Name="Foo">
    <PV Type="float" Name="SR11BCM01:CURRENT_MONITOR" Value="49.9616064269032"/>
  </Group>
  <Group Name="Bar">
    <Array Number="14" Type="int" Name="SR00BLM00:ACTIVE_MAP">
      <Element Value="201" Index="0"/>
      <Element Value="202" Index="1"/>
      <Element Value="203" Index="2"/>
      <Element Value="304" Index="3"/>
      <Element Value="305" Index="4"/>
      <Element Value="406" Index="5"/>
      <Element Value="407" Index="6"/>
      <Element Value="508" Index="7"/>
      <Element Value="509" Index="8"/>
      <Element Value="610" Index="9"/>
      <Element Value="811" Index="10"/>
      <Element Value="812" Index="11"/>
      <Element Value="113" Index="12"/>
      <Element Value="114" Index="13"/>
    </Array>
  </Group>
</QEPvLoadSave>
```

Figure 44 QEPvLoadSave – xml file example.

Future Enhancements

Currently the same PV name is used for both reading from and writing to the system. A mechanism will (eventually) be developed to allow different PV names to be used. For example a PV node could be configured to read from XYZ:MOTOR.RBV (the read back field) but write to XYZ:MOTOR.VAL (the set point field).

QEPvProperties

The QEPvProperties widget displays information about a Process Variable (PV) together with a tabular view of the fields and field values of the record associated with the widget's current PV. A typical example is shown in Figure 45 (this example was a snap shot of the built-in QEGui form, accessible from the "PV Properties" menu item).

The features of this widget are:

- a) the NAME field: this shows the current process variable used to source which record is being probed, i.e. SR11BCM01:CURRENT_MONITOR.
- b) the VAL field: this shows the current value of the process variable. This is displayed using a QELabel, and as such has all the features of a QELabel such as showing the colour coded alarm state, has a tool tip and the standard QEWidget context menu, and may be dragged just like a standalone QELabel;
- c) the HOST field shows the Channel Access server providing this process variable. This will show the gateway host name as opposed to the IOC host name if the PV is being viewed through an EPICS gateway;
- d) the TIME field shows the time of the last update received for this Process Variable;
- e) the DBF field shows the PV's field type;
- f) the INDEX field shows the element number and total number of elements for the PV. This widget displays element numbers in the range 1 to N (as opposed to 0 to N-1, the display is for users, not C programmers).

Note: the QE framework currently only supports dragging and dropping, copying and pasting whole PV names as opposed to PV Name plus element number, so this field will always be of the form "1 / N" for the time being;

- g) the enumeration values section: when the DBF field is DBF_ENUM, this shows the enumeration values associated with the PV. At the bottom of the enumeration values part of the display is a pale blue bar that may be grabbed (left clicked) and dragged up or down to decrease or increase the size of this section - see example in Figure 46 below; and

- h) the field names and values table: this table is populated with the field names and the values of the (first element) of the field.

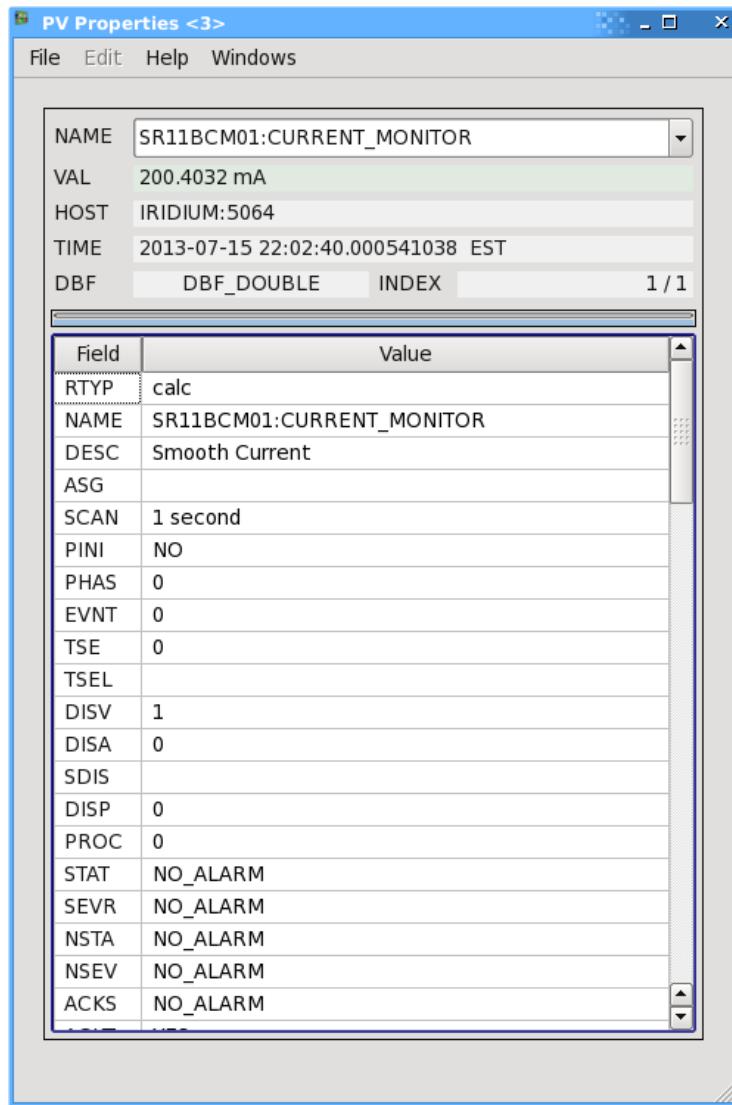


Figure 45 QEPvProperties widget example examining a calc record.

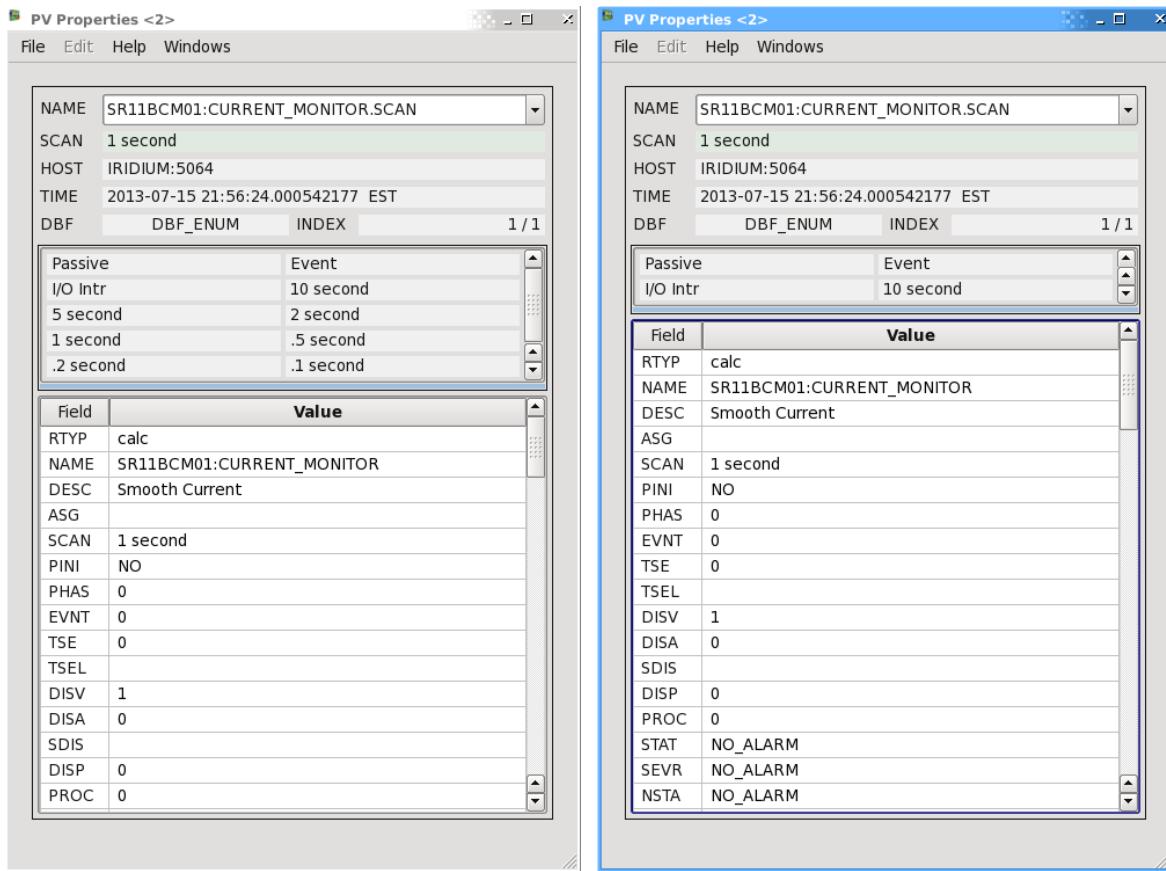


Figure 46 QEPvProperties widget example examining an enumeration PV.

Selecting a PV name

A PV name may be selected by any one of the following means:

- at design time by specifying the variableName property (together with optional substitutions);
- at run time by typing a PV name into the NAME field and pressing enter;
- by using the combo box drop down menu to select a previously used PV name;
- by dragging another EPICS aware QEWidget onto the QEPvProperties widget;
- by copying and pasting a variable name into the QEPvProperties widget;
- by opening the context menu (right-clicking) over a table field name and selecting "Properties". The "<record_name>.<field_name>" becomes the selected PV. The field names and values table is essentially unaffected by this action;
- by opening the context menu (right-clicking) over a table value field and selecting "Properties". The "Properties" item is only enabled if the widget believes the contents is a valid PV name. By repeatedly clicking on the FLNK value field, one may follow a set of FLNK records; and

- h) When running from within QEGui, by opening the context menu (right-clicking) over an EPICS aware QEWidget and selecting "Examine Properties". This will open a new instance of the "PV Properties" form and then setting up the name.

Selecting Displayed Field Names

When the QEPvProperties widget is given a new PV to probe, as well as configuring the internal QELabel, it strips off any field name to form the under-lying record name. It then attempts to read the value of the "<record_name>.RTYP" pseudo field in order to determine the record type. This is a regular channel access DBR_STRING request as opposed to a DBR_CLASS_NAME request, and as such is not stymied by an intervening gateway.

The record type is then used to access an internally held list of fields for that records type. The set of records with defined field list comprises all the records from base-3-14-11, most of the records from the synApps distribution, together with the Australian Synchrotron developed concat record, i.e. the following record types:

ai, ao, aSub, asyn, bi, bo, busy, calc, calcout, camac, compress, concat, dfanout, dpx, epid, er, erevent, event, fanout, genSub, histogram, longin, longout mbbi, mbbiDirect, mbbo, mbboDirect, mca, motor, permissive, sCalcout, scaler, scanparm, sel, seq, sscan, sseq, state, status, stringin, stringout, subArray, sub, swait, table, timestamp, transform, vme and waveform.

In each case, the record type's dbd file was processed to produce simple list of field names to which was added the RTYP field. Only the name was extract, no other filed information is used by the QEPvProperties widget other than that provided via Channel Access.

If the record type is unknown then a default list of fields is used. The default list includes the RTYPE pseudo field, fields common to all records plus the VAL field.

If the environment variable QE_RECORD_FIELD_LIST specifies a file, then this file is read and will be used to define additional record types and/or completely replace the field set of an internally specified record type. It **cannot** be used to define extra fields for an predefined record type. The format of the file is a simple ASCII file consisting of:

```
# example
-- comment lines - ignored
-- blank lines - ignored
<<record_type1>>
field_name1
field_name2
field_name3
-- introduce record type, e.g. <<aai>>
-- field name, e.g. RTYP
-- field name, e.g. DESC
-- field name, e.g. SCAN
<<record_type2>>
field_name1
field_name2
field_name3
-- introduce record type, e.g. <<ao>>
-- field name, e.g. RTYP
-- field name, e.g. DESC
-- field name, e.g. SCAN
```

All field names are associated with the preceding record type.

QERecipe

The QERecipe widget is currently under development. It will allow a user to define, save and restore a named set of variables and values. This would typically be used by a user to restore a system to a state previously identified and named by the user.

QEScratchPad

<to document>

QEScript

The QERecipe widget is currently under development. It will allow a user to specify a list of scripts to be run when requested. This widget will be useful for running a simple series of user scripts. For example, user scripts before and after a scan. Script interaction will be minimal and be limited to the ability to abort or continue with the next script in the event of a script returning an error code.

QEShape

The QEShape widget is an EPICS aware widget which displays a geometric object such as a line or a rectangle. Attributes of the object displayed in the widget can be animated by EPICS data. For example, variables representing the size and position of a beam can be used to animate the dimensions and position of an ellipse object displayed in the widget as shown in Figure 47. In addition this example also uses the variable representing beam current to animate the fill colour. The higher the beam current the more solid the fill colour.

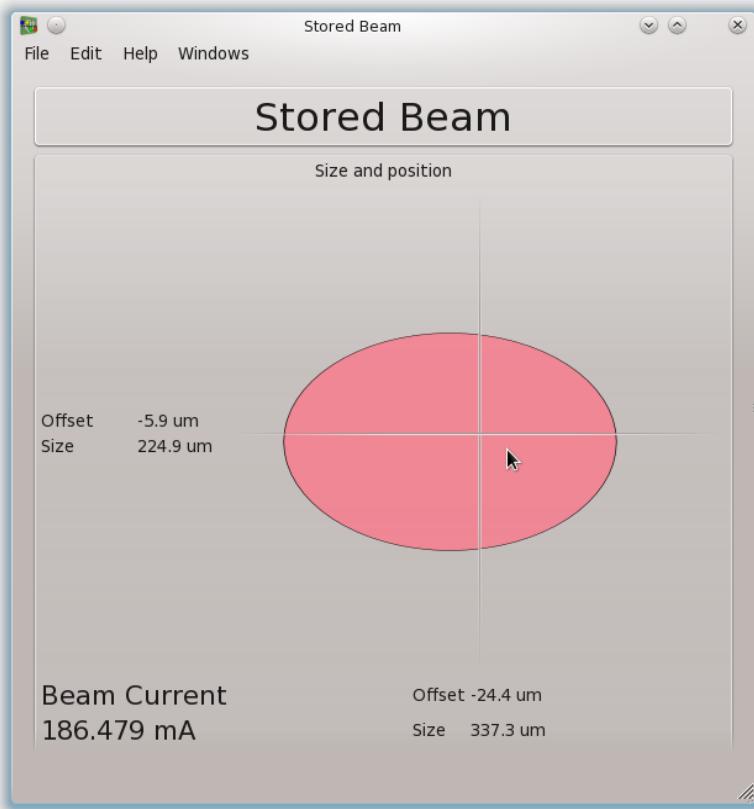


Figure 47 QEShape displaying stored beam

General configuration

To use the QEShape widget, the widget is created with enough area to draw the shape. Then:

- The required shape is selected, such as line or rectangle
- The properties defining the shape are set such as its position, size, and line thickness.
- One or more variables are set using properties ‘variable1’ to ‘variable6’.
- Scales and offsets are defined for the variables used to bring the variable values into a useful range for manipulating the shape. The scale and offset properties are ‘scale1’ to ‘scale6’ and ‘offset1’ to ‘offset6’
- The attribute to be animated by the variable is selected using properties ‘animation1’ to ‘animation6’
- Variable, scale, offset, and attribute can be set for up to six variables. The same variable can be used to animate more than one attribute.

Displayed object selection

The shapeOptions property is determines the object displayed within the widget. The following objects are available:

- Line
- Points
- Polyline
- Polygon
- Rect
- RoundedRectangle
- Ellipse
- Arc
- Chord
- Pie
- Path

Associating variable values with object attributes

Up to 6 variables can simultaneously animate various attributes of the object displayed in the widget. As each variable update occurs, the value is scaled, an offset is applied, then the modified value is used to alter any of the following attributes, usually by multiplication:

- Width
- Height
- X
- Y
- Transparency
- Rotation
- ColourHue
- ColourSaturation
- ColourValue
- ColourIndex
- Penwidth

Variables used are set by properties ‘variable1’ to ‘variable6’. Values for each variable are scaled by properties ‘scale1’ to ‘scale6’. Values for each variable are offset by properties ‘offset1’ to ‘offset6’. Values are applied to an attribute of the object by properties ‘animation1’ to ‘animation6’.

For example...

- The QEShape object shown in Figure 47 contains an ellipse 400 pixels wide.
- ‘variable1’ is set to SR10BM02IMG01:X_SIZE_MONITOR which represents beam width and has a range of 0.0 to 1000.0 um.
- ‘scale1’ is set to 0.002.
- ‘offset1’ is set to 0.0
- ‘animation1’ is set to ‘Width’

If the current beam width is 240.9 um, the ellipse will be drawn with a width of $400 \times 240.9 \times 0.002 = 192$ pixels

Properties defining objects

A common set of properties are used to define most objects that can be displayed by the QEShape widget. For example, the ‘point1’ property is used to hold the start of a line object or the top left of a rectangle object. The table below lists the relevant properties for each object:

Object Type	Property	Use
• Line	point1	Line start
	point2	Line end
	lineWidth	Thickness of line in pixels
	color1 to color10	Line color selected by value after scaling and offset.
• Points	point1 to point10	Up to 10 points displayed
	numPoints	Number of points used
	lineWidth	Diameter of points in pixels
	color1 to color10	Point color selected by value after scaling and offset.
• Polyline	point1 to point10	Up to 10 points defining the line segments
	numPoints	Number of points used
	lineWidth	Diameter of points in pixels
	color1 to color10	Line color selected by value after scaling and offset.
• Polygon	point1 to point10	Up to 10 points defining the line segments
	numPoints	Number of points used
	drawBorder	Set if border is required
	fill	Set if fill is required
	lineWidth	Line thickness of border in pixels
	color1 to color10	Fill color selected by value after scaling and offset.
• Rect	point1	Top Left
	point2	Size
	drawBorder	Set if border is required
	fill	Set if fill is required
	lineWidth	Line thickness of border in pixels
	color1 to color10	Fill color selected by value after scaling and offset.
• RoundedRect	point1	Top Left
	point2	Size
	drawBorder	Set if border is required
	fill	Set if fill is required
	lineWidth	Line thickness of border in pixels
	color1 to color10	Fill color selected by value after scaling and offset.

Object Type	Property	Use
• Ellipse	point1	Top left of rectangle enclosing ellipse
	point2	Size of rectangle enclosing ellipse
	drawBorder	Set if border is required
	fill	Set if fill is required
	lineWidth	Line thickness of border in pixels
	color1 to color10	Fill color selected by value after scaling and offset.
• Arc	point1	Top left of rectangle enclosing ellipse of which arc is a part
	point2	Size of rectangle enclosing ellipse of which arc is a part
	startAngle	Start angle in degrees. Zero is at 3 o'clock incrementing anti clockwise
	arcLength	Arc span in degrees incrementing anti clockwise.
	lineWidth	Line thickness of arc in pixels
	color1 to color10	Line color selected by value after scaling and offset.
• Chord	point1	Top left of rectangle enclosing ellipse of which chord is a part
	point2	Size of rectangle enclosing ellipse of which chord is a part
	startAngle	Start angle in degrees. Zero is at 3 o'clock incrementing anti clockwise
	arcLength	Arc span in degrees incrementing anti clockwise.
	drawBorder	Set if border is required
	fill	Set if fill is required
• Pie	point1	Top left of rectangle enclosing ellipse of which pie is a part
	point2	Size of rectangle enclosing ellipse of which pie is a part
	startAngle	Start angle in degrees. Zero is at 3 o'clock incrementing anti clockwise
	arcLength	Arc span in degrees incrementing anti clockwise.
	drawBorder	Set if border is required
	fill	Set if fill is required
• Path	lineWidth	Line thickness of border in pixels
	color1 to color10	Fill color selected by value after scaling and offset.
	point1	Start point
	point2	First control point
	point3	Second control point
	point4	End point
• Path	drawBorder	Set if border is required
	fill	Set if fill is required

Object Type	Property	Use
	lineWidth	Thickness of line in pixels
	color1 to color10	Fill color selected by value after scaling and offset.

Properties defining object views

The ‘rotation’ and ‘originTranslation’ properties apply to all objects as they affect how the widget is viewed, not how it is drawn.

By default the origin (position 0,0) of the object drawing area is located at the top left of the QEShape widget. This origin can be moved within the QEShape widget using the ‘originTranslation’ property. Since variable data is often used to scale the objects geometry, it is often useful to have the origin somewhere other than top left as geometry is scaled around the drawing area origin.

In Figure 48, four QEShape widgets are shown. Each draws a 40x40 pixel ellipse object and has a variable animating both the ellipse width and height. The left hand pair have an ellipse starting at (0,0) and no offsetTranslation. This means the top left of the QEShape widget is at the origin of the object drawing area and scaling will be towards or away from the top left corner of the widget. The right hand pair have an ellipse starting at (-20,-20) and an offsetTranslation of (-40,-40). An offsetTranslation of (-40,-40) means the top left of the QEShape widget is located at position (-40,-40) of the object drawing area. This places the origin of the drawing area at the centre of the QEShape widget. As the ellipse is being drawn around the origin of the drawing area and which is now in the centre of the widget, the ellipse appears in the centre of the QEShape widget and is scaled around the centre.

The difference is in how the object expands as the width and height are scaled by the data value changing from 1 to 2 is shown in the top and bottom widgets respectively. The left hand QEShape widgets show the ellipse growing out from the top left hand corner, the right hand QEShape widgets show the ellipse growing around the centre of the widget.

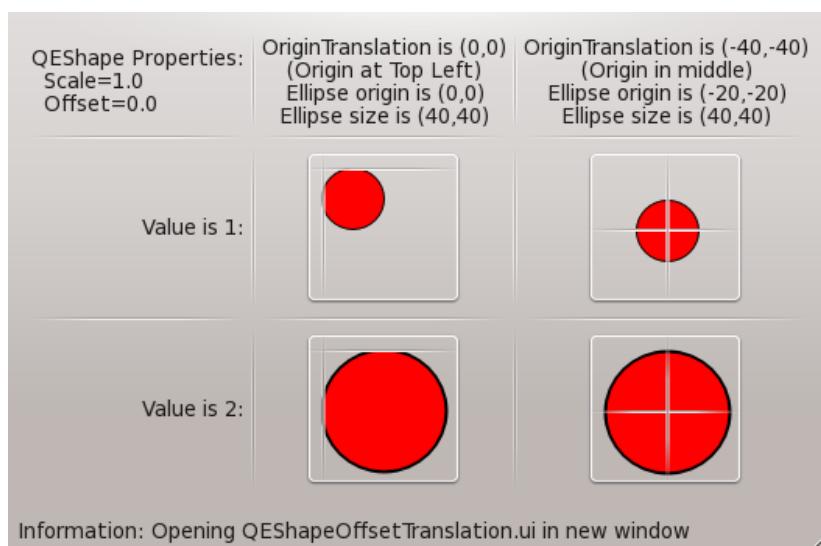


Figure 48 QEShape originTranslation example

In Figure 49 a single QEShape widget is shown implementing a meter needle on a background of a meter scale. The QEShape widget draws a line object and has a variable animating the line rotation. The ‘originTranslation’ property has been set to (-118,-124) to place the origin of the drawing area in the centre of the meter, and the line coordinates have been set to (0,20) (0,-100) to draw the line through the origin. ‘scale1’ has been set to 2.63 to convert a variable value range of 0-100 to a rotation of 0 to 270 degrees. ‘offset1’ has been set to -130 degrees so the line starts at the zero point on the scale for a variable value of zero.

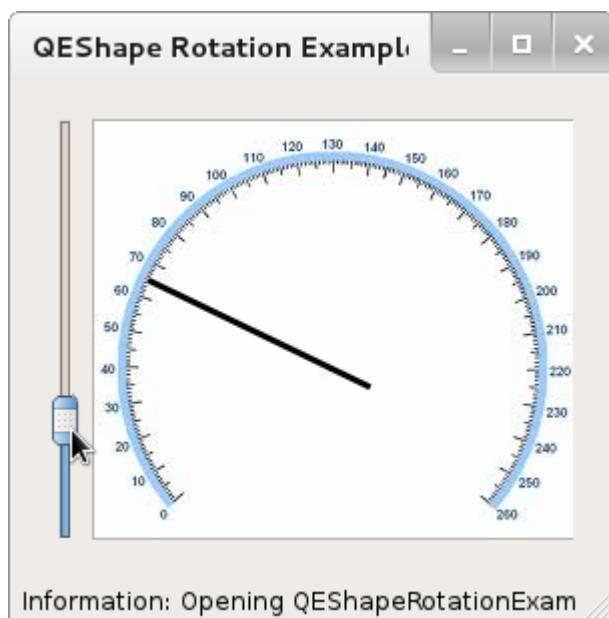


Figure 49 QEShape rotation example

Traps

The QEShape widget provides a view onto the drawing area where the shape is created. The shape may seem to disappear if the properties defining the geometry of the shape places it outside the area that can be seen by the QEShape widget, or variable values have modified the shape’s position so it is no longer viewable within the QEShape widget.

QESimpleShape

The QESimpleShape widget is an EPICS aware widget which uses either the alarm state or the value of a single PV to determine the colour of the shape. The shape itself is determined by the widget’s shape property, and may be one of: circle, ellipse, rectangle, roundRectangle, roundSquare, square, triangleUp, triangleDown, triangleLeft, triangleRight, diamond, or equalDiamond. The size of the shape is maximised to just fit within the geometry of the widget. For circle, square, roundSquare and equalDiamond the size is determined by the lesser of the widget’s width and height.

When the `displayAlarmState` (above) property is set true (the default), the colour of the widget is determined by the alarm state of the PV. Standard framework alarm colours are used, i.e. green for no alarm, yellow for minor alarm, red for major alarm and white for invalid alarm.

When the `displayAlarmState` property is set false, the value of the PV is used to select a colour from a set of 16 colour properties, i.e. `color0`, `colour1`, and so on to `colour15`. The value of the PV must be capable of being interpreted as an integer. Modulo 16 arithmetic is used to ensure the PV value yields a number in the range 0 to 15. The selection of the number of colours properties was some-what arbitrary. 16 was chosen so that a colour could be associated with each value of an `mbbi`/`mbbo` record.

Figure 50 below shows examples of this widget. All the `QESimpleShape` widgets are monitoring the same PV and have geometries which all have a width of 40 and a height of 20. The first row of widgets all have `displayAlarmState` set to false, and are blue because the value of the PV is 2 and `color2` property has been set to blue. The second row of widgets all have `displayAlarmState` set to true, and are green because the PV's severity is no alarm (the third row contains a `QELabel` which shows the actual value of the PV).

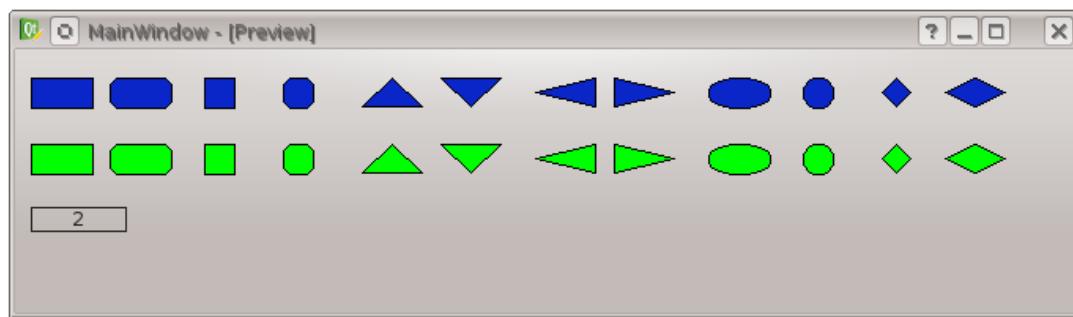


Figure 50 `QESimpleShape` examples

When disconnected the `QESimpleShape` is displayed as washed-out gray with a light gray border.

Figure 51 below shows the properties values selected for the second row of widgets.

QE Simple Shape Properties	
> variable	SR00IE01:INPUT_TRIGGER_COUNT
> variableSubstitutions	
variableAsToolTip	<input checked="" type="checkbox"/>
enabled	<input checked="" type="checkbox"/>
allowDrop	<input type="checkbox"/>
visible	<input checked="" type="checkbox"/>
messageSourceId	0
> userLevelUserStyle	
> userLevelScientistStyle	
> userLevelEngineerStyle	
userLevelVisibility	User
userLevelEnabled	User
displayAlarmState	<input type="checkbox"/>
shape	diamond
> colour0	█ [200, 44, 17] (255)
> colour1	█ [8, 200, 8] (255)
> colour2	█ [10, 38, 200] (255)
> colour3	█ [200, 43, 85] (255)
> colour4	█ [200, 133, 61] (255)
> colour5	█ [57, 200, 196] (255)
> colour6	█ [255, 255, 0] (255)
> colour7	█ [255, 255, 255] (255)
> colour8	█ [58, 58, 58] (255)
> colour9	█ [169, 83, 200] (255)
> colour10	█ [200, 198, 75] (255)
> colour11	█ [75, 134, 70] (255)
> colour12	█ [200, 64, 23] (129)
> colour13	█ [11, 200, 17] (97)
> colour14	█ [14, 14, 200] (102)
> colour15	█ [255, 0, 255] (255)

Figure 51 QE Simple Shape properties

QESlider

The QESlider widget provides the ability to display and modify the value of a single PV using a slider. This widget is derived from QSlider. The example in Figure 52 shows several QESlider widgets connected to a variable. The QESlider subscribes to the variable by default (subscribe property set by default).

For many variables, the standard QSlider ‘minimum’ and ‘maximum’ properties can be used to set the range of the slider to match the variable data. This is not adequate for some variables. For example an appropriate integer maximum and minimum cannot be set if the variable is a floating point type with a range of 0.0 to 1.0. In cases like this the QESlider ‘scale’ and ‘offset’ properties can be used to prescale the variable to allow sensible QSlider ‘maximum’ and ‘minimum’ values. For example a scale of 1000 and a maximum of 1000 would allow a floating point value of 0.0 to 1.0 to be set with a precision of 0.1 (as long as the slider had a range of at least 1000 pixels).

Scale and offset properties

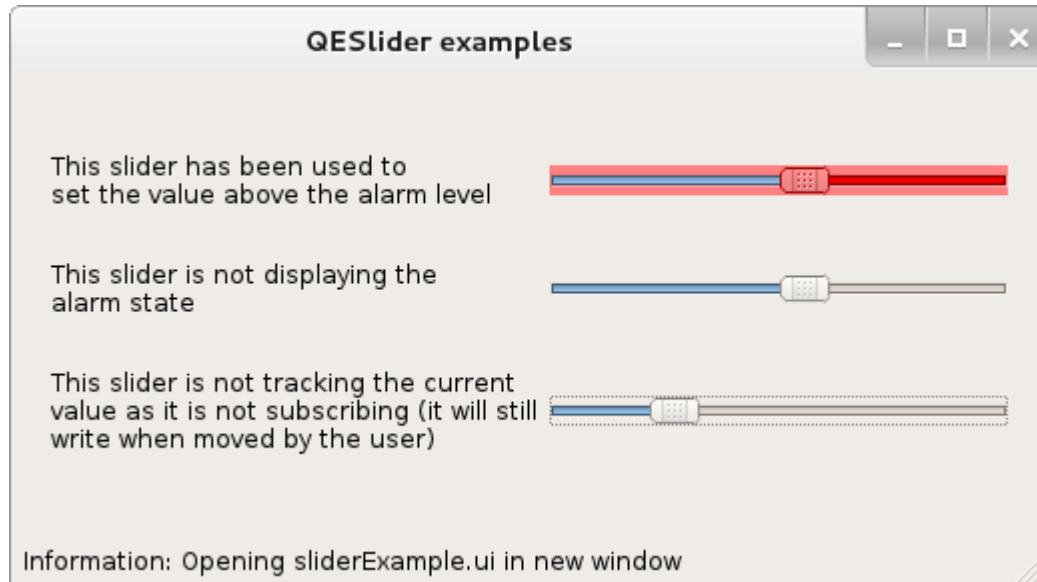


Figure 52 QESlider examples

QESpinBox

The QESpinBox widget provides the ability to display and modify the value of a single PV using a spin box. This widget is derived from QDoubleSpinBox. For variables with a large range, QESpinBox may not be the best choice as the step size is set at design time. In these instances, a QENumericEdit widget may be more appropriate. The example in Figure 53 shows several QESpinBox widgets, some appropriate for the variable range and some not so appropriate

The ‘addUnits’ property will set the ‘suffix’ property to the engineering units read for the variable from the database. Alternately the ‘suffix’ property can be set directly. When set directly ‘addUnits’ must be cleared or ‘suffix’ will be overwritten with the database value.

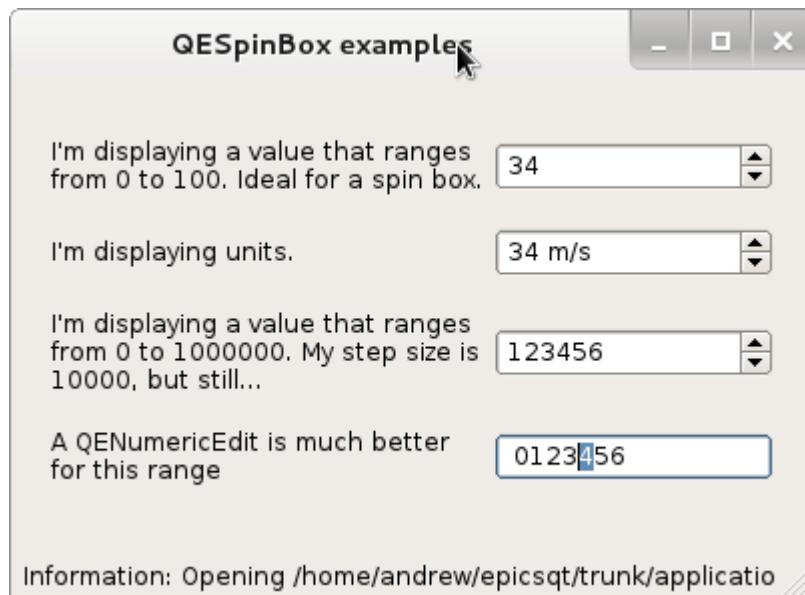


Figure 53 QESpinBox examples with a QENumericEdit where more appropriate

QEStripChart

<to document>

QESubstitutedLabel

A QESubstitutedLabel adds macro substitution capability to a standard QLabel widget. A QESubstitutedLabel widget with macros in the text is typically used in a form to produce varying text depending on the macro substitutions used on the form. For example, a form may include a QESubstitutedLabel with the text ‘Pump \$(NUM)’ as a title. If the macro substitutions applied to one instance of the form include ‘NUM=1’ and ‘NUM=2’ for another, the form title labels will be ‘Pump 1’ and ‘Pump 2’ respectively. Another example of using a QESubstitutedLabel to vary a title in multiple instances of a sub form is shown in Figure 54

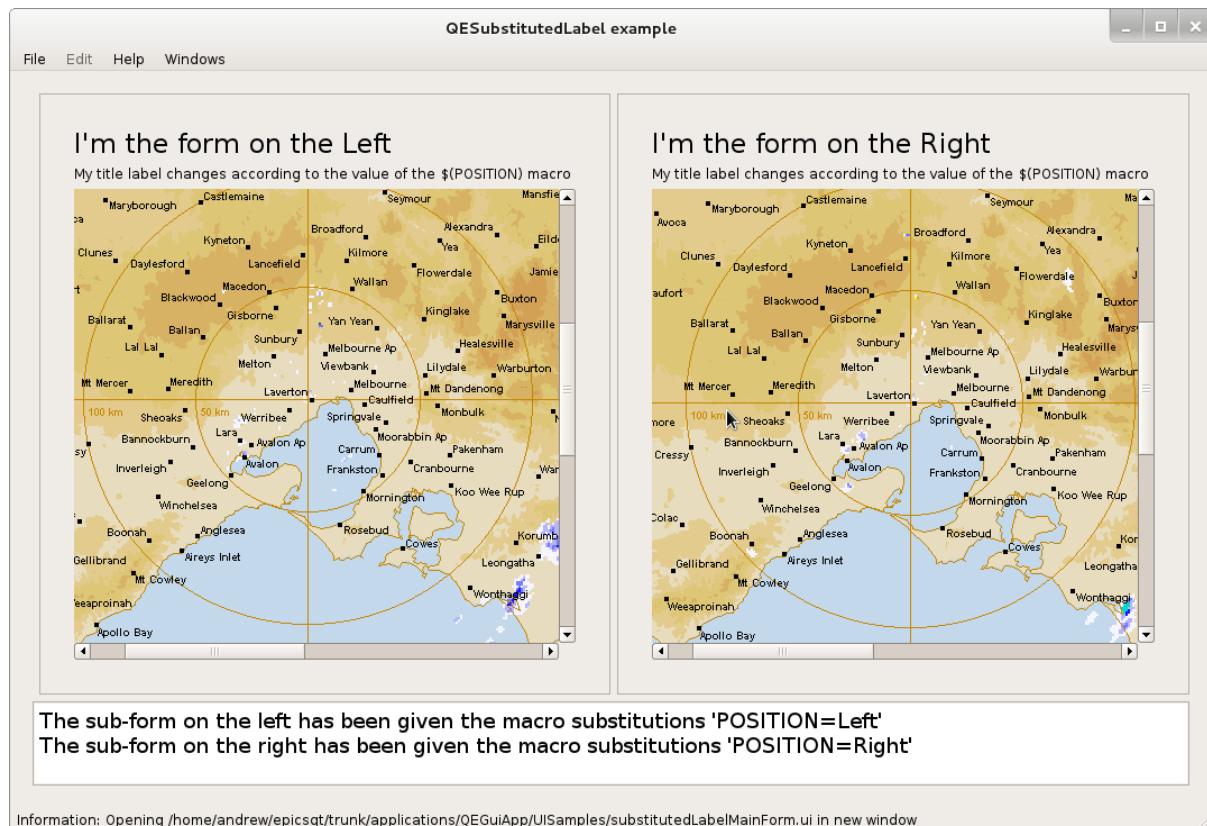


Figure 54 QESubstitutedLabel used to vary title in sub forms

QERadioGroup

The QERadioGroup widget comprises of a standard group box with a number of embedded radio buttons. Each button is presented with an enumeration value as the button text. Essentially this widget provides the same functionality that is provided by QEComboBox widget, albeit presented very differently. On selection of one of the embedded buttons, the underlying value is written to the associated PV. Typically a QERadioGroup widget would be used with an mbbo record.

As with the QEComboBox, within Qt's designer, the user may elect to use the enumeration strings that are defined in the database and these will be assigned to the buttons within the radio group if the 'useDbEnumerations' property is set (the default). If the 'useDbEnumerations' property is not set, then the strings used by the radio group for each variable value must be set up in localEnumeration property (see String formatting properties, localEnumeration for details).

The example in Figure 55 shows two QERadioGroup widgets connected to the same mbbo record. The widget on the left is using the database provided enumeration strings, and the widget on the right is using the enumeration values defined using the localEnumeration property.

The `columns` property can be used to set the number of columns (in the range 1 to 16, default is 2).

QERadioGroup	
<input type="radio"/> No Action	<input type="radio"/> Stop
<input type="radio"/> Continuous	<input type="radio"/> One Shot
<input type="radio"/> Pattern Fill	<input type="radio"/> Pattern Resume
<input type="radio"/> Next Shot	<input checked="" type="radio"/> Top Up
<input type="radio"/> Recovery	<input type="radio"/> Periodic
QERadioGroup	
<input type="radio"/> None	<input type="radio"/> Halt
<input type="radio"/> Always	<input type="radio"/> One-off
<input type="radio"/> Filling	<input type="radio"/> More filling
<input type="radio"/> One more shot	<input checked="" type="radio"/> Steady current
<input type="radio"/> Help	<input type="radio"/> Regular

Figure 55 QERadioGroup example showing local and database defined enumeration strings

Appendix A

GNU Free Documentation Licence

GNU Free Documentation License
Version 1.3, 3 November 2008

Copyright (C) 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.
<<http://fsf.org/>>

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondary, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of

the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of

the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent

copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise

the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME.
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

A copy of the license is included in the section entitled "GNU

Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.