



# QE Framework - QE Widget Specifications

Andrew Rhyder

Andrew Starritt

24<sup>th</sup> July 2020

Copyright (c) 2019-2020 Australian Synchrotron

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 .....	5
License.....	5
QAnalogIndicator and QAnalogProgressBar .....	5
QArchiveStatus.....	6
QAnalogSlider and QAnalogSlider.....	6
QAnalogSlider .....	7
QAnalogSlider .....	9
QBitStatus and QBitStatus .....	9
QComboBox .....	9
QConfiguredLayout .....	10
QCorrelation.....	12
QFileBrowser.....	14
QFileImage .....	15
QForm .....	17
QFormGrid .....	19
Properties.....	20
Nested QFormGrid .....	21
Examples .....	21
QFrame and QPvFrame.....	23
QGeneralEdit.....	24
QGroupBox.....	26
QImage .....	26
QLabel .....	26
QDescriptionLabel.....	29
QELCDNumber .....	29
QLineEdit.....	30
QELink .....	31
QCalcout.....	32
QLog .....	33
QLogin .....	34
QNumericEdit and QNumericEdit .....	37
QPeriodic.....	37

QEPlot .....	42
QEPlotter.....	42
Expressions.....	43
Scaling and Presentation Control.....	44
QEPushButton, QERadioButton and QECheckBox.....	44
QEMenuButton .....	55
General Description .....	55
Restrictions .....	57
Customisation Menus .....	57
QEPvLoadSave.....	57
Tool Bar .....	58
Context Menu .....	60
Drop .....	61
XML File Format .....	61
Future Enhancements.....	62
QEPvLoadSaveButton.....	62
QEPvProperties .....	63
Selecting a PV name.....	65
Selecting Displayed Field Names .....	66
QRadioGroup and QERadioGroup .....	67
QERecipe .....	67
QEScratchPad.....	67
QEScript.....	68
QEScalarHistogram and QEWaveformHistogram .....	70
QEShape .....	72
QESimpleShape .....	79
QESlider.....	82
QESpinBox.....	83
QEStripChart .....	84
QESubstitutedLabel.....	84
QETable .....	85
QENTTable .....	86
Appendix A .....	87

GNU Free Documentation Licence.....	87
-------------------------------------	----

### Introduction

This document describes in detail the various widgets provided by the EPICS Qt, aka QE, Framework.

This document was created by extracting the widget specification information from the QE QEGui and User Interface Design document. The main reason for this is ease of maintenance and avoiding editing unwieldily large word documents. In future, specific widgets or groups of widgets may also be extracted into their own documentation as well.

### License

The QE Framework is distributed under the GNU Lesser General Public License version 3, distributed with the framework in the file LICENSE. It may also be obtained from here:

<http://www.gnu.org/licenses/lgpl-3.0-standalone.html>

### 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 class 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
- The QEAnalogProgressBar widget has an arrayIndex property that can be used to select a single element from an array of data to provide the analog value. The default is 0.

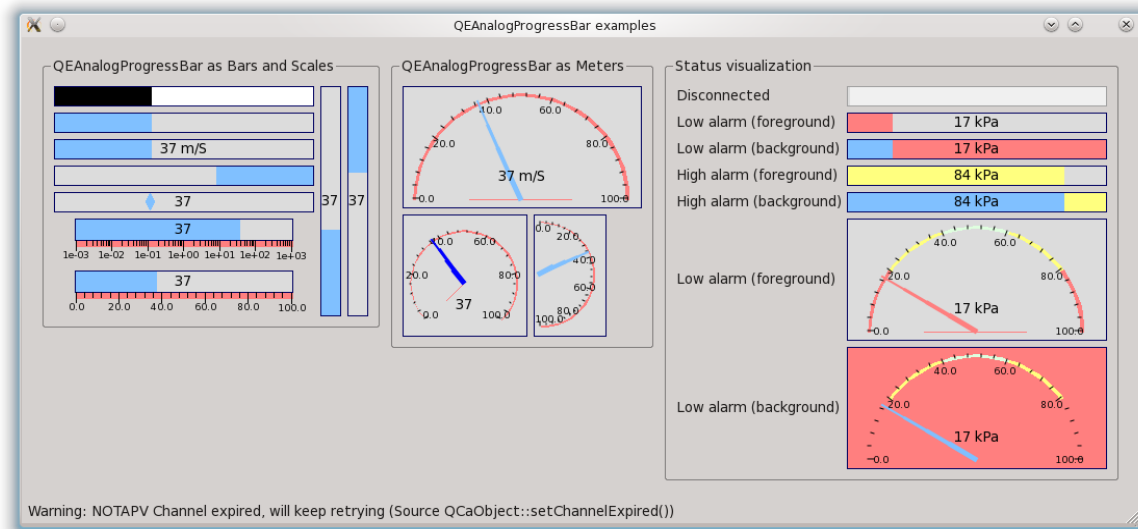


Figure 1 QEAnalogProgressBar examples

## 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 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 2 QEArchiveStatus example

## QAnalogSlider and QEAnalogSlider

The QAnalogSlider is a non-EPICS aware slider widget that provides an analog equivalent of the QSlider. It is deemed analog as it can be set by and emits a floating point (double) value as opposed to integer value. It is also decorated with a scale and text label showing the current value, and also provides a local save and restore capability.

## QE Framework – Widget Specifications

Unlike its QSlider counter-part, a QAnalogSlider is always horizontal and (currently, at least) always increases in value from left to right.

Just as QESlider inherits from QSlider and extends it by providing EPICS awareness, QEAnalogSlider directly inherits from QAnalogSlider and extends QAnalogSlider by providing EPICS awareness.

### QAnalogSlider

QAnalogSlider itself inherits directly from QFrame, although the default frameShape property is NoFrame. Internally, the QAnalogSlider uses a QSlider widget to provide the slider control. The widget also has some internal text labels and buttons which are described below.

Figure 3 below shows five examples of QAnalogSlider.

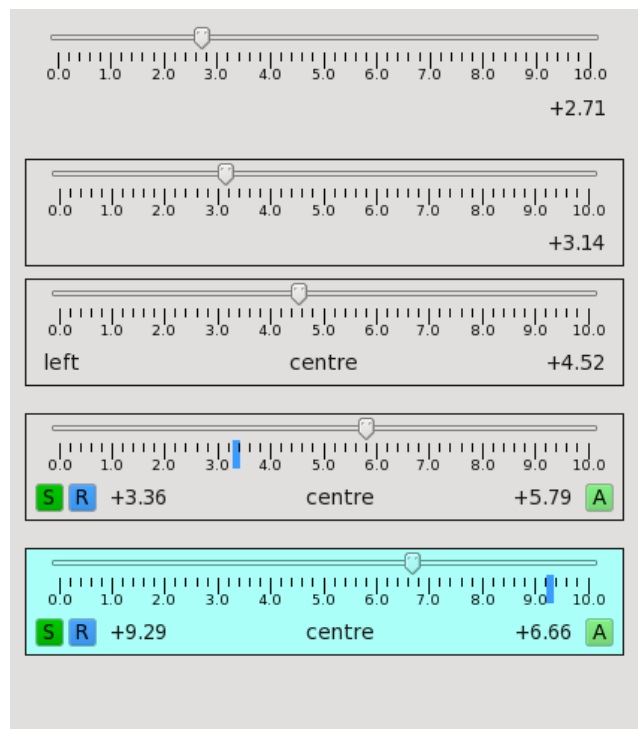


Figure 3 QAnalogSlider examples

The first shows a slider that allows a value in the range 0.0 to 10.0 to be selected. The slider has been set to 2.71 - note that the text in the lower right hand side of the widget also shows the current value. If tracking is enabled (the default), the slider emits the valueChanged () signal while the slider is being dragged. If tracking is disabled, the slider emits the valueChanged () signal only when the user releases the slider.

The second is similar to the first example, save that the QFrame frameShape property set to Box and the slider itself has been set to 3.14.

## QE Framework – Widget Specifications



The third is similar to the second example, however the `leftText` and `centreText` properties have been set to "left" and "centre" respectively.

The fourth example shows a `QAnalogSlider` with both the `showSaveRevert` and the `showApply` boolean properties set to true. The `showSaveRevert` cause a green save (S) and a blue revert (R) button pair to be display in the lower left hand side of the widget. The `leftText` property is ignored and the left label now used to show the saved position (i.e. 3.36 in this example). The saved position is also indicated graphically by a blue bar painted on the scale. The current slider value can be saved by clicking on the save button. Clicking on the revert button will set the slider current value to the saved value. The slider emits the `valueChanged ()` signal after a revert.

Clicking on the apply button causes the current value to be emitted via the `appliedValue ()` signal. The `valueChanged()` and `appliedValue()` signals both emit the value of the widget as a double number, and apart from the name, have the same signature. The rationale for a separate signal instigated by the apply button is to allow any such value changes to be used/applied specifically as a result of the user requesting it.

The last example shows a `QAnalogSlider` with a non-default style-sheet.

### Properties Summary

Name	Type	Default	Description
value	double	0.0	The widget's value. The value is constrained to be within <i>minimum</i> to <i>maximum</i> .
precision	integer	2	Specified the precision used for the current value and saved value texts. The value is constrained to be in the range 0 to 12.
minimum	double	0.0	Defines the minimum allowed slider value. The widget ensures <i>maximum</i> is always no less than <i>minimum</i> .
maximum	double	10.0	Defines the maximum allowed slider value. The widget ensures <i>minimum</i> is always no greater than <i>maximum</i> .
minorInterval	double	0.2	Defines the axis marker interval displayed without any associated scale text. The value, <i>x</i> , is constrained such that: $x \geq (maximum - minimum) / 1000.0$
majorInterval	double	1.0	Defines the axis marker interval displayed with some associated scale text. The value is constrained to be a positive integer multiple of <i>minorInterval</i> .
tracking	bool	True	If tracking is enabled, the slider emits the <code>valueChanged ()</code> signal while the slider is being dragged. If tracking is disabled, the slider emits the <code>valueChanged ()</code> signal only when the user releases the slider.



## QE Framework – Widget Specifications



leftText	string	""	Specifies the text displayed on the lower left hand side of the widget. Note: if <i>showSaveRevert</i> is enabled, this text field if a commandeered to display the current saved value.
centreText	string	""	Specifies the text displayed on the lower centre of the widget.
rightText	string	""	Specifies the text displayed on the lower right hand side of the widget. Note: this field is will be overwritten by the slider value whenever the slider is used. Anything other than the default value is probably of little use.
showSaveRevert	bool	False	When enabled, activates the save restore capability.
showApply	bool	False	When enabled, this makes the apply button visible. When pressed this causes the widget to emit <i>appliedValue</i> signal.

### QEAnalogSlider

This is an EPICS aware widget based on the QAnalogSlider widget. Details are TBD.

### QBitStatus and QEBitStatus

The QBitStatus and QEBitStatus widgets are now described in own document.

### 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 4 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 QE\_QEGuiAndUserInterfaceDesign.docx 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).

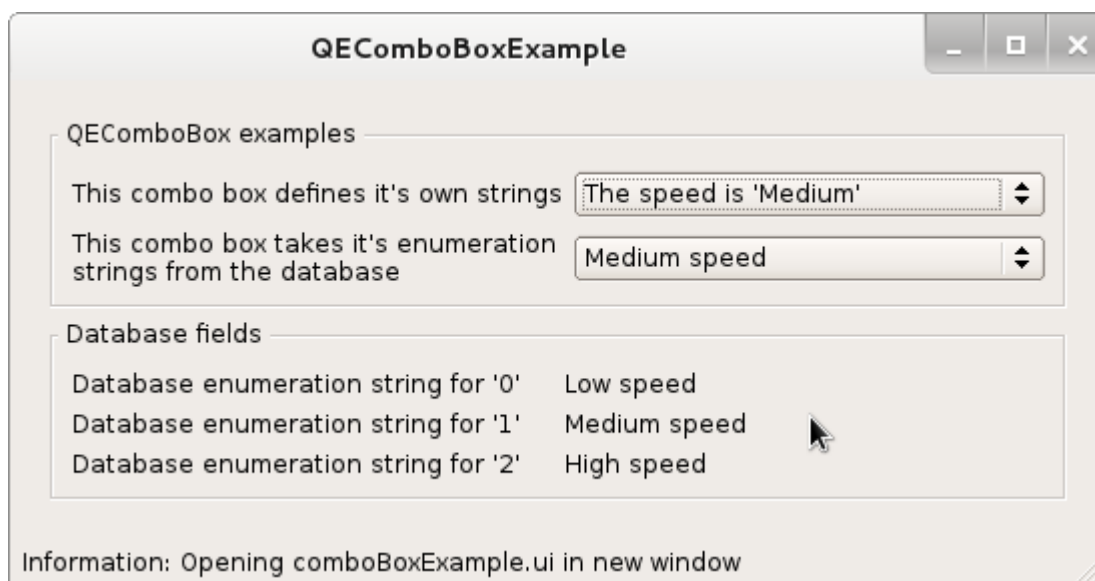


Figure 4 QEComboBox example showing local and database defined enumeration strings

### 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.

The widget can include drop down menu for selecting one of a number of items to display using the 'showItemList'.

If the XML definition is stored in a file, the 'configurationFile' property must reference that file and the 'configurationType' property must be set to 'File'. The file is located using the rules defined in '**Error! Reference source not found.**' (page **Error! Bookmark not defined.**). Alternatively the XML may be defined directly in the 'configurationText' property in which case the 'configurationType' property must be set to 'Text'.

The following is a sample of sample XML defining two motor stages where each is stage has a set point and readback for 2 axis. The result of this XML is shown in Figure 5.

```
<epicsqt>
  <item name="First Stage">
    <field name="X Set point" processvariable="STAGE1:X_SET" type="linedit"/>
    <field name="Readback" processvariable="STAGE1:X" type="label" join="true"/>
  </item>
</epicsqt>
```

## QE Framework – Widget Specifications

```
<field name="Y Set point" processvariable="STAGE1:Y_SET" type="linedit"/>
<field name="Readback" processvariable="STAGE1:Y" type="label" join="true"/>
</item>
<item name="Second Stage">
  <field name="Z1 Set point" processvariable="STAGE2:Z1_SET" type="linedit"/>
  <field name="Readback" processvariable="STAGE2:Z2" type="label" join="true"/>
  <field name="Z2 Set point" processvariable="STAGE2:Z2_SET" type="linedit"/>
  <field name="Readback" processvariable="STAGE2:Z2" type="label" join="true"/>
</item>
</epicsqt>
```

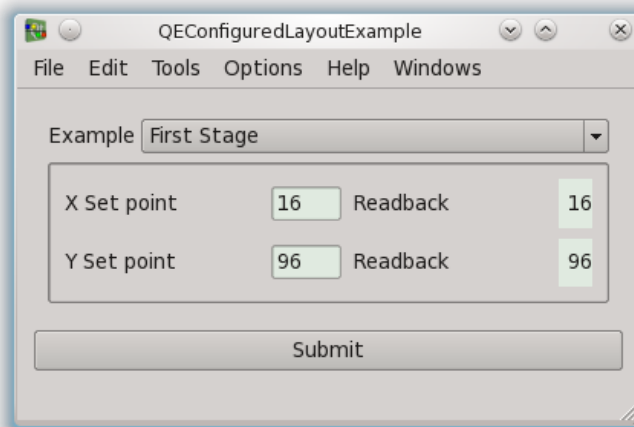


Figure 5 QEConfiguredLayout example

The following table defines the XML elements and tags that may be used to define the layout of a QEConfiguredLayout:

Tag name	Element description	Attributes (*Mandatory)	Child element tags (*Mandatory)
epicsqt	A single element with this tag is expected in each configured layout xml definition.		item

## QE Framework – Widget Specifications



Tag name	Element description	Attributes (*Mandatory)	Child element tags (*Mandatory)
item	A user selectable configured layout.	name substitution visible	field
field	A field in the layout	name processvariable join type group visible editable	

### QECorrelation

The QECorrelation inherits directly from QEAbstractDynamicWidget, which in turn inherits from QEFrame.

The QECorrelation widget provides both graphical and textual information about the correlation between two scalar PVs. See example in Figure 6 below.

The two PV names may be drag/dropped onto the widget, copy/pasted to the widget, or entered using the PV name selection dialog accessed from the X and Y buttons, or via the context menu associated with the (red and green) PV name labels. This is similar to the way PVs are added to QEStripChart and QEPlotter widgets.

As the two PVs may update independently, the QECorrelation widget samples both PV values at a fixed time interval (user selectable from 0.2 to 3600 seconds) to form a pair of X and Y values.

The number of PVs pair values retained to provide the correlation information is also user selectable (from 4 to 5000 pairs). The data is presented graphically as a set of points (as in the example) or as a line. The calculated correlation value is also presented.

PVs pair values may also be retrieved for the archive.

The functionality may be access via the QEGui built-in PV Correlation form (via menu **Tools | PV Correlation...**) or by placing a QECorrelation widget within a user defined form. In the latter case, the PV variable names can be set using the variableX and variableY properties accessible when using designer.

## QE Framework – Widget Specifications

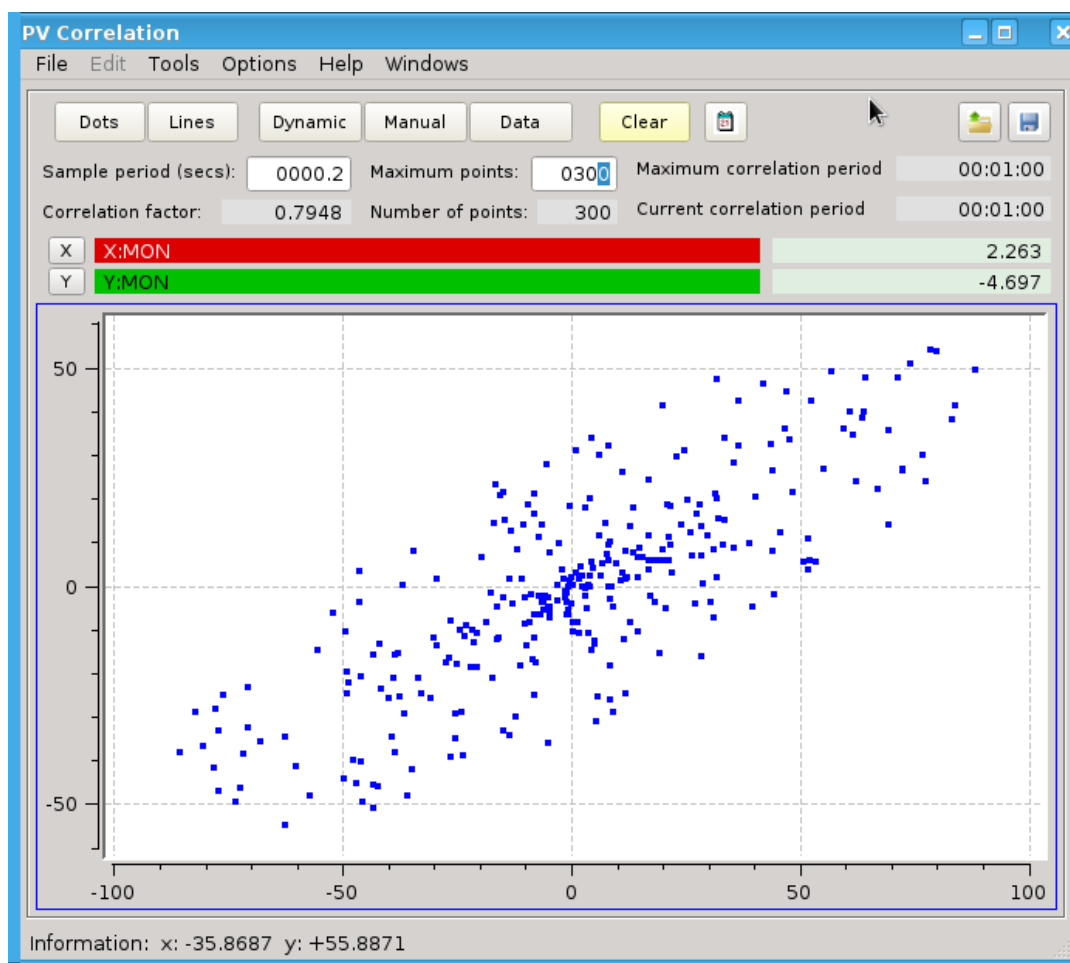


Figure 6 QECorrelation

The QECorrelation has the following properties (that can be controlled by the user):

property	description
variableX	The x-axis process variable name
variableY	The y-axis process variable name
variableSubstitutions	The default substitutions to be applied to the PV names
sampleInterval	The sample interval
numberPoints	The number of points to be store/plotted.
xLogarithmic	When true/checked, the x-axis is displayed using a log scale
yLogarithmic	When true/checked, the y-axis is displayed using a log scale

### QEFileBrowser

The QEFileBrowser widget allows the user to browse existing files from a certain directory.

When connected to a QEFileImage it can be used to select the file being viewed. In this scenario, the QEFileBrowser 'selected' signal is connected to the QEFileImage 'setImageFileName' slot. Note, this is only one method a QEFileImage widget can use to source its image. Refer to 'QEFileImage' (page 15) for more details.

Within Qt Designer, it has the following graphical representation (surrounded by a red rectangle):

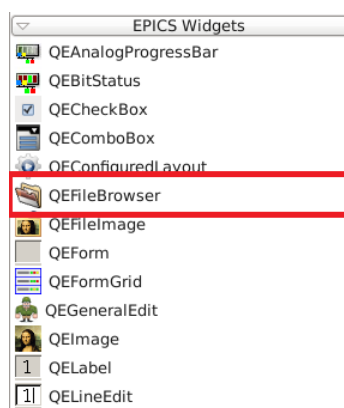


Figure 7 QEFileBrowser within Qt Designer

The QEFileBrowser has the following properties (that can be controlled by the user):

- **directoryPath**  
Default directory where to browse files when QEFileBrowser is launched for the first time
- **showDirectoryPath**  
Show/hide directory path line edit where the user can specify the directory to browse files
- **showDirectoryBrowser**  
Show/hide button to open the dialog window to browse for directories and files
- **showRefresh**  
Show/hide button to refresh the table containing the list of files being browsed
- **showTable**  
show/hide table containing the list of files being browsed
- **showColumnTime**  
show/hide column containing the time of creation of files
- **showColumnSize**  
Show/hide column containing the size (in bytes) of files
- **showColumnFilename**

## QE Framework – Widget Specifications

- Show/hide column containing the name of files
- **showFileExtension**  
show/hide the extension of files
- **fileDialogDirectoriesOnly**  
Enable/disable the browsing of directories-only when opening the dialog window
- **fileFilter**  
Specify which files to browse. To specify more than one filter, please separate them with a ";".  
Example: \*.py;\*.ui (this will only display files with an extension .py or .ui).
- **optionsLayout**  
Change the order of the widgets. Valid orders are: TOP, BOTTOM, LEFT and RIGHT.

The following figure illustrates the QEFileBrowser widget in production:

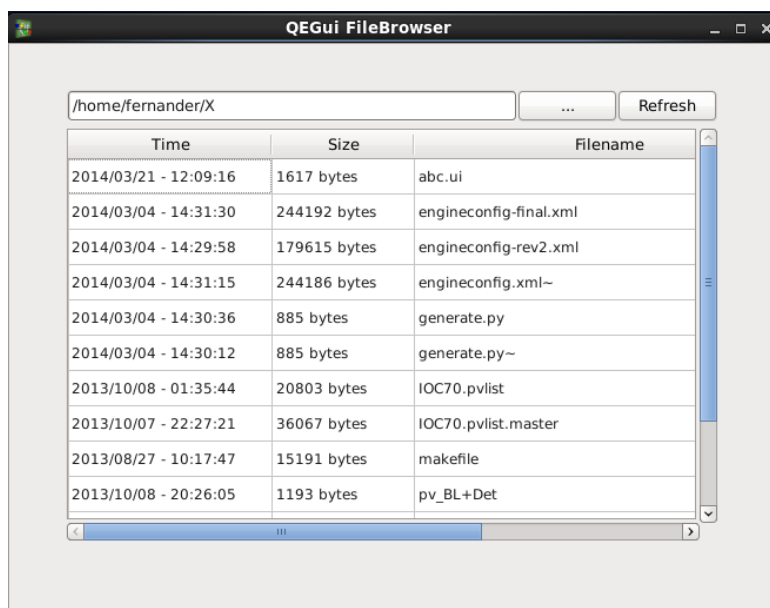


Figure 8 QEFileBrowser displaying existing files in directory "/home/fernander/X"

### QEFileImage

The QEFileImage displays an image file where the name of the file to display has been provided through a variable. The file type can be any type that can be loaded into a QPixmap – for example, .png, .jpg, .bmp, .tiff, etc.

If the file referenced changes it is updated in the widget.

This widget can be used in several ways:

## QE Framework – Widget Specifications



- Displaying an updating image. This is useful where a third party system is generating an image file that is not integrated into EPICS.
- Displaying the last image captured in a scan where a variable is set to point to the last most recent image capture during a scan. Note, the image path must be valid on the machine where the widget is used.
- Selecting a graphic for display. A calculation can be used to select a file name based on a value. Note, using the widget like this embeds GUI functionality in the control system which is generally not good practice.
- Previewing image file selected using the QFileDialog widget. This may be performed in a couple of ways:
  - Linking the QFileDialog widget's 'selected' signal to the QFileDialog 'setImageFileName' slot directly on the GUI.
  - Where the QFileDialog widget's output is written to a variable, using that variable as the 'variable' property of the QFileDialog widget.

Note, if an image is available directly through channel access of an mpeg stream the QImage widget can be used to display the image.

Figure 9 (page 17) shows examples of QFileDialog used to display an image as specified by an EPICS variable, and to preview an image selected from a QFileDialog widget. The QFileDialog preview uses the 'selected' signal from the QFileDialog widget connected to the 'setImageFileName' slot of the QFileDialog widget.

The QFileDialog widget has the following unique properties:

### **variableName**

The variable providing the file name text. The file will be searched for using standard rules for locating files described in section '**Error! Reference source not found.**' (page**Error! Bookmark not defined.**). The variable name can also be set directly using the 'setImageFileName' slot.



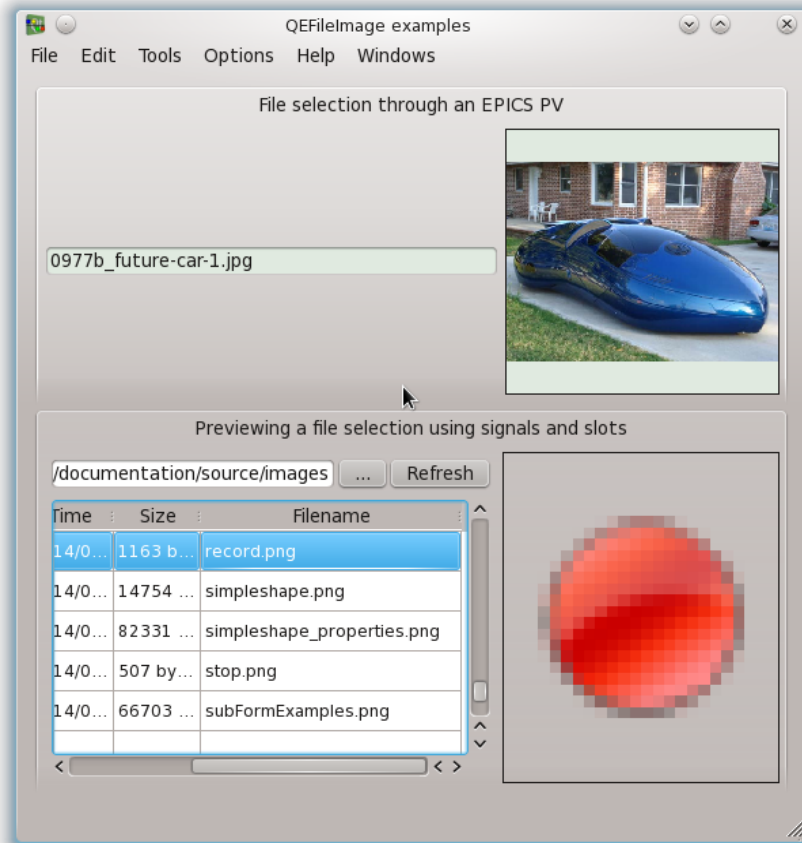


Figure 9 QEFileImage widget taking file information from variable and from a signal

## 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 constant rules for locating the file common to all QE widgets that access files. Refer to **Error! Reference source not found.** (page **Error! Bookmark not defined.**) 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.

## QE Framework – Widget Specifications



- 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. The 'variableSubstitutions' property is used to define macro substitutions unique to the sub form. These macro substitutions take precedence over any other macro substitutions current when the QEForm is created.

QEForms help manage messages emitted by QE widgets. Messages can be filtered and displayed based on the QEForm they reside in. Refer to **Error! Reference source not found.** (page **Error! Bookmark not defined.**) 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 **Error! Reference source not found.** (page **Error! Bookmark not defined.**) for details on how this file is located.
- handleGuiLaunchRequests  
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

## QE Framework – Widget Specifications

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 10, 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 QEForm, rather than the other way around.

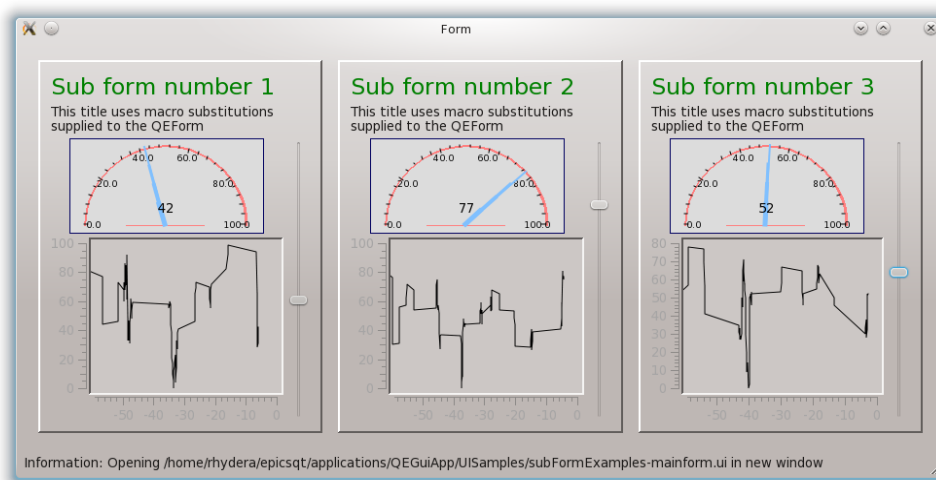


Figure 10 QEForm examples

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

The QEFormGrid widget directly inherits from QFrame. This widget provides a grid (or matrix) of QEForm sub-forms, each of which loads the ui file defined by the uiFile property.

The widget also provides a variableSubstitutions property that may be used to provide (default) values for any macro used within the uiFile property.

## QE Framework – Widget Specifications



Each sub-form may be parameterised by six priority macros definitions. The actual value associated with these macros is determined from the row and column position within the grid and property values. The formal macros names are themselves defined by three macro prefix properties.

### Properties

The following properties are specified to the QEFormGrid widget.

- a) `uiFile`: this defines the ui file to be loaded into each sub-form;
- b) `variableSubstitutions`: this provides the default substitution values for any macros used within `uiFile`;
- c) `number`: this defines the number of elements in the grid. The minimum, maximum and default values are 1, 210, and 4 respectively;
- d) `columns`: this defines the number of columns in the grid. The minimum, maximum and default values are 1, 42, and 1 respectively. The actual number of columns,  $N_c$ , will not exceed the number of elements in the grid.

The number of rows,  $N_r$ , is calculated such that  $N_r$  is the minimum value that satisfies:

$$N_r * N_c \geq \text{number}.$$

When the grid is incomplete, i.e.  $\text{number} < N_r * N_c$ , empty slots appear at the bottom left of the grid;

- e) `gridOrder`: this property defines the grid's slot layout scheme. This property is an enumeration with two values, viz: `RowMajor` (default) and `ColMajor`. `RowMajor` means slot numbers first increase left to right rows, and then by column, e.g.:

```
1  2  3  4
5  6  7  8
9 10
```

`ColMajor` means slot numbers first increase top to bottom in columns, and then by row, e.g.:

```
1  4  7 10
2  5  8
3  6  9
```

- f) `margin`: the grid of QEForm objects is layed out using a `QGridLayout` object. This property is effectively the grid layout's margin property;
- g) `spacing`: This property is effectively the grid layout's spacing property;
- h) `slotMacroPrefix`: this defines the prefix for the two slot related macros. The default prefix is `SLOT` and the default macro names are thus `SLOT` and `SLOTNAME`. The slot values are numeric and start from the `slotNumberOffset` value in the order defined by `gridOrder`. The slot-name values are defined by the `slotStrings` property;
- i) `slotNumberOffset`: This defines the first slot number. The default value is 1. Whereas typically this will be 0 or 1, any integer value is allowed. For accessing the `slotStrings` list to determine the slotname value, the offset is always zero;

## QE Framework – Widget Specifications



- j) slotNumberWidth: This defines the minimum image width. The minimum, maximum and default values are 1, 6 and 2 respectively. Where necessary the slot value is zero left padded to achieve the required width;
- k) slotStrings: This property is a QStringList and holds the set of string values used to populate values for the slotname macro;
- l) rowMacroPrefix: this defines the prefix for the two row related macros. The default prefix is ROW and the default macro names are thus ROW and ROWNAME. The row values are numeric and start from the rowNumberOffset value. The rowname values are defined by the rowStrings property;
- m) rowNumberOffset: This defines the first row number. The default value is 1;
- n) rowNumberWidth: This defines the minimum image width. The default value is 2;
- o) rowStrings: This property is a QStringList and holds the set of string values used to populate values for the rowname macro;
- p) colMacroPrefix: this defines the prefix for the two column related macros. The default prefix is COL and the default macro names are thus COL and COLNAME. The col values are numeric and start from the colNumberOffset value. The column name values are defined by the colStrings property;
- q) colNumberOffset: This defines the first column number. The default value is 1;
- r) colNumberWidth: This defines the minimum image width. The default value is 2; and
- s) colStrings: This property is a QStringList and holds the set of string values used to populate values for the column name macro.

### Nested QEFormGrid

The loaded ui File may itself contain a QEFormGrid widget that in turn loads further ui files.

Note: Care should be taken to avoid recursively loading the same form either directly or indirectly. There is currently no check to prevent this and this will eventually lead to a segmentation fault.

### Examples

Figure 11 below shows an example of a nested set of QEFormGrids in designer.

The ccg\_unit.ui form (upper left in the figure) is a basic form with one QESubstitutedLabel (labelText is \$(CCG)), and two QELabels for displaying cold cathode gauge PVs. The associated variable properties are specified as SR\$(SECTOR)CCG\$(CCG):STATUS and SR\$(SECTOR)CCG\$(CCG):PRESSURE\_MONITOR.

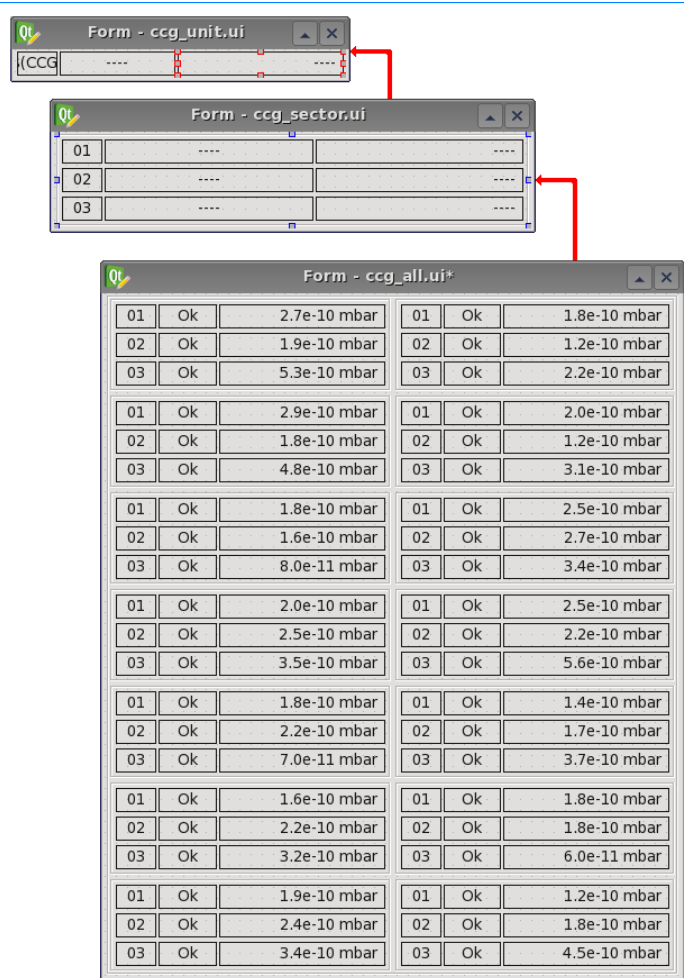
The ccg\_sector.ui file (2<sup>nd</sup> form from top) contains one QEFormGrid object. The relevant properties are shown on the upper right. Note worthy is that the uiFile property specifies ccg\_unit.ui form described in the previous paragraph, and that the rowMacroPrefix property is set to CCG, and thus the CCG macro is defined as "01", "02", and "03" for each row respectively.

The ccg\_all.ui file (lower left in the figure) contains one QEFormGrid object. The relevant properties are shown on the lower right. Note the uiFile property specifies ccg\_sector.ui form described in the previous

## QE Framework – Widget Specifications

paragraph, and that the slotMacroPrefix property is set to SECTOR, and thus the SECTOR macro is defined as "01", "02", ... "14" for each slot respectively.

Figure 12 below shows the form as presented by qegui.



Property	Value
<b>uiFile</b>	/home/starritt/ui/ccg_unit.ui
<b>variableSubstitutions</b>	
<b>number</b>	3
columns	1
gridOrder	RowMajor
margin	2
spacing	0
<b>slotMacroPrefix</b>	SLOT
slotNumberOffset	1
slotNumberWidth	2
slotStrings	
<b>rowMacroPrefix</b>	CCG
rowNumberOffset	1
rowNumberWidth	2
rowStrings	
<b>colMacroPrefix</b>	COL
colNumberOffset	1
colNumberWidth	2
colStrings	

<b>uiFile</b>	/home/starritt/ui/ccg_sector.ui
<b>variableSubstitutions</b>	
<b>number</b>	14
<b>columns</b>	2
<b>gridOrder</b>	ColMajor
<b>margin</b>	0
<b>spacing</b>	0
<b>slotMacroPrefix</b>	SECTOR
slotNumberOffset	1
slotNumberWidth	2
slotStrings	
<b>rowMacroPrefix</b>	ROW
rowNumberOffset	1
rowNumberWidth	2
rowStrings	
<b>colMacroPrefix</b>	COL
colNumberOffset	1
colNumberWidth	2
colStrings	

Figure 11 QEFormGrid in designer

File Edit Tools Options Help Windows			
01	Ok	2.7e-10 mbar	
02	Ok	1.9e-10 mbar	
03	Ok	5.3e-10 mbar	
01	Ok	2.9e-10 mbar	
02	Ok	1.8e-10 mbar	
03	Ok	4.8e-10 mbar	
01	Ok	1.8e-10 mbar	
02	Ok	1.6e-10 mbar	
03	Ok	8.0e-11 mbar	
01	Ok	2.0e-10 mbar	
02	Ok	2.5e-10 mbar	
03	Ok	3.5e-10 mbar	
01	Ok	1.8e-10 mbar	
02	Ok	2.2e-10 mbar	
03	Ok	7.0e-11 mbar	
01	Ok	1.6e-10 mbar	
02	Ok	2.2e-10 mbar	
03	Ok	3.2e-10 mbar	
01	Ok	1.9e-10 mbar	
02	Ok	2.4e-10 mbar	
03	Ok	3.4e-10 mbar	
01	Ok	1.8e-10 mbar	
02	Ok	1.2e-10 mbar	
03	Ok	2.2e-10 mbar	
01	Ok	2.0e-10 mbar	
02	Ok	1.2e-10 mbar	
03	Ok	3.1e-10 mbar	
01	Ok	2.5e-10 mbar	
02	Ok	2.7e-10 mbar	
03	Ok	3.4e-10 mbar	
01	Ok	2.5e-10 mbar	
02	Ok	2.2e-10 mbar	
03	Ok	5.6e-10 mbar	
01	Ok	1.4e-10 mbar	
02	Ok	1.7e-10 mbar	
03	Ok	3.7e-10 mbar	
01	Ok	1.8e-10 mbar	
02	Ok	1.8e-10 mbar	
03	Ok	6.0e-11 mbar	
01	Ok	1.2e-10 mbar	
02	Ok	1.8e-10 mbar	
03	Ok	4.5e-10 mbar	

Figure 12 QEFormGrid example

### QEFrame and QEPvFrame

The QEFrame widget provides a minimalist extension to the QFrame widget. Like the QEGroupBox widget it provides user level enabled and user level visibility control to the frame but more significantly to all the widgets enclosed within the QEFrame container as well.

## QE Framework – Widget Specifications

A QEFrame can also have up to 8 background images, set by properties pixmap0, pixmap1, ..., pixmap7. The pixmap property is deprecated and not available in designer, and is an alias for pixmap0 .is deprecated. The image, if any, associated with pixmap0 is used.

Two properties 'pixmap0' and 'scaledContents' allow an image to be specified and scaled if required in exactly the same way these properties work in a QLabel widget. A background image is particularly useful in GUIs where components are placed over a schematic. If the 'scaledContents' property is set, the pixmap will be scaled to fill the QEFrame. If the frame's contents relates to a position on the background image, the contents should be managed by a layout in such a way that the components remain positioned over the appropriate point in the background image as the frame is resized. Alternatively, the frame may be set to a fixed size.

The QEPvFrame class inherited directly from QEFrame and allows the specification of a process variable using the 'variable' and 'arrayIndex' properties. The variable is subscribed for as an integer, and provided the value is in the range 0 to 7 is used to select the appropriate pixmap used as background image. The value is not in this range, no background pixmap image is used.

### QEGeneralEdit

The QEGeneralEdit widget is a general purpose scalar PV edit widget. Whilst this widget may be included in any form, it is primarily intended for use in one of the qeui's built in forms. When the user level is engineer, the standard PV context menu is extended to include "Edit PV", as per Figure 13below.

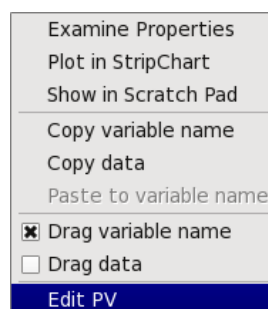


Figure 13 Modified Context Menu - Engineer User Level

When selected, this causes the general edit form window to be launched, which contains a single QEGeneralEdit widget. The QEGeneralEdit comprises the following widgets:

- a) QLabel;
- b) QELabel;
- c) QNumericEdit and QENumericEdit;
- d) QERadioGroup; and
- e) QELineEdit.



## QE Framework – Widget Specifications

The text of widget (a) is set to the name of the selected PV and widget (b)'s variable name is set to the name of the selected PV, thus displays the selected PVs current value. Depending of the data type (numeric, enumeration or string) the visibility widget (c), widget (d) or widget (e) is set true respectively, whilst the visibility of the other two widgets is set false. As appropriate, the variable name of item (c), item (d) or item (e) is set to the name of the selected PV. Figure 14, Figure 15, and Figure 16 show examples of editing a numeric, enumeration and string PV respectively.

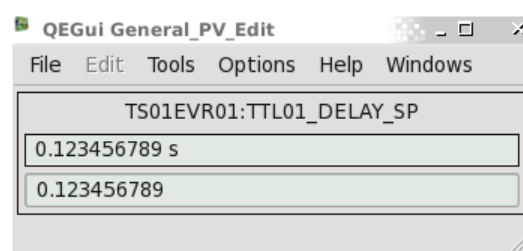


Figure 14 QEGeneralEdit example for a numeric for PV

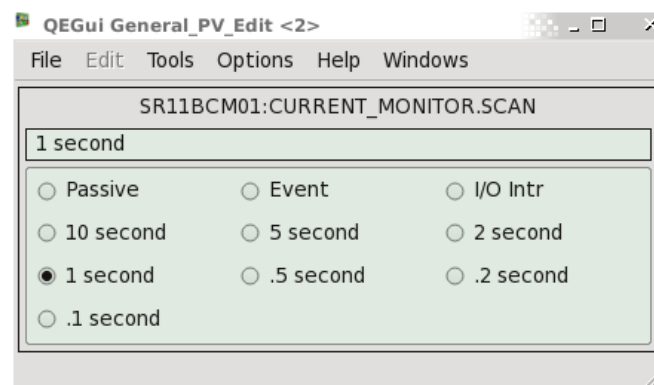


Figure 15 QEGeneralEdit example for an enumeration PV

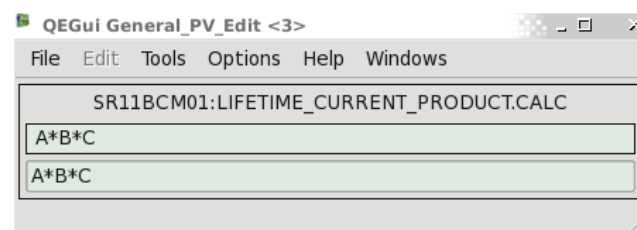


Figure 16 QEGeneralEdit example for a string PV

## QE Framework – Widget Specifications

### QEGroupBox

The QEGroupBox widget provide a minimalist extension to the QGroupBox widget. Like the QEFrame widget, it provides user level enabled and user level visibility control to the group box but more significantly to all the widgets enclosed within the QEGroupBox container as well.

The group box title, normally set through the QGroupBox title property, can be set through the QEGroupBox substitutedTitle and textSubstitutions properties. This is useful when the QEGroupBox is used as a sub-form, or within a sub form. An example of this is shown in Figure 17.

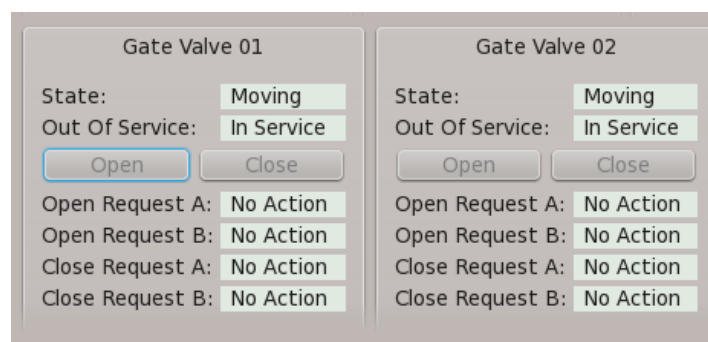


Figure 17 QEGroupBox sub forms with macro substitutions applied to the titles

### QEImage

The QEImage widget is now described in its own document.

### 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 QE\_QEGuiAndUserInterfaceDesign.docx for details about the standard text formatting. In particular, note how local enumerations can include style hints for QELabel widgets.

If the data that is being presented in a QELabel is array data, the data is limited to 10000 elements. This arbitrary limit allows for arrays to be presented as strings but avoids processing overhead in the case of very large arrays, such as high resolution images, being inappropriately used as the data source for a QELabel.

## QE Framework – Widget Specifications

If a QLabel is being used as a source of data for a QELink widget and the label text does not need to be visible, the 'visible' property can be set false so the label will not be visible. Note, it will remain visible when viewed within Qt Designer or Qt Designer's preview system.

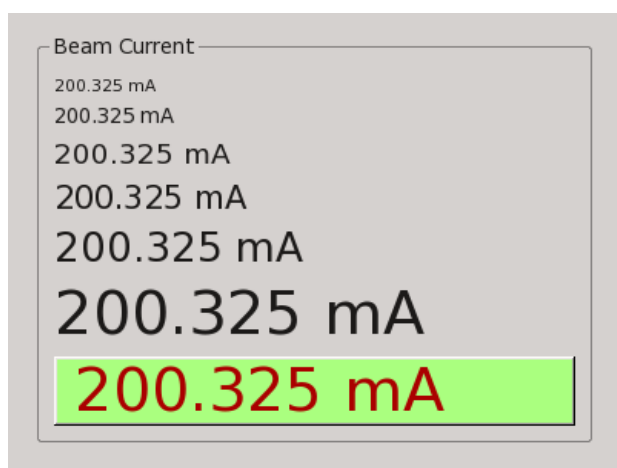


Figure 18 QLabel examples with variations to QLabel properties



Figure 19 QLabel used to display a pump failure

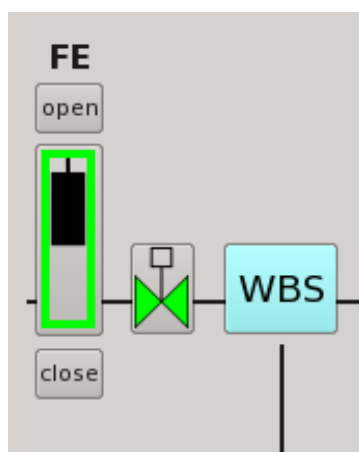


Figure 20 QELabels used icons to represent states

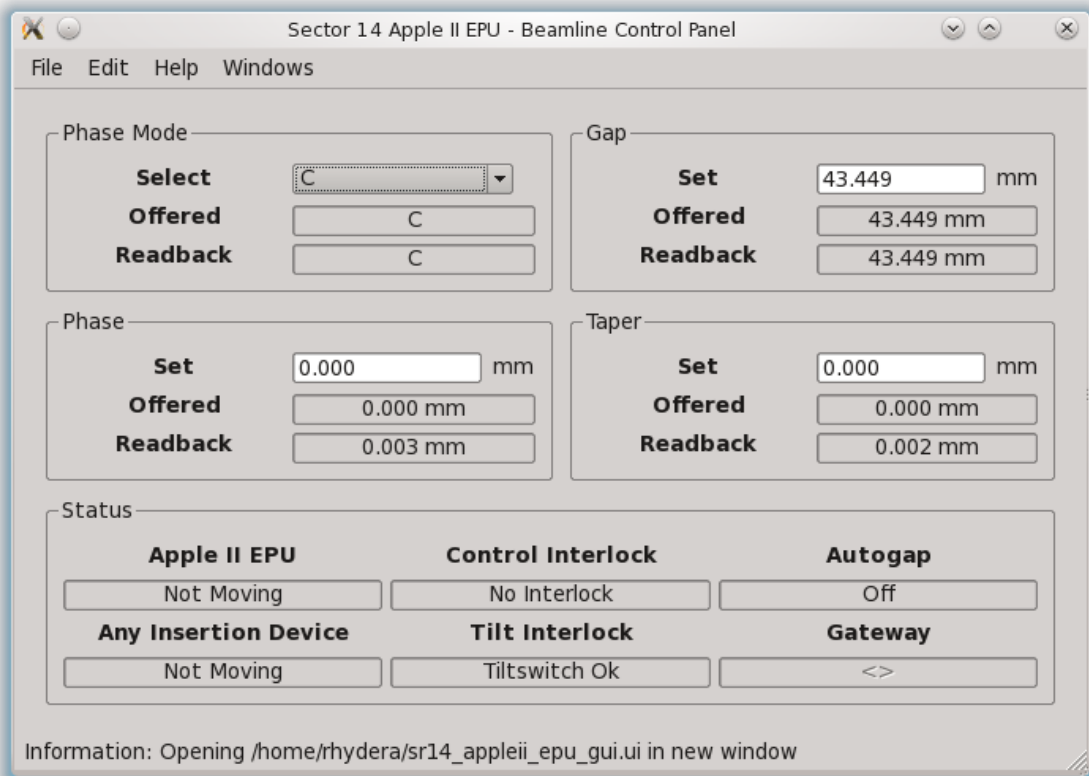


Figure 21 GUI using mostly QLabels to represent numeric and textual data

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

- **Invalid** (not connected) – The QLabel is displayed not-enabled, or ‘greyed out’.
- **In alarm condition** – The QLabel 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 QLabel 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. When the display of the alarm state is not selected, the default style is a slightly lighter than background colour.

### QEDescriptionLabel

The QEDescriptionLabel inherits directly from QELabel. It provides no additional properties or any additional functionality. However it does have different default values for some of the properties that make its appearance more like a QLabel:

- a) The default value for displayAlarmStateOption is "Never" (as opposed to "Always");
- b) The defaultStyle property is clear (as opposed to lighter than background);
- c) The font size is 8 (as opposed to 9); and
- d) The default indent value is -1 (as opposed to 6).

This widget is intended for textual labels, the content being provided by the .DESC field of a record or any other string PV.

### QELCDNumber

The QELCDNumber widget provides an EPICS aware monitor widget based the standard QLCDNumber widget. Specifically, the QELCDNumber widget inherits directly from QEFrame (which provides many of the standard EPICS Qt related properties) and contains a single internal QLCDNumber widget to provide the LCD display functionality. Due to the nature of the QLCDNumber widget, the QELCDNumber widget is only suitable for numeric PV values, and there is no option to display the PV's engineering units.

See Figure 22 QELCDNumber properties below.

qelcdnumber : QELCDNumber	
Property	Value
<b>QObject</b>	
<b>objectName</b>	qelcdnumber
<b>QWidget</b>	
<b>QFrame</b>	
<b>QEFrame</b>	
variableAsToolTip	<input checked="" type="checkbox"/>
allowDrop	<input type="checkbox"/>
visible	<input checked="" type="checkbox"/>
messageSourceId	0
▶ defaultStyle	
▶ userLevelUserStyle	
▶ userLevelScientistStyle	
▶ userLevelEngineerStyle	
userLevelVisibility	User
userLevelEnabled	User
displayAlarmStateOption	Always
scaledContents	<input checked="" type="checkbox"/>
pixmap0	
pixmap1	
pixmap2	
pixmap3	
pixmap4	
pixmap5	
pixmap6	
pixmap7	
<b>QELCDNumber</b>	
▶ <b>variable</b>	SR11BCM01:CURRENT_MONITOR
▶ variableSubstitutions	
elementsRequired	0
arrayIndex	0
<b>smallDecimalPoint</b>	<input checked="" type="checkbox"/>
segmentStyle	Filled
precision	4
useDbPrecision	<input checked="" type="checkbox"/>
notation	Fixed

Figure 22 QELCDNumber properties

### QELineEdit

The QELineEdit widget is now described separately in its own document.

### 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. This may be overridden by setting the runVisible property to True.

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 23, 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 24 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 made 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.

## QE Framework – Widget Specifications

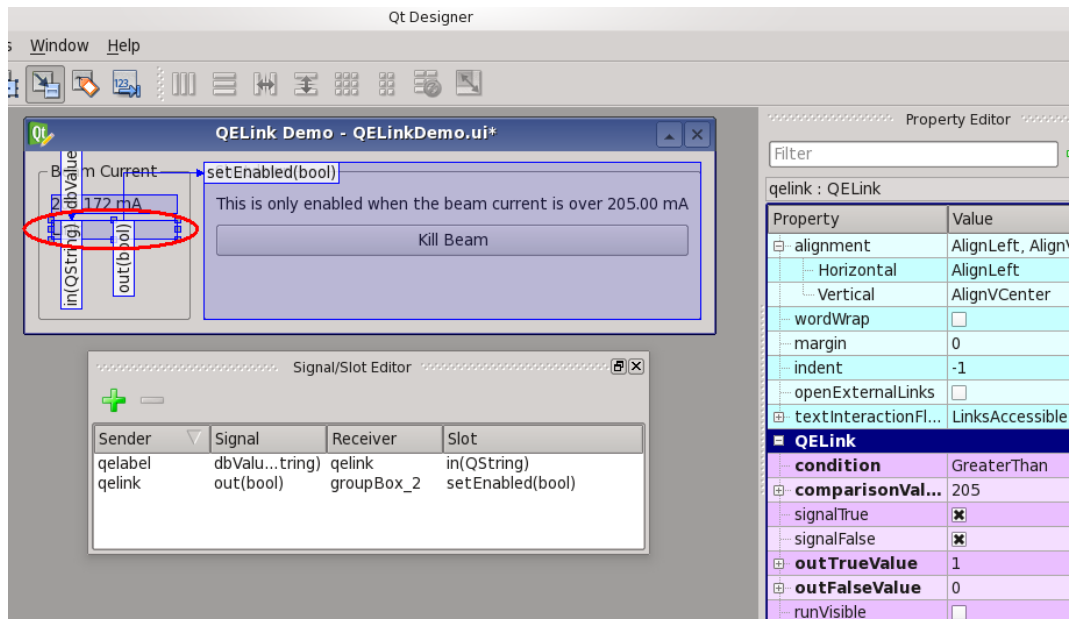


Figure 23 QELink being configured

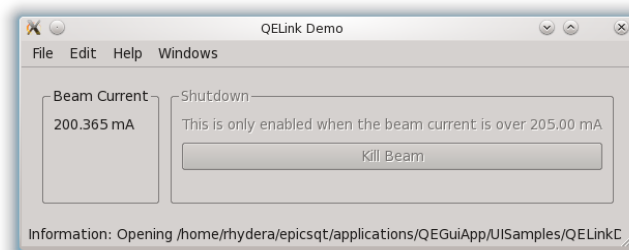


Figure 24 QELink in use

## QECalcout

The QECalcout widget may be used instead of and/or in conjunction with the QELink widget. This widget can accept up to 12 value (double or int) signals from other widgets (QE and none-QE widget) and performs a calculations to both generate the output signal and to determine if the output signal should be sent. The widget name was chosen because of the functional similarity to the calcout record. Likewise, the property names, where applicable, were chosen to match the calcout record.

The QECalcout widget can be made visible at all times by setting the 'visible' property. The widget is based on a QLabel and the displayed text is the same as the out (QString) signal.



## QE Framework – Widget Specifications

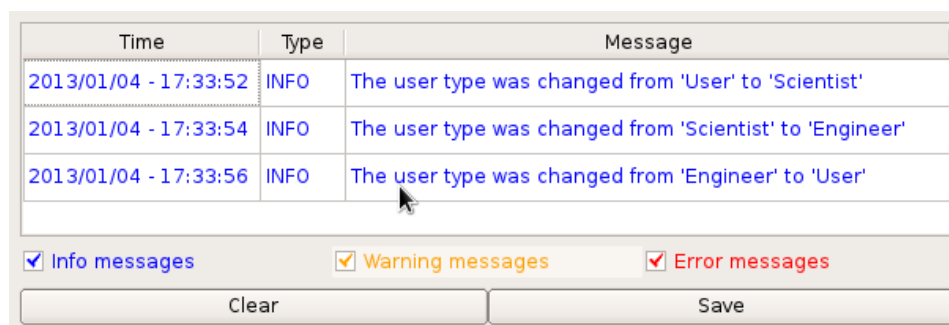
As with QELink, this widget mitigates the need to create control PVs with the sole purpose to support the GUI. Conversely, while it is tempting to use this widget to quickly and easily do GUI-side calculations, you should ask yourself whether this should really be done in an IOC? Such a PV can then be archived, alarmed, plotted and be available to the rest of the control system in general.

### 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 '**Error! Reference source not found.**' (page **Error! Bookmark not defined.**) 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 25 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.

## QE Framework – Widget Specifications



- 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'.

User levels affect the behaviour of the QEGui application and most QE widgets.

## QE Framework – Widget Specifications

The QEGui application uses the current user level to control if menu items and tool bar buttons are enabled or visible. Refer to '**Error! Reference source not found.**' (page **Error! Bookmark not defined.**) for details.

MostQE 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 '**Error! Reference source not found.**' (page **Error! Bookmark not defined.**) 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 26 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 26. 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.

## QE Framework – Widget Specifications

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 27 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.)

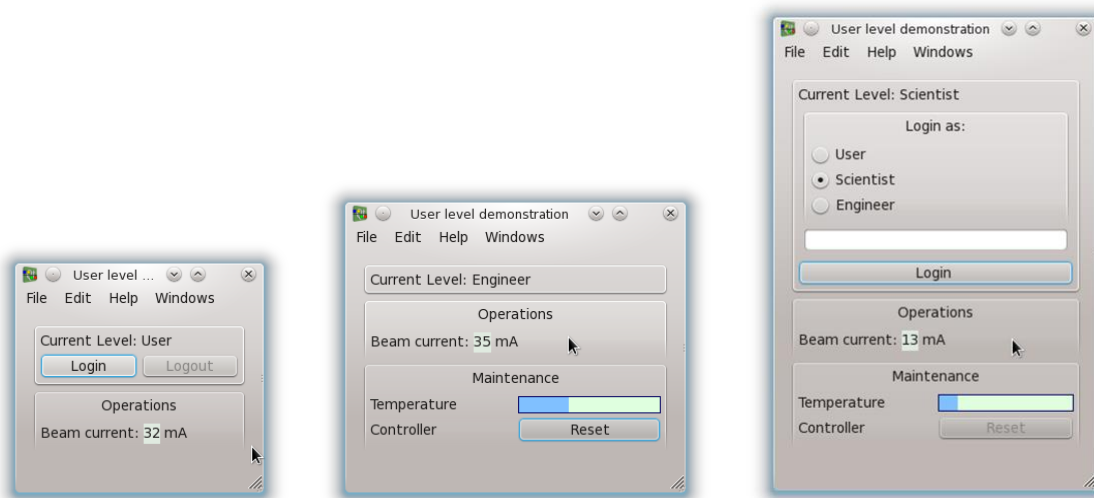


Figure 27 QELogin widgets in various modes and user levels

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. 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.

## QE Framework – Widget Specifications

### QNumericEdit and QNumericEdit

The QNumericEdit and QNumericEdit widgets are now described in their own document.

### QEPeiodic

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

Alternatively, the QEPeiodic widget can be used independently of EPICS variables, using signals and slots to set an element, or to obtain a user selection of an element. Note, most of the following description explains the QEPeiodic widget's interaction when EPICS variables are defined.

For example, a two axis reference foil stage may be controlled with a QEPeiodic 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 QEPeiodic 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 QEPeiodic widget a variable holding ionization energy may be set directly by a user selecting an element from a dialog containing a periodic table.

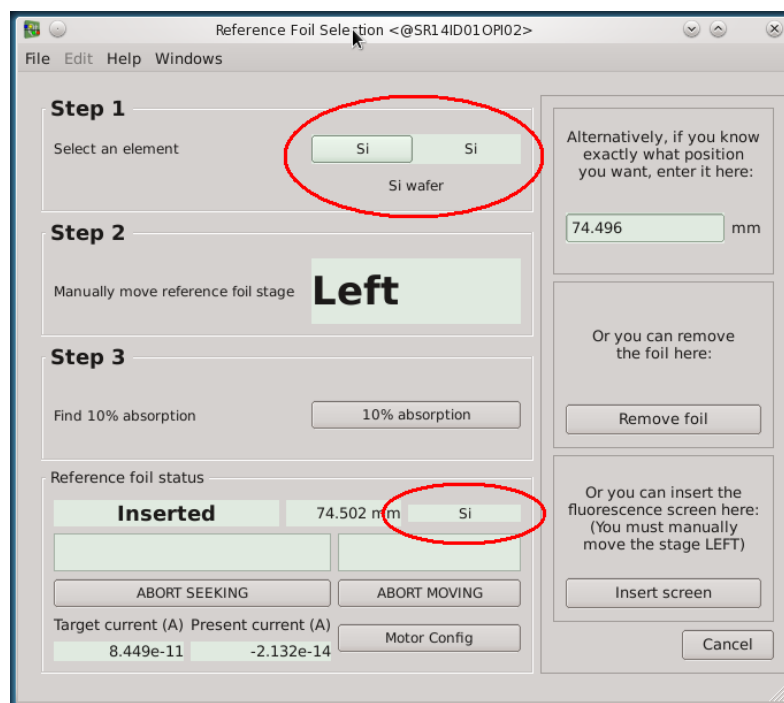


Figure 28 QEPeiodic used for both read-back and control by element.

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

## QE Framework – Widget Specifications

- 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 29. 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 30. The two properties defining the one or two variables written to are:

- writeButtonVariable1
- writeButtonVariable2

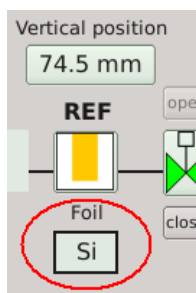


Figure 29 QEPeriodic widget used to represent variables by element in a read only mode.

## QE Framework – Widget Specifications

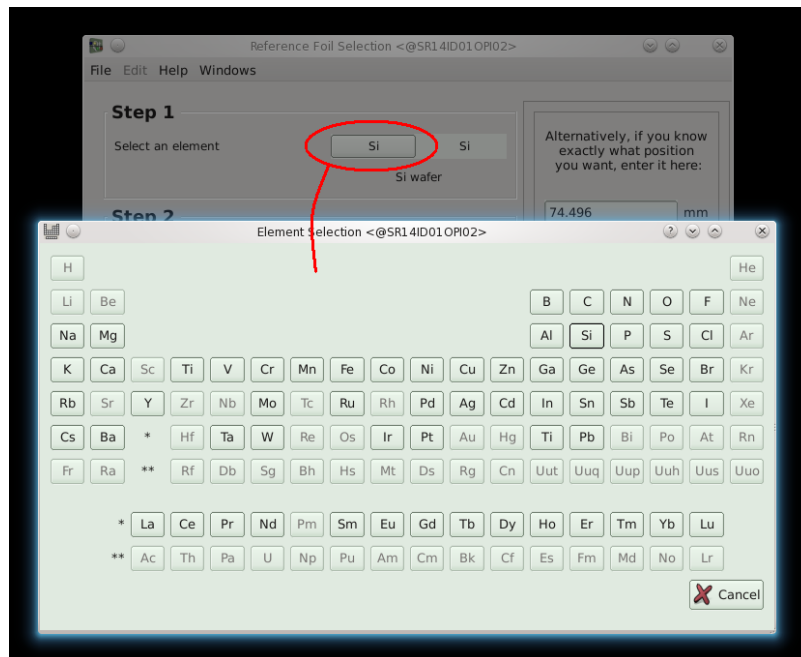


Figure 30 QEPeriodic widget used to manipulate variables by element selection

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

## QE Framework – Widget Specifications



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 can be defined at design time and stored within the QEPeriodic widget, or in a file referenced by the widget. The three relevant properties are:

- **userInfo** An XML string defining these values.
- **userInfoFile** A file name of a file containing XML defining these values.
- **userInfoSourceOption** If 'userInfoSourceText' then the 'userInfo' property is used to define the values. If 'userInfoSourceFile' then the XML in file specified is used to specify the values.

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, or the contents of the file specified by the 'userInfoFile' property, may be edited directly, it is easier to use the User Info editor shown in Figure 31. This editor may be invoked by right clicking on the QEPeriodic widget in 'Designer' and selecting 'Edit User Info...'. Figure 32 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). When the editor is closed, the 'userInfo' property is updated if the 'userInfoSourceOption' property is set to 'userInfoSourceText' or the contents of the file specified by the 'userInfoFile' property is updated if the 'userInfoSourceOption' property is set to 'userInfoSourceFile'. Note, the file will not be created if it does not exist.

As well as defining the values associated with the element, some text may also be defined which will be emitted by the dbElementChanged and dbAtomicNumberChanged signals when the read-back label updates. This may be, for example, connected to a standard QLabel setText slot as shown in Figure 28.



## QE Framework – Widget Specifications

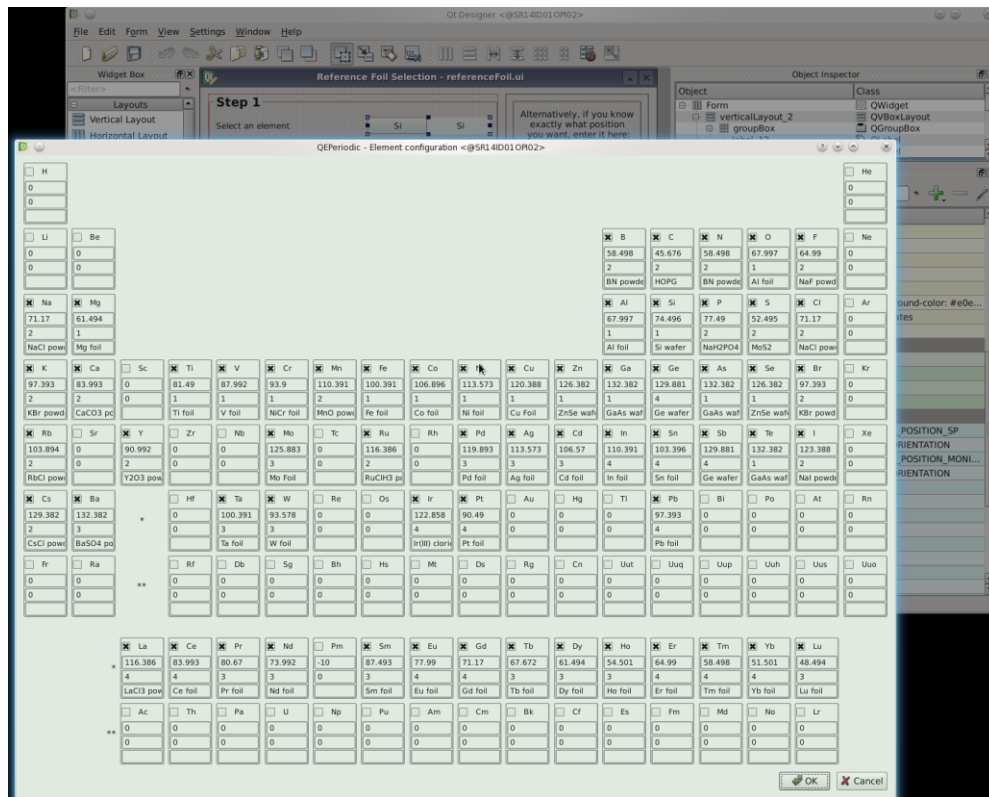


Figure 31 Editing the QEperiodic userInfo property - the relationship between each element and variable values

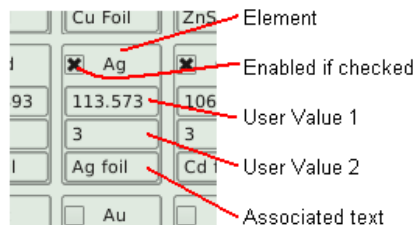


Figure 32 Editing the QEperiodic userInfo property - what is associated with each element

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
- variableTolerance1

The following signals and slots allow for use without EPICS variables defined. (Note, the signals and slots still function even if EPICS variables defined)

## QE Framework – Widget Specifications



- Slot: `setElement( QString symbol )`  
`setAtomicNumber( const int atomicNumber )`
- Signal: `userElementChanged( const QString& symbol )`  
`userAtomicNumberChanged( const int atomicNumber )`

Note, the `userElementChanged()` and signals `userAtomicNumberChanged()` will be emitted as user selects an element. This differs from the `dbElementChanged()` and `dbAtomicNumberChanged()` signals which is emitted when the current is set due to an EPICS value change.

A colourised property (Boolean, default false) now allows the element category to be indicated by a pale, not too intrusive colour. Note, this property affects the run-time dialog only. The design-time configuration dialog is always colourised.

### QEPlot

The QEPlot widget is now described in its own document.

### QEPlotter

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.

## QE Framework – Widget Specifications

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 is 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

## QE Framework – Widget Specifications



arguments represents the value provided by the 2<sup>nd</sup> Y variable, and C the value provided by the 3<sup>rd</sup> 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 QECheckBox). 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

## QE Framework – Widget Specifications



- 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:

Data formatting properties are used for both reading and writing data. If enumeration values (local, or from the database) are involved in the formatting specified, the values written must be compatible with this formatting.

Before considering how QE buttons use enumerated values, if you simply want to write 0 or a 1 to a variable, set the 'format' property to 'Integer'. The defaults for the properties defining the values to write ('clickText', 'clickCheckedText', 'presstext', 'releaseText') are all integers (0 or 1). With the 'format' property to 'Integer', these values will all just work as they are.

The QEPushButton widget can display variable data in the button label and, like many QE widgets, standard formatting will be applied to the variable data using properties such as the 'format', 'precision', or 'localEnumeration' properties (See '**Error! Reference source not found.**' - page **Error! Bookmark not defined.** for full details on formatting for presentation). While these formatting properties are only used for variable presentation in the QEPushButton, they are used by all QE buttons when writing data (as do most QE widgets that write data). These properties will determine how the text written will be formatted. If the 'format' property is set to 'Default' and the database provides a set of enumeration values, or the 'format' property is set to 'LocalEnumeration', then the text written must match the enumerations.

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', 'clickText', and 'clickCheckedText' 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.
```

## QE Framework – Widget Specifications



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.

### **Signal on program completion:**

A 'programComplete' signal is emitted when a program initiated by a QE button has completed.

For example, the standard Qt 'clicked' signal can disable controls that should not be used while a program is running. The 'programComplete' signal can re-enable the controls.

### **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.

## QE Framework – Widget Specifications



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'



## QE Framework – Widget Specifications



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 defined, 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 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)
  - TextAndState     Data updates will update the button text and state
  - IconAndState     Data updates will update the button icon and state
  - TextIconAndState     Data updates will update the button text, icon and state
- **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.

## QE Framework – Widget Specifications



### 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 '**Error! Reference source not found.**' (page **Error! Bookmark not defined.**) for more details on how to view content logged to the QE message system.

A 'programComplete' signal is emitted by QE buttons when the system command completes.

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 ('programStartupOption' = 'Terminal') where a command interpreter is started automatically.

## QE Framework – Widget Specifications

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:          firefoxwww.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
```

## QE Framework – Widget Specifications



- 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.

QEWidgets use a common set of rules for locating a file. Refer to **Error! Reference source not found.** (page **Error! Bookmark not defined.**) 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             |
| ○ DockTopTabbed    | Open new GUI in a tabbed top dock        |
| ○ DockBottomTabbed | Open new GUI in a tabbed bottom dock     |
| ○ DockLeftTabbed   | Open new GUI in a tabbed left dock       |
| ○ DockRightTabbed  | Open new GUI in a tabbed 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 substitutions 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 substitutions 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.

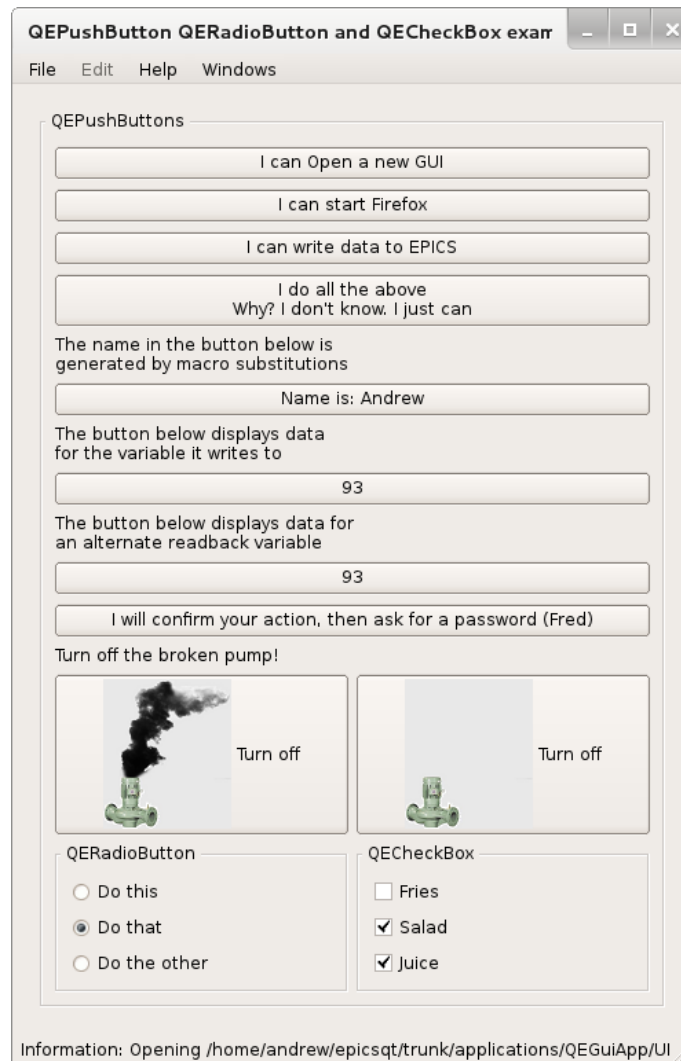


Figure 33 QEPushButton, QERadioButton and QECheckBox examples

### Applying a style based on how the button is used

A dynamic property “StyleOption” is defined with a value of “PV”, “Program”, “UI” and “”. This property can be used for stylesheet configuration to configure a different style to its button. For example, if the widget is configured to write/read PV, the value will be “PV”; if it is configured to run a program, the value will be “Program”; if it is configured to load a ui form, the value will be “UI”; otherwise, the value will be “” as a default. To avoid a possible conflict, the priority has been set in the order of writing/reading PV, running a program and loading a ui file.

Stylesheet example:

```
QEPushButton[StyleOption="PV"] {color:purple}
QEPushButton[StyleOption="Program"] {color:red}
```

## QE Framework – Widget Specifications

```
QEPushButton[StyleOption="UI"]      {color:green}  
QEPushButton                        {color:blue}  
QEPushButton:!enabled               {color:grey}
```

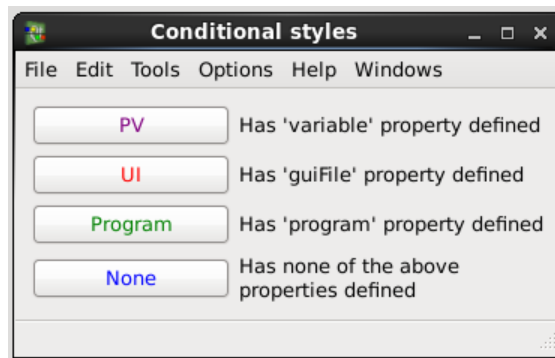


Figure 34 Conditional Style

## QEMenuButton

### General Description

The QEMenuButton widget is a QPushButton with an associated menu. Each menu entry provides a sub-set of the functionality provided by an individual QEPushButton, i.e. each menu item provides the following non-exclusive functions:

- a) Write to variable;
- b) Issue a command to the operating system; and
- c) Open a new GUI form.

The functionality provided is a sub-set as this widget does:

- a) **not** read and present variable values;
- b) **not** emit dbChanged like signals; and
- c) **only** provides a 'clickText' value only (as opposed to pressed, released and checked values).

The QEMenuButton may be configured from within designed by right-clicking on the widget and selecting the "Edit Menu Info..." option which launches the Menu Button Setup dialog (see example in Figure 35 Menu Button Setup dialog below).

## QE Framework – Widget Specifications



The left hand side of the set up dialog provides a menu tree, while the right hand side the set of "properties" associated with the selected menu item. The context menu over the tree provides three options:

- a) Add Menu Item – creates a menu action item
- b) Add Sub menu – create a sub menu item holder (like the shutter node in Figure 35 below); and
- c) Delete menu Item – delete the menu item and any associated sub menu items.

Menu items may also be dragged and dropped *within* the menu tree in order to allow the menu tree to be arranged. The default allocated menu names are of the form, e.g. X00011, and should be renamed.

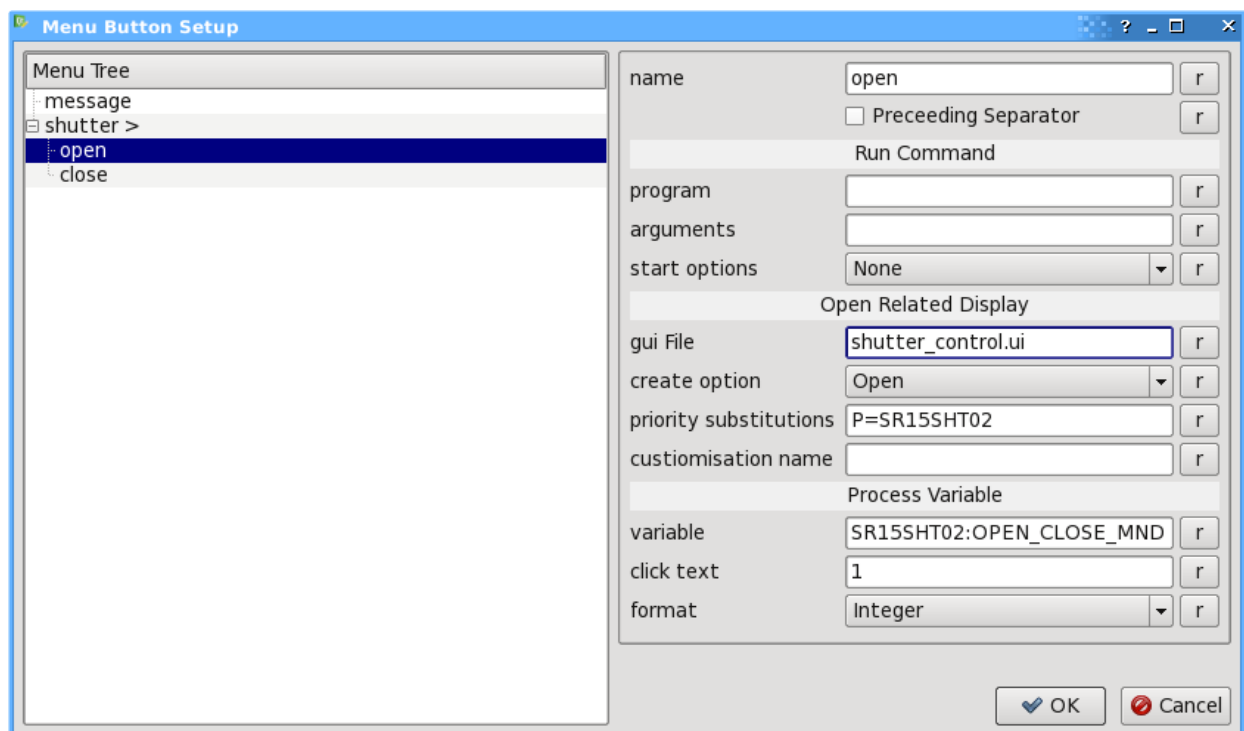


Figure 35 Menu Button Setup dialog

The right hand side enables the program to run, gui file to open and/or variable to write to to be defined. These are essentially as described in QEPushButton, QERadioButton and QECheckBox section above. The only additional "property" is the preceding separator checkbox which adds a menu separator.



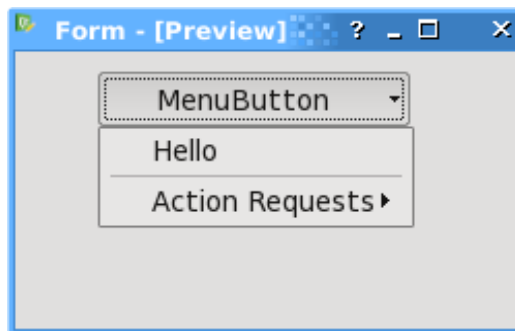


Figure 36 Menu Button example

### Restrictions

The following are not (currently) implement for QEMenuButton:

- a) Copy/paste within the menu hierarchy tree;
- b) Checkable menu items; and
- c) User Level visibility control of individual menu items (although the QEMenuButton as a whole has the regular user level style/visibility controls).

### Customisation Menus

An alternative to the QEMenuButton is the definition of a customisation file which is described in **Error! Reference source not found.** (Error! Reference source not found., page Error! Bookmark not defined.).

## 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:

- a) Write the values to the 'system' – the system being whatever IOCs and other Channel Access servers are currently available to the QEGui application;
- b) Read the values from the system;
- c) Read the values from the archive for a user nominated time;

## QE Framework – Widget Specifications










- 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 37 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 date 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;

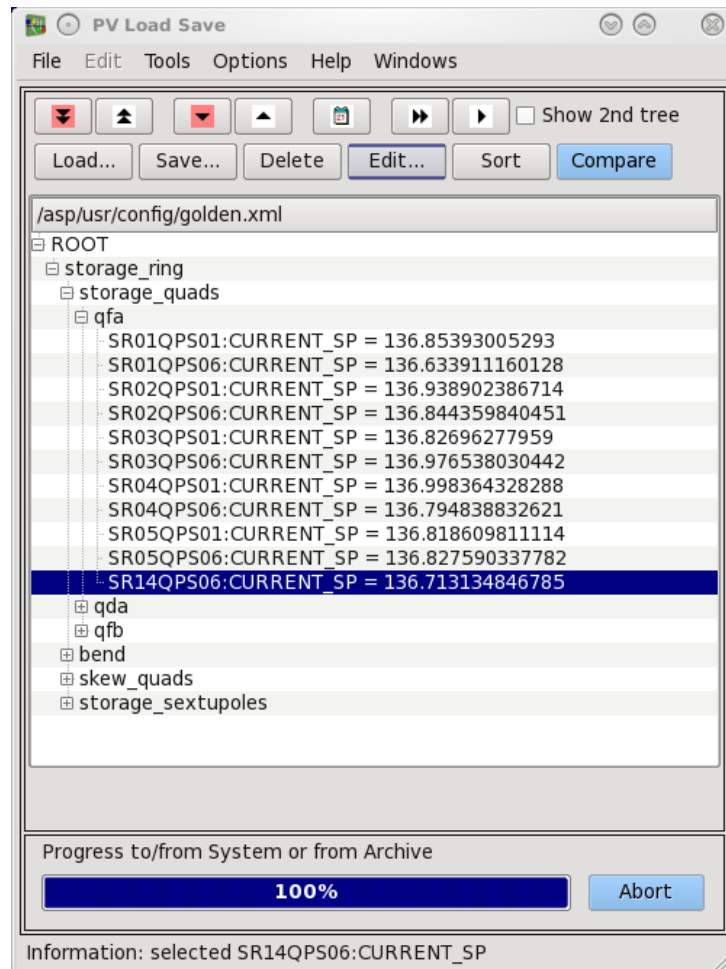


Figure 37 QEPvLoadSave – basic example.

- i) Load... - this button allows the user to navigate the file system to load a PV-Value xml file. (Optionally once 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

## QE Framework – Widget Specifications

- 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*).

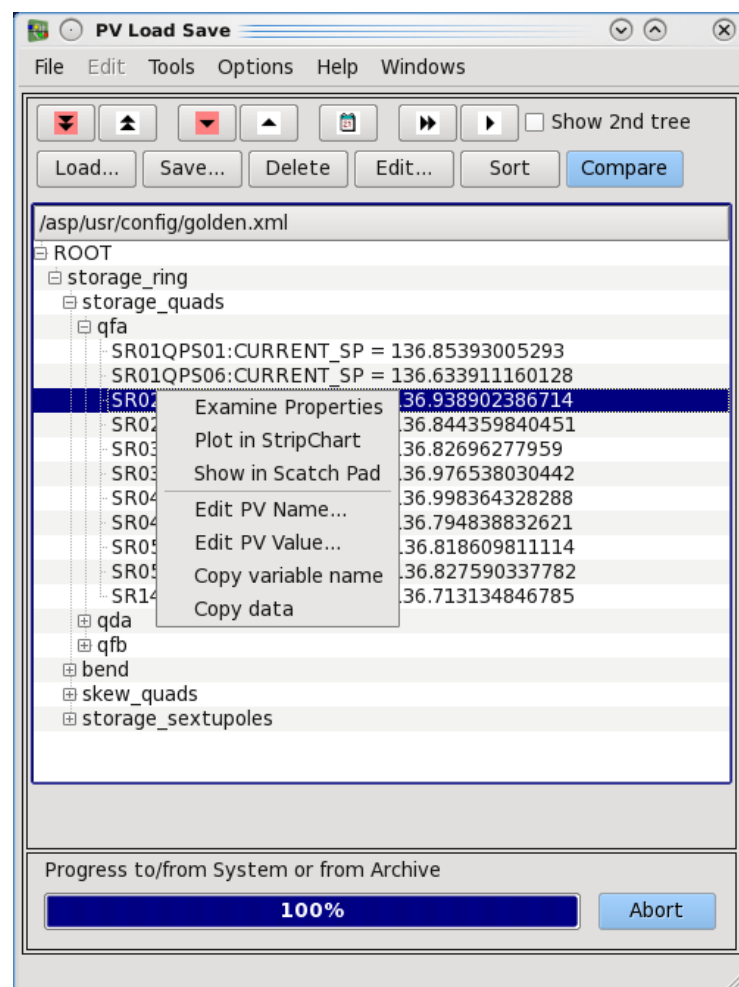


Figure 38 QEPvLoadSave – context menu example.

### 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 38 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:

## QE Framework – Widget Specifications



- 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 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 node is a group node, the new PV name is added to the end of the group. If the dropped 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 39 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 39 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).

### QEPvLoadSaveButton

The QEPvLoadSaveButton widget has the same basic functionality as the QEPvLoadSave widget but is an express version of it. Users can define an XML file, which has to have the same structure as the one used with QEPvLoadSave (see XML File Format XML File Format.), and the action they want the button to perform (Load PV values from file to the system, or read PVs from system to the file) and then load or save PVs using a single click.

The QEPvLoadSaveButton has the following custom properties (that can be controlled by the user):

- **configurationFile**  
Absolute path to the load/save XML configuration file. This file should follow the format described in the XML File Format XML File Format. configurationFile property supports macros

## QE Framework – Widget Specifications



- **defaultSubstitutions**  
Default substitutions for the macros defined in the configurationFile
- **action**  
SaveToFile/LoadToPVs – select one of the two behaviours this button can have
- **showProgressDialog**  
Display a progress modal dialog while the action is being executed

### 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 40 (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:

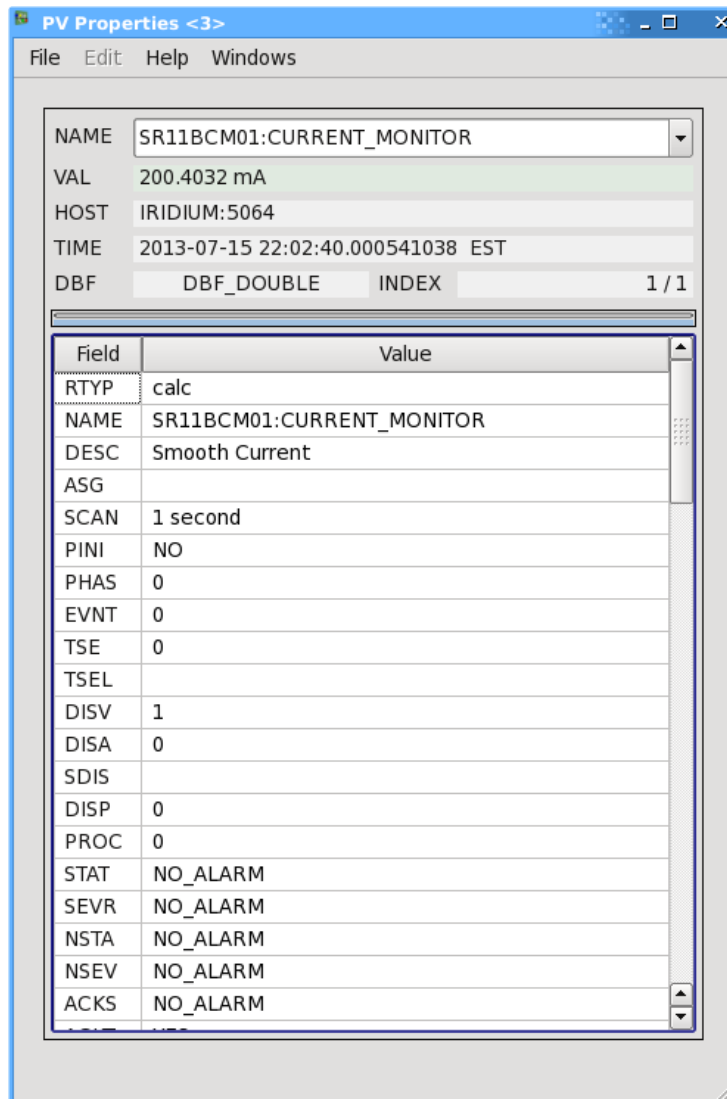
- a) the NAME field: this shows the current process variable used to source which record is being probed, i.e. SR11BCM01:CURRENT\_MONITOR.
- b) the VAL field: this shows the current value of the process variable. This is displayed using a QELabel, and as such has all the features of a QELabel such as 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 the this Process Variable;
- e) the DBF field show the PV's field type;
- f) the INDEX field show the element number and total number of elements for the PV. This widgets 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 filed 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 41below; and

## QE Framework – Widget Specifications

- 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.



The screenshot shows a window titled "PV Properties <3>". It has a menu bar with "File", "Edit", "Help", and "Windows". Below the menu bar, there are several fields for properties: "NAME" (SR11BCM01:CURRENT\_MONITOR), "VAL" (200.4032 mA), "HOST" (IRIDIUM:5064), "TIME" (2013-07-15 22:02:40.000541038 EST), and "DBF" (DBF\_DOUBLE INDEX 1 / 1). Below these fields is a table with two columns: "Field" and "Value". The table contains the following data:

Field	Value
RTYP	calc
NAME	SR11BCM01:CURRENT_MONITOR
DESC	Smooth Current
ASG	
SCAN	1 second
PINI	NO
PHAS	0
EVNT	0
TSE	0
TSEL	
DISV	1
DISA	0
SDIS	
DISP	0
PROC	0
STAT	NO_ALARM
SEVR	NO_ALARM
NSTA	NO_ALARM
NSEV	NO_ALARM
ACKS	NO_ALARM

Figure 40 QEPvProperties widget example examining a calc record.



## QE Framework – Widget Specifications

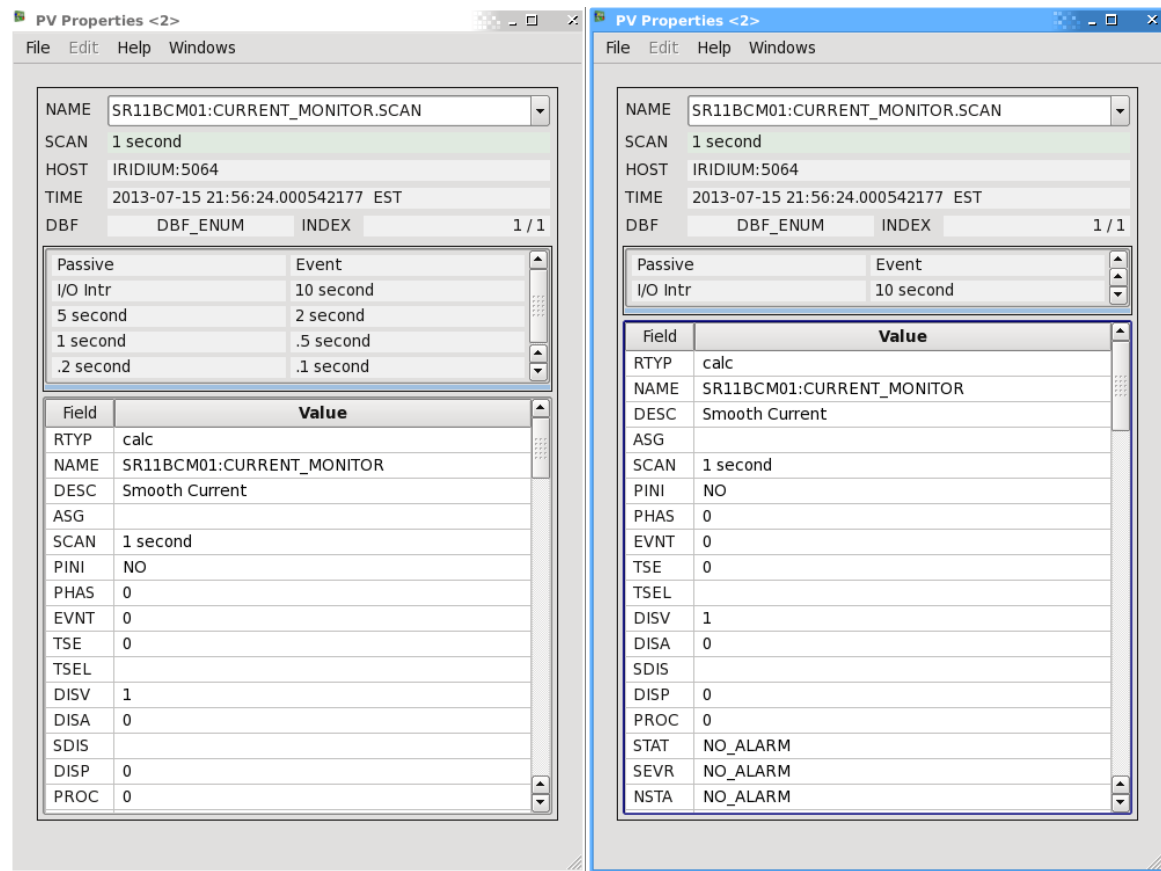


Figure 41 QEPvProperties widget example examining an enumeration PV.

### 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 in to 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;

## QE Framework – Widget Specifications



- g) 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, dxp, epid, er, erevent, event, fanout, genSub, histogram, login, logout 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 a predefined record type. The format of the file is a simple ASCII file consisting of:

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

## QE Framework – Widget Specifications



```
<<record_type2>>    -- introduce record type, e.g. <<aao>>
field_name1          -- field name, e.g. RTYP
field_name2          -- field name, e.g. DESC
field_name3          -- field name, e.g. SCAN
```

All field names are associated with the preceding record type.

### QRadioGroup and QERadioGroup

These widgets are now described in a separate document.

### 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

The QEScratchPad widget is designed and provided primarily to support the in-built Scratch Pad 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 scratch pad widget allows arbitrary process variables to be displayed in one convenient place on the user desktop. Up to 48 PVs may be displayed per widget instance. PVs are added to the widget dynamically at run time (details below), and cannot be predefined at design time as there are *no* variable properties associated with this widget.

Three fields are displayed for each PV added to the scratch pad, namely the PV Name itself, the value of the associated .DESC field plus the value of the PV. See example in Figure 42 below.

A screenshot of the QEScratchPad widget, which is a window with a menu bar (File, Edit, Tools, Options, Help, Windows) and a table. The table has three columns: PV Name, Description, and Value. It contains three rows of data, each with a light green background.

PV Name	Description	Value
SR11BCM01:CURRENT_MONITOR	BCM Smooth Current	6.679 mA
SR11BCM01:LIFETIME_MONITOR	BCM Lifetime	18.29 Hrs
SR11BCM01:LIFETIME_MONITOR.EGU	BCM Lifetime	Hrs

Figure 42 QEScratchPad displaying 3 PVs

PVs may be added to the scratch pad by:

## QE Framework – Widget Specifications

- Right clicking on an arbitrary widget to launch its context menu and then selecting "Show In Scratch Pad". This will open an new instance of the built-in Scratch Pad form and set the PV name as first entry on the form;
- Dragging an arbitrary EPICS aware widget onto an empty line on a scratch pad widget (unless full, an empty line is always maintained at the bottom of the widget);
- Right clicking on an empty PV Name field to launch the context menu and selecting either "Add PV Name..." or "Paste PV Name"; or
- Right clicking on an existing PV Name field to launch the context menu and selecting "Edit PV Name..."

### QEScript

The QEScript widget allows the user to define a certain sequence of external programs to be executed. This sequence may be saved, modified or loaded for future usage. Within Qt Designer, it has the following graphical representation (surrounded by a red rectangle):

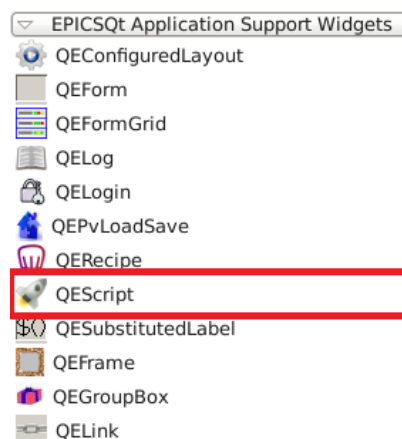


Figure 43 QEScript within Qt Designer

The QEScript has the following properties (that can be controlled by the user):

- showScriptList**  
Show/hide combobox that contains the list of existing scripts created by the user
- showNew**  
Show/hide button to reset (initialize) the table that contains the sequence of programs to be executed
- showSave**  
Show/hide button to save/overwrite a new/existing script

- **showDelete**  
Show/hide button to delete an existing script
- **showExecute**  
Show/hide button to execute a sequence of programs
- **showAbort**  
Show/hide button to abort the execution of a sequence of programs
- **showTable**  
Show/hide table that contains a sequence of programs to be executed
- **editableTable**  
Enable/disable table edition
- **showTableControl**  
Show/hide the controls of the table that contains a sequence of programs to be executed
- **showColumnNumber**  
Show/hide the column '#' that displays the sequential number of programs
- **showColumnEnable**  
Show/hide the column 'Enable' that enables the execution of programs
- **showColumnProgram**  
Show/hide the column 'Program' that contains the external programs to be executed
- **showColumnParameters**  
Show/hide the column 'Parameters' that contains the parameters that are passed to external programs to be executed
- **showColumnWorkingDirectory**  
Show/hide the column 'Directory' that defines the working directory to be used when external programs are executed
- **showColumnTimeout**  
Show/hide the column 'Timeout' that defines a time out period in seconds (if equal to 0 then the program runs until it finishes; otherwise if greater than 0 then the program will only run during this amount of seconds and will be aborted beyond this time)
- **showColumnStop**  
Show/hide the column 'Stop' that enables stopping the execution of subsequent programs when the current one exited with an error code different from 0
- **showColumnLog**  
Show/hide the column 'Log' that enables the generation of log messages (these messages may be displayed using the QELog widget)
- **scriptType**  
Select if the scripts are to be loaded/saved from an XML file or from an XML text
- **scriptFile**  
Define the file where to load/save the scripts (if not defined then the scripts will be loaded/saved in a file named "QEScript.xml")

## QE Framework – Widget Specifications

- **scriptText**  
Define the XML text that contains the scripts
- **scriptDefault**  
Define the script (previously saved by the user) that will be loaded as the default script when the widget starts
- **executeText**  
Define the caption of the button responsible for starting the execution of external programs (if not defined then the caption will be "Execute")
- **optionsLayout**  
Change the order of the widgets. Valid orders are: TOP, BOTTOM, LEFT and RIGHT.

The following figure illustrates the QEScript widget in production:

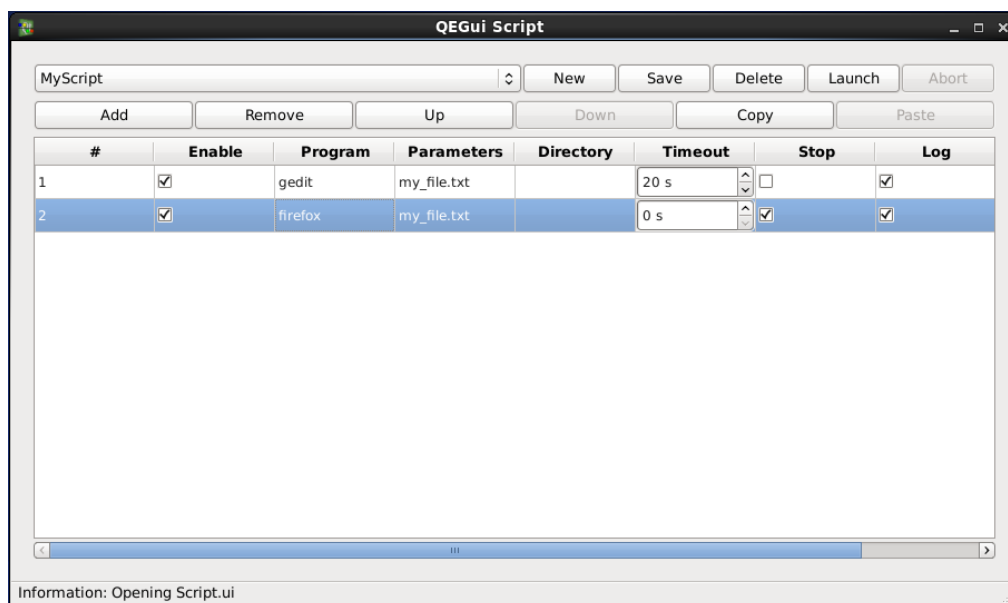


Figure 44 QEScript displaying a sequence of external programs to be executed(in this case "gedit" and "firefox")

## QEScalarHistogram and QEWaveformHistogram

The QEScalarHistogram and QEWaveformHistogram provide a means to display values as a histogram, aka bar chart. The former may be used to display up-to 100 scalar PV values, whereas the later may display a single array PV – each element of the array providing one of the values for the histogram. Apart from that, these widgets are so similar that they are describes together.

## QE Framework – Widget Specifications

Figure 45 shows an example of the QEWaveformHistogram widget displaying a 500 element array. Where, given the width of the bar and intervening spaces (both controllable via properties) is greater than the available space within the widget, and scroll bar is automatically enabled/made visible which allows the user to scroll the histogram display. If the user moves the cursor over an element of the histogram, the widget sends an information message which appears on the forms status bar. The message contains the PV name, element index (QEWaveformHistogram only) and current value. Note: for user display purposes, element indices are show as "[1]" to, say, "[500]", the display being for users and not for a C/C++ compiler.

The standard context menu and drag capability are supported by both forms of the histogram widget.

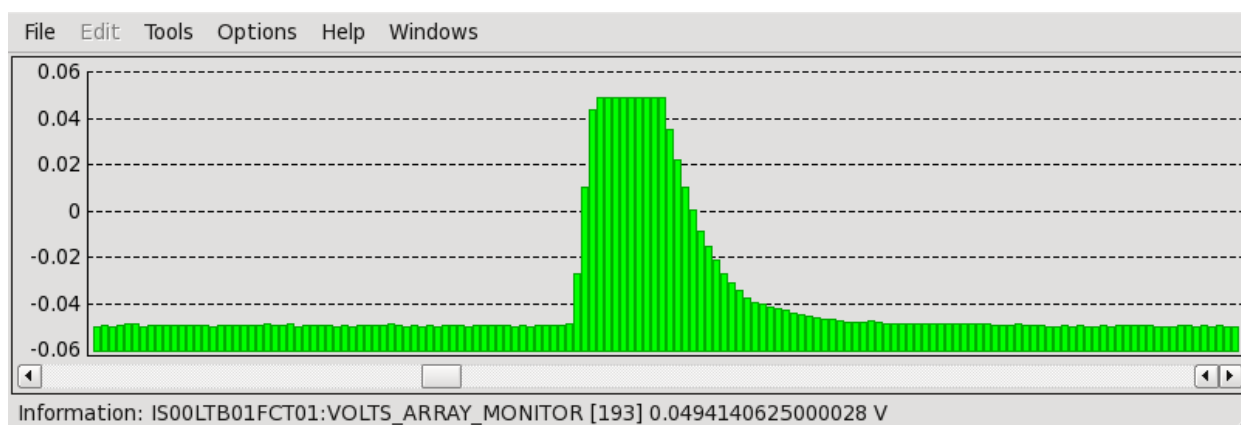


Figure 45 QEWaveformHistogram showing a 500 element array PV

### Properties

The following properties are specific to QEScalarHistogram and QEWaveformHistogram:

- variable** (QEWaveformHistogram) or **variable1, variable2, ... variable100** (QEScalarHistogram): defines the variable name(s) associated with this widget;
- variableSubstitutions**: defines any default substitutions to be applies to the variable name(s);
- autoBarGapWidths**: (boolean, default false) when true, the widget attempts to optimise the values used for the bar widths and the inter bar gaps so that the histogram best fits the available size;
- barWidth**: (int, default 8) defines the bar pixel width;
- gap**: (int, default 3) defines the pixel gap between bars;
- scaleMode**: (enumeration, default Manual) defines how the histogram scales the displaying values. Options are:
  - Manual**: Use the values specified by the minimum and maximum properties;
  - Auto**: Dynamically scale to accommodate the current values being displayed; and

3. **OperationalRange**: Use the (defined) LOPR and HOPR value(s) to define the range to be displayed.
- g) **minimum**: (double, default 0.0) defines the lower display range (manual mode);
  - h) **maximum**: (double, default 10.0) defines the lower display range (manual mode);
  - i) **baseline**: (double, default 0.0) defines the origin from where the bar is draw from (in the example above, this property was set to -0.06);
  - j) **logScale**: (Boolean, default false) when true, values are displayed using a logarithmic scale;
  - k) **barColour**: (QColor, default blue) when displayAlarmStateOption is 'Never', or when displayAlarmStateOption is 'WhenInAlarm' and the displayed variable is not in an alarm state, this property defines the colour to be used to draw the bars;
  - l) **drawBorder**: (boolean, default true) when true each bar is drawn with a boarder;
  - m) **orientation**: (enumeration, default Horizontal). Defines whether the histogram is drawn horizontally or vertically.

**NOTE**: The orientation property is currently ignored. This is for a planned future enhancement.

### 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 46. 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.



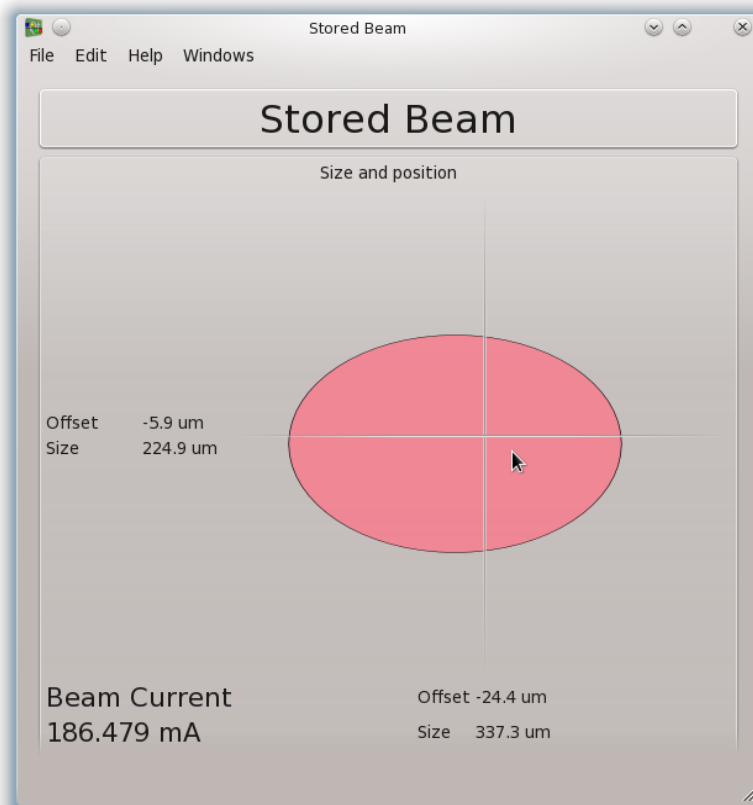


Figure 46 QEShape displaying stored beam

### 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.

## QE Framework – Widget Specifications



### Displayed object selection

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

- Line
- Points
- Polyline
- Polygon
- Rect
- RoundedRect
- 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
- Transperency
- 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...

## QE Framework – Widget Specifications



- The QEShape object shown in Figure 46. 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

## QE Framework – Widget Specifications



Object Type	Property	Use
	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.
• RoundedRectangle	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.
• 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
	lineWidth	Line thickness of border in pixels
	color1 to color10	Fill color selected by value after scaling and offset.
• 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

## QE Framework – Widget Specifications



Object Type	Property	Use
	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
	lineWidth	Line thickness of border in pixels
	color1 to color10	Fill color selected by value after scaling and offset.
• Path	point1	Start point
	point2	First control point
	point3	Second control point
	point4	End point
	drawBorder	Set if border is required
	fill	Set if fill is required
	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 47, 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

## QE Framework – Widget Specifications

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.

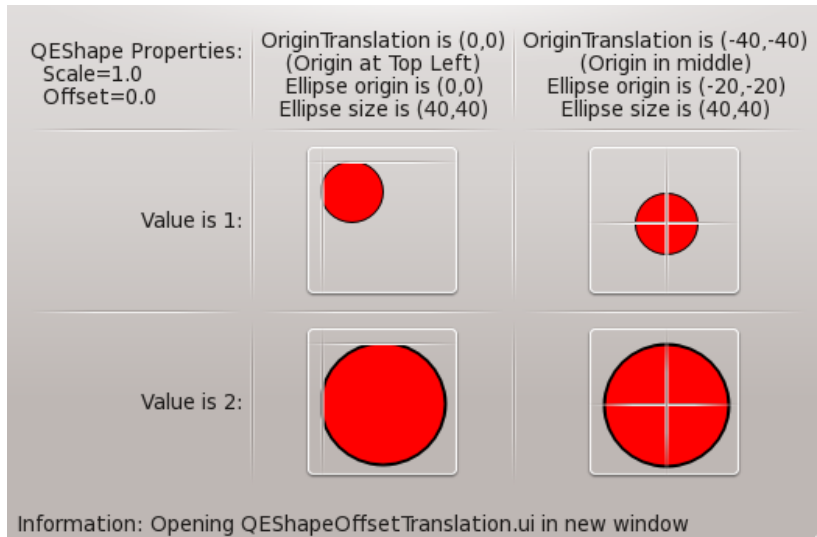


Figure 47 QEShape originTranslation example

In Figure 48 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.

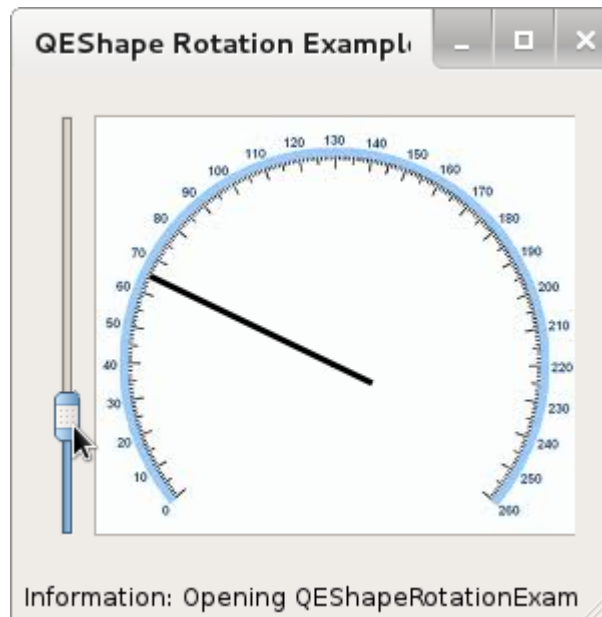


Figure 48 QEShape rotation example

### 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. It displays alarm state by default. 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, equalDiamond, arrowUp, arrowDown, arrowLeft, arrowRight, crossHorizontal, crossVertical. 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 *displayAlarmStateOption* property is set to 'Always' (the default) or is set to 'WhenInAlarm' and the PV is in an alarm state, the colour of the widget is determined by the alarm state of the PV.

## QE Framework – Widget Specifications



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 *displayAlarmStateOption* property is set 'Never', 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 arithmetic is used to ensure the PV value yields a number in the range. The modulus property (range 2 to 16, default 16) defines the modulo arithmetic behaviour. The widget has an *arrayIndex* property that can be used to select a single element from an array of data to provide the state value. The default array index value is 0.

The decision to provide up to 16 colours properties was some-what arbitrary; and while a user can only readily identify a limited number of colours (as opposed to distinguishing between subtle shade differences presented side by side) 16 was chosen so that a colour could be associated with each value of an *mbbi/mbbo* record.

Associated with each of the possible 16 values (again using modulo interpolation of the PV value) are a set of 16 flash properties (*flash0*, *flash1*, and so on to *flash15*, all default to false) that determine whether the widget should flash in that state. To support flashing, there are two additional properties. These are:

- a) *flashRate*: One of VerySlow, Slow, Medium (the default), Fast and VeryFast. These currently correspond to flashing rates of 0.25Hz, 0.5Hz, 1Hz, 2Hz and 4Hz respectively; and
- b) *flashOffColour*: This specified the colour used as the alternative to the regular "on" colour. The default off colour has an alpha value of 0, hence is clear.

All states that are flashing use the same flash rate and the same flash off colour. Even when the *displayAlarmStateOption* property is 'Always' or is 'WhenInAlarm' and the variable is in an alarm state, i.e. the colour being derived from the PV alarm state, the is-flashing state is determined from the PV value. If flashing or not flashing by alarm state is required, one option would be to monitor a record's SEVR field.

Figure 49 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 *displayAlarmStateOption* set to 'Never', 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 *displayAlarmStateOption* set to 'Always', 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).



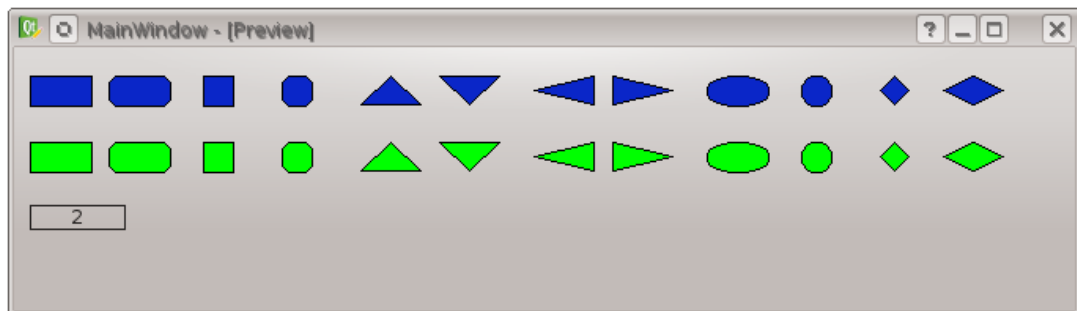


Figure 49 QESimpleShape examples

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

Figure 50 below shows the properties values selected for the second row of widgets.

Property Editor

Filter

qesimpleshape : QESimpleShape

Property	Value
<b>variable</b>	TS01EVR03:TIMESTAMP_MONITOR
variableSubstitutions	
shape	rectangle
format	FixedText
fixedText	
addUnits	<input checked="" type="checkbox"/>
flashRate	VeryFast
flashOffColour	<input type="checkbox"/> [200, 200, 200] (0)
localEnumeration	
colour0	<input checked="" type="checkbox"/> [200, 1, 131] (255)
colour1	<input checked="" type="checkbox"/> [7, 27, 200] (255)
colour2	<input checked="" type="checkbox"/> [8, 200, 146] (255)
colour3	<input checked="" type="checkbox"/> [200, 194, 0] (255)
colour4	<input checked="" type="checkbox"/> [200, 72, 13] (255)
colour5	<input checked="" type="checkbox"/> [200, 109, 144] (255)
colour6	<input checked="" type="checkbox"/> [147, 125, 200] (255)
colour7	<input checked="" type="checkbox"/> [135, 200, 137] (255)
colour8	<input checked="" type="checkbox"/> [200, 155, 147] (255)
colour9	<input checked="" type="checkbox"/> [255, 255, 255] (255)
colour10	<input checked="" type="checkbox"/> [0, 255, 85] (255)
colour11	<input checked="" type="checkbox"/> [110, 110, 110] (255)
colour12	<input checked="" type="checkbox"/> [12, 12, 12] (255)
colour13	<input checked="" type="checkbox"/> [200, 200, 200] (255)
colour14	<input checked="" type="checkbox"/> [0, 85, 255] (255)
colour15	<input checked="" type="checkbox"/> [255, 255, 0] (255)
flash0	<input type="checkbox"/>
flash1	<input checked="" type="checkbox"/>
flash2	<input checked="" type="checkbox"/>
<b>flash3</b>	<input checked="" type="checkbox"/>
flash4	<input type="checkbox"/>
flash5	<input checked="" type="checkbox"/>
flash6	<input type="checkbox"/>
flash7	<input checked="" type="checkbox"/>
flash8	<input type="checkbox"/>
flash9	<input type="checkbox"/>
flash10	<input type="checkbox"/>
flash11	<input type="checkbox"/>
flash12	<input type="checkbox"/>
flash13	<input type="checkbox"/>

Figure 50 QESimpleShape 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 51 shows several QESlider widgets connected a variable. The QESlider subscribes to the variable by default (subscribe property set by default).

## QE Framework – Widget Specifications

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

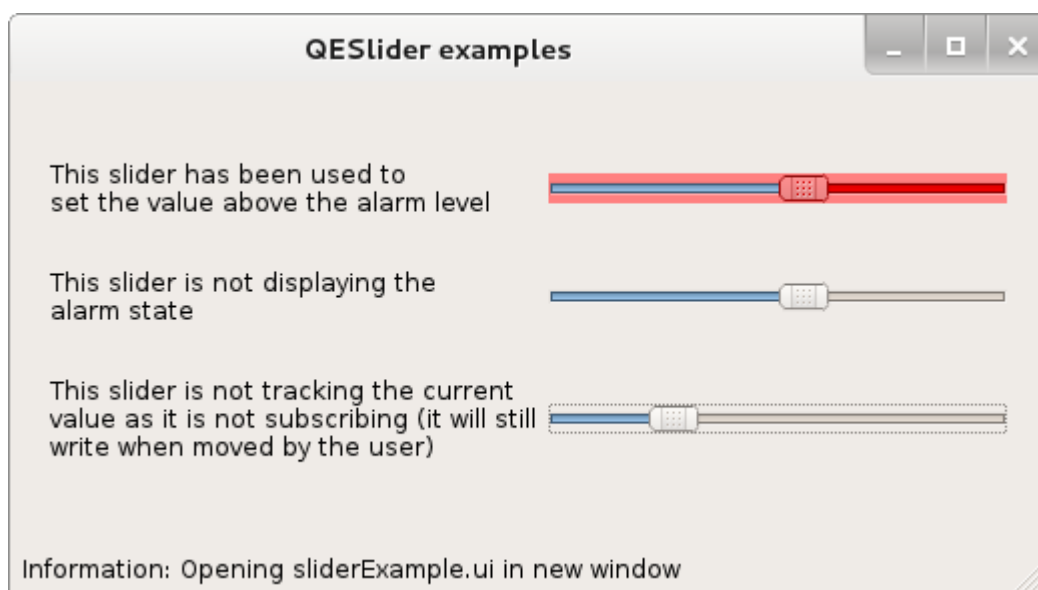


Figure 51 QESlider examples

## 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 QNumericEdit and QENumericEdit widget may be more appropriate. The example in Figure 52 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.

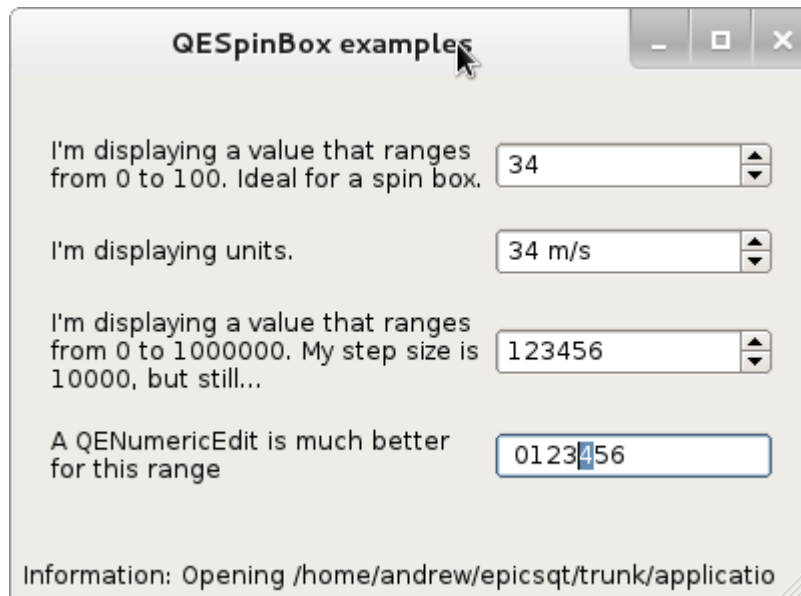


Figure 52 QESpinBox examples with a QNumericEdit where more appropriate

### QEStripChart

Please see the associated Strip\_Chart\_User\_Guide 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 53

## QE Framework – Widget Specifications

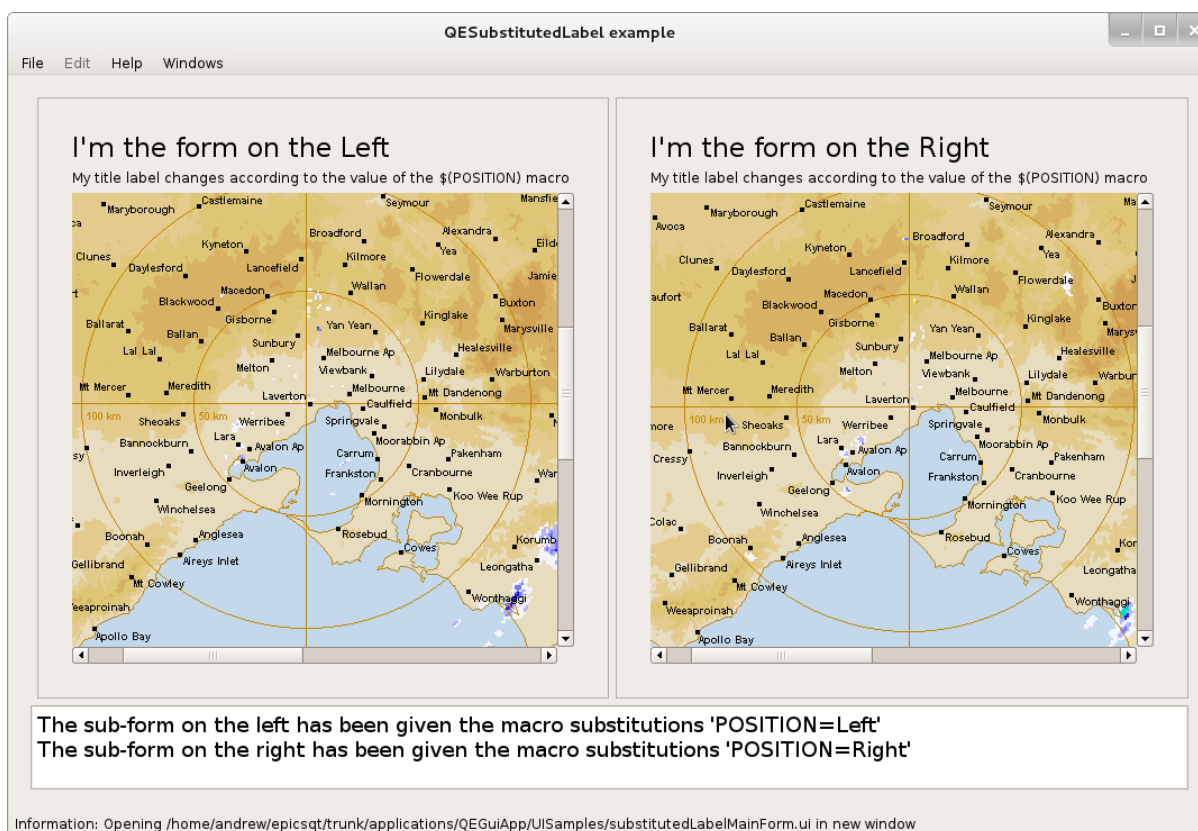


Figure 53 QESubstitutedLabel used to vary title in sub forms

## QETable

The QETable widget provides an EPICS aware table widget which is capable of displaying up to 20 array PVs in tabular form.

While independent of the QEPlotter widget it is particularly effective when connected to QEPlotter signals. Specifically, the QEPlotter 'crosshairIndexChanged' signal can be connected to the QETable 'setSelection' slot. When the same variables are being viewed by both widgets the cursor can be used in the plotter to simultaneously mark a point in the plot and select the equivalent data row in the table.

<Include figure of QEPlotter linked with QETable>

When in the default vertical orientation each column displays a consecutive element from an array EPICS variable. When in horizontal mode, the table and functionality is transposed.

## QE Framework – Widget Specifications



### QENTTable

This class provides an EPICS aware table widget which is capable of displaying a PV Access Normative NTable Type in tabular form. This is new since release 3.7.1

### 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. Secondly, 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

## QE Framework – Widget Specifications



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.



## QE Framework – Widget Specifications



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

## QE Framework – Widget Specifications



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

## QE Framework – Widget Specifications



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

## QE Framework – Widget Specifications



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.

## QE Framework – Widget Specifications



"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.

