QE Framework - QEGui and User Interface Design

Andrew Rhyder

Andrew Starritt

1st July 2020

Contents

Introduction

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

It assumes you have the QE Framework installed ready to develop 'code free' Control GUI systems. Refer to QE_GettingStarted.pdf for details on installing and configuring the QE Framework.

While widget properties are referenced, a definitive list of the available properties is available in document QE_ReferenceManual.pdf and more pragmatically from the Property Editor panel within designer.

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

Note: the specification of individual QE Widgets may now be found in the QE_QEWidgetSpecifications document or in specific widget documents.

License

Overview

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

Qt Designer

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

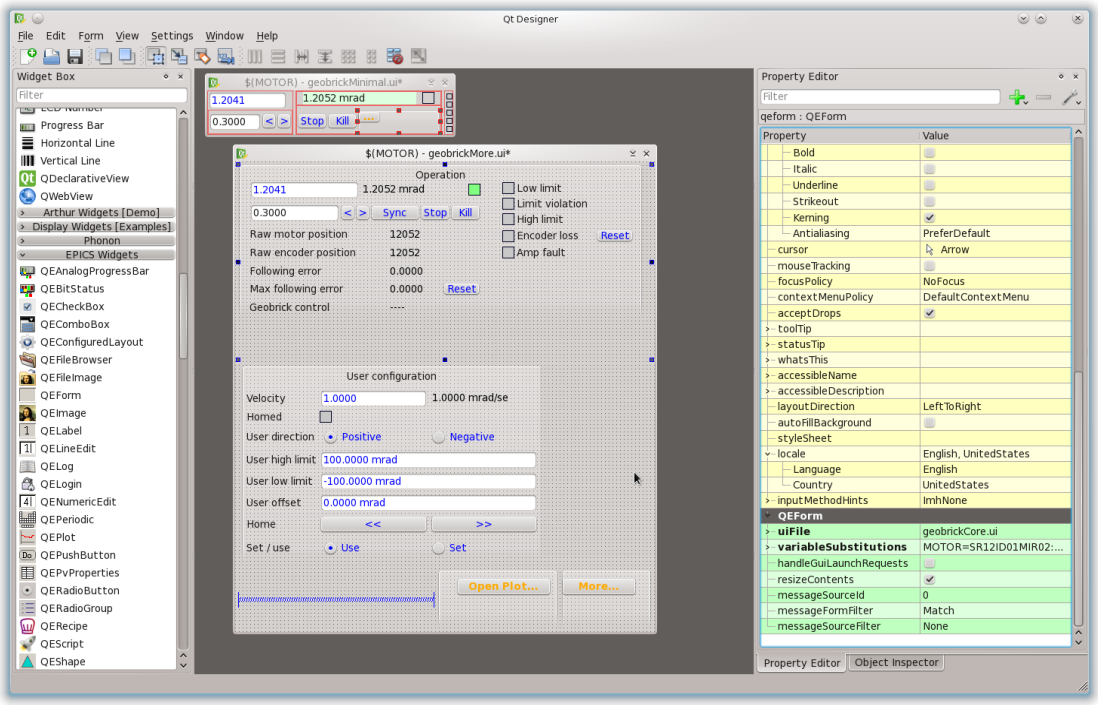

Figure 1 Designer being used to construct GUIs for use by the QEGui application.

QEGui

QEGui is an application use to display Qt User Interface files (.ui files). Almost all of the functionality of a Control System GUI based on the QE Framework is implemented by the widgets in the user interface files. QEGui simply presents these user interface files in new windows, or new tabs, and provides support such as a window menu and application wide logging. Indeed, the QE Framework widgets are readily useable in any Qt based display manager that can load .ui files, so is not a prerequisite to use the framework's widgets.

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

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

Refer to 'QEGui' (page 8) for documentation on using QEGui.

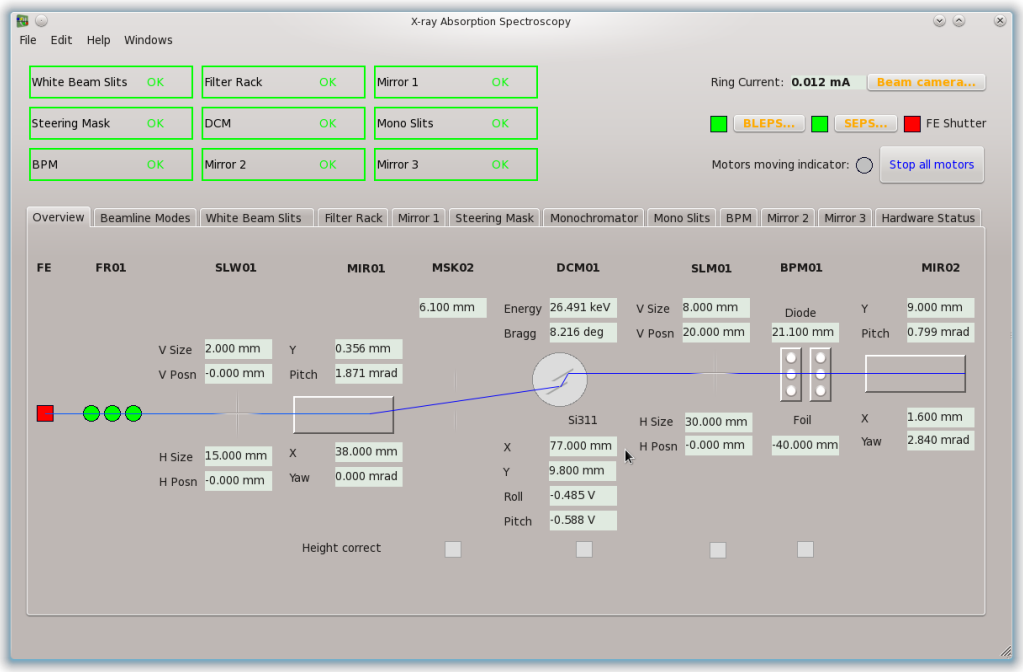

Figure 2 QEGui GUI display application example

QE widgets

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

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

QE widgets fall into two categories:

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

Example applications

The QEGui display application, along with Qt's Designer provides a code free package for developing and using comprehensive GUI applications. If you want to code your own application the QE framework provides support for:

- Creating applications using the QE framework widgets. The framework provides functionality you can use in your application for interaction with QE widgets and for functionality available in QEGui.
- Creating custom widgets based on QE framework widgets, or using QE framework functionality.
- Creating console based applications using QE framework functionality.

The following example applications are available within the framework to demonstrate how the QE framework can be used in your own application:

- QEMonitor
  A simple console based example application. Similar in operation to camonitor.
- QEWidgetDisplay
  A simple graphical example application that demonstrates using QE widgets within your code.
- QEByteArrayTest
  A simple console based example application that demonstrates using reading and writing raw data through a QEByteArray class.
- examplePlugin
  A Qt designer plugin with a single sample widget. This plugin shows you how to create your own custom set of widgets within a 'designer' plugin. Your own custom widgets can be based on the QE framework widgets, or use functionality provided by the QE framework library.

QEGui

Command format:

```
qegui[-a scale] [-f fontscale] [-s] [-e] [-b] [-r [configuration]] [-c
configuration] [-h] [-v] [-mmacros] [-ppath-
list][filename][filename][filename...]
```

Since version 3.6.4, QEGui accepts both long and short form options, and short forms can no longer be grouped, e.g. while:

```
qegui  -obu
```

was previously valid, each switch must now be separated:

```
qegui -o -b -u
```

Command switches and parameters are as follows:

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

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

- **-e, --edit Enable edit menu option.**
  When the edit menu is enabled Designer can be launched from QEGui, typically to edit the current GUI. Use of this option turns on the automatic reloading of ui files when a file modification is detected.

- **-a, --adjust_scale *scale* Adjust Scale.**
  Adjust the GUIs scaling. This option takes a single value with is the percentage scaling to be applies to each GUI. The value may be either an integer or a floating point number. If specified its value will be constrained to the range 40 to 400.
- **-f, --font_scale *fontscale* Adjust Font Scale.**
  Additional font scaling above and beyond the general GUIs scaling specified by the –a option. This option takes a single value which is the percentage additional font scaling. The value may be either an integer or a floating point number. If specified its value will be constrained to the range 40 to 400, although typically would be in the range 80 to 120.
- **-b, --disable_menu Disable the menu bar**
- **-u, --disablle_status Disable the status bar**
- **-o, --disable_autosave Disable configuration Auto-Save**
  Every 30 seconds the current configuration is saved. Also, the configuration is saved on exit. These features can be disabled with this switch.
- **-r, --restore [*configuration name*] Restore Configuration.**
  Ignore any filenames provided and restore the named configuration. If no name is provided 'Default' is assumed. Note, multiple configurations may be saved in the same configuration file. If configuration auto-save is active, the current configuration is saved with the name 'AutoSave' every 30 seconds and the configuration when the application is closed by the user is saved with the name 'ExitSave'. Both of these names may be used like any other configuration name. Using 'ExitSave' is useful as this effectively restarts the application where you left off.
- **-c, --configuration *configuration file* Configuration file.**
  Use the specified configuration file when saving and restoring configurations. If no file is specified 'QEGuiConfig.xml'in the current working directory is assumed.
- **-p, --path *path-list* Search paths.**
  When opening a file, this list of paths may be used when searching for the file. Refer to 'File location rules' (page11) for the rules QEGui uses when searching for a file.
  The search path format is platform specific and should be in the following forms:
  Linux: /home/mydir:/tmp:/home/yourdir
  Windows: 'C:\Documents and Settings\All Users; C:\temp;C:\epicsqt'
  (Note, platform specific path separators ':' and ';')
  (Quotes required if spaces are included in the paths)
  The search path may end with '...'in which case all sub directories under the path are searched. For example, assuming /temp/aaa and /temp/bbb exist, –p /temp/... will cause files to be looked for in /temp/aaa and /temp/bbb.

- **-h, --help Display help text explaining these options and exit.**
- **-v, --version Display version information and exit.**
- **-m, --macros *macros* Macro substitutions applied to GUIs.**
  Macro substitutions are in the form:
  keyword=substitution,keyword=substitution,... and should be enclosed in single quotes (‘) if there are any spaces.
  Typically substitutions are used to specify specific variable names when loading generic template forms. Substitutions are not limited to template forms, and some QEWidgets use macro substitutions for purposes other than variable names.
- **-w, --customisation_file *filename* Window customisation file.**
  This file contains named sets of window menu bar and tool bar customisations.
  Named customisations will be read from this file. If this option is not provided an attempt will be made to use QEGuiCustomisation.xml in the current working directory. A customisation file is optional.
- **-n, --customisation_name *customisation-name* Startup window customisation name.**
  The name of the window and menu bar customisation set to apply to windows created when the application starts (the .ui files specified on the command line). This name should be the name of one of the sets of window customisations read from the window customisation file.
- **-d, --default_customisation_name *customisation-name***
  Default window customisation name.
  The name of the window and menu bar customisation set to apply to all windows created when no customisation name has otherwise been provided. This name should be the name of one of the sets of window customisations read from the window customisation file. Typically, this is required when windows created through the ‘Open’ file dialog, or windows created through a QE push button require a different set of customisations to the system defaults. Note the customisation set specified in the –n switch only applies to windows created at startup and specified on the command line.
- **-t, --tile *title* Application title.**
  This title will be used instead of the default application title of 'QEGui'. Note, the application title is not always displayed in the title bar. Refer to ‘GUI titles’ (page 29) for details.

***filename filename ...* GUI filenames to open.**
Each filename is a separate parameter
If no filenames are supplied and no customisation name is supplied, the ‘File Open’ dialog is presented. Refer to ‘File location rules’ (page11) for the rules QEGui uses when searching for a file.

Switches may nolonger be grouped.

While not a part of QEGui, Qt style parameters can be added to the QEGui command line. Also, unknown options are ignored by QEGui, and are available to third party widgets via the QEOptions class For example:

```
Qegui –stylesheet mystyle.qss
```

```
qegui -style plastique

qegui –thired_party_option
```

File location rules

If a user interface file, customisation file, or any other file to be opened by QEGui has a path that is absolute, QEGui will simply attempt to open it as is. If the file path is not absolute, QEGui looks for it in the following locations in order:

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

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

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

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

QEGui uses file location rules defined by the QE framework. Refer to Finding files (page 35) for more details.

Saving and restoring configurations

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

- Save configuration.
  Saves the current configuration with a user specified name. The name of the last configuration read is offered as the default name. The user may also specify that the configuration is to be used when QEGui is started with the –r parameter.

- Restore configuration.
  Loads a configuration with a user specified name. The name of the last configuration read or written is offered as the default name.
- Manage configurations.
  One of more configurations can be selected and deleted.

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

QEGui automatically saves the current configuration every 30 seconds to a configuration called 'AutoSave'. When the user exits QEGui the application removes the 'AutoSave' configuration.

When QEGui starts it checks if there is an 'AutoSave' configuration. If it is present QEGui assumes that it did not shut down cleanly and offers to restore the 'AutoSave' configuration, returning the user to the point where QEGui failed. Note, if two instances of QEGui are started using the same configuration file, the second instance may see the 'AutoSave' configuration created by the first and assume it did not shut down cleanly.

QEGui also automatically saves the current configuration when the user exits to a configuration called 'ExitSave'. If the user wants to close the application then later return to where they left off they can restore the 'ExitSave' configuration manually from the 'Options' menu or on startup with the '–r' parameter (-r ExitSave).

Both 'AutoSave' and 'ExitSave' configurations can be disabled with the '-o' startup switch.

Like any other configuration, the user can restore, overwrite, or delete, both 'AutoSave' and 'ExitSave' configurations. Apart from restoring the last 'ExitSave' configuration after QEGui has started, however, these operations may not be particularly useful.

The configuration loaded on startup, the configuration file in use, and the status of the Configuration Auto Save is available from the 'About...' option in the 'Help' menu in the 'Configuration' tab.

Opening GUIs

New GUIs can be opened as follows:

- 'File->New Window' menu option. Creates a new window and presents the GUI file selection dialog. If the user selects a GUI file (a .ui file) the GUI is opened in the new window.
- 'File->New Tab' menu option. Creates a new tab in the current window and presents the GUI file selection dialog. If the user selects a GUI file (a .ui file) the GUI is opened in the new tab.
- From an already opened form in a tabbed window, the form may be reopened as a new form by selecting the "Reopen tab as new window" entry from the tab's context menu.
  Note: this action removes the tab/form from the current form.
- 'File->Open' menu option. Presents the GUI file selection dialog. If the user selects a GUI file (a .ui file) the current GUI (if any) is closed and the selected GUI is opened in its place.

- All QE framework buttons (QEPushButton, QERadioButton or QECheckBox) can open new GUIs. Refer to '**Error! Reference source not found.**' (page **Error! Bookmark not defined.**) for details.
- 'File->Recent...'menu option. Create a new window opening a recent GUI file (a .ui file). The path list and macro substitutions that were current when the file first added to the recent file list are used.

Note, a QEGui window does not need to be displaying a gui to add docks to it.

Built in forms

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

- PV Properties
- Strip chart
- User Level
- Message Log
- Plotter
- Table
- Scratch Pad
- PV Load/Save
- PV Correlation
- Archive Status

Generally, QE widgets can be dragged to these forms.

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

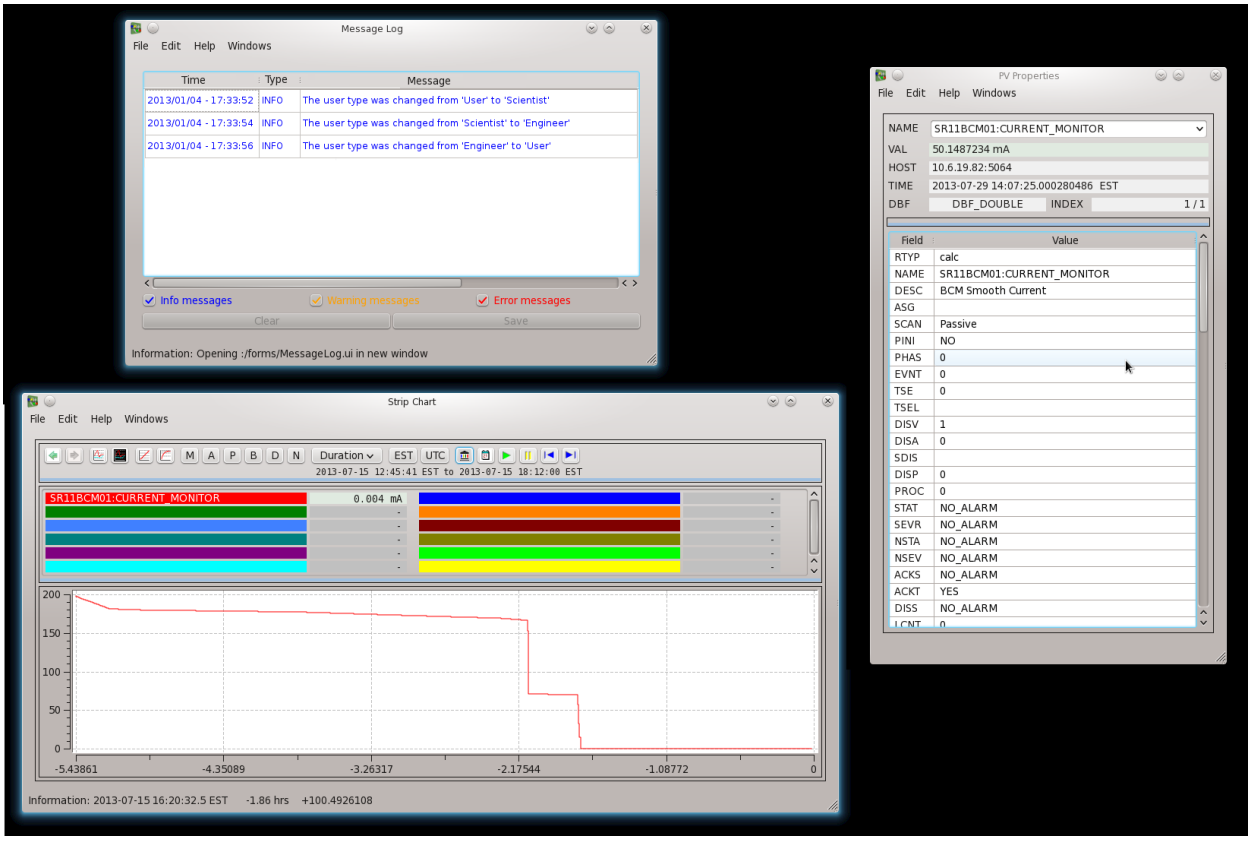

Figure 3 Some of QEGui built in forms

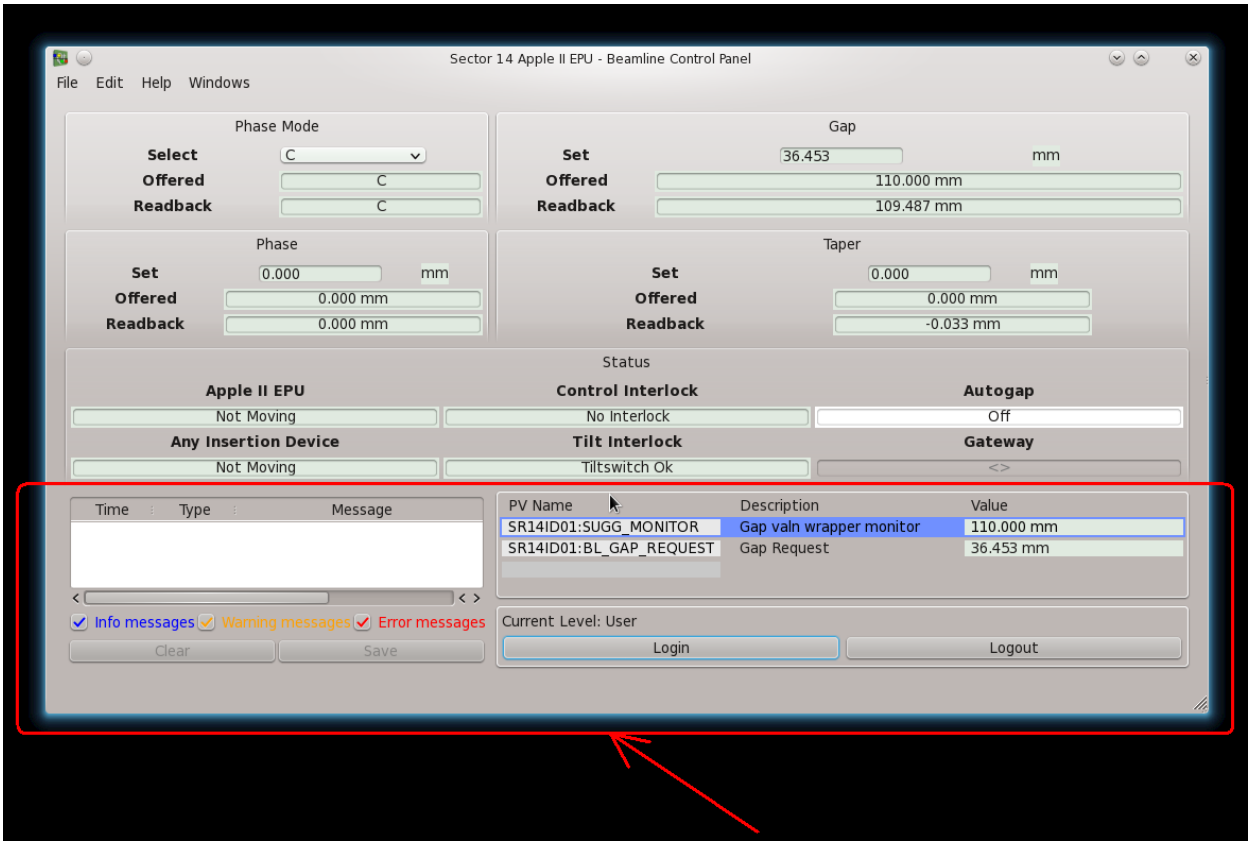

Figure 4 QE widgets used in QEGui's built in forms can be used in any GUI

Editing GUIs

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

- **Designer...**
  Start Qt's designer. This is just a convenient way to start designer.
- **Open Current Form in Designer...**
  Open the current GUI in Qt's designer. When saved, or when designer is closed, the current GUI will refresh to reflect any changes. This is a simple but powerful integration of QEGui and designer. A user looking at a GUI in QEGui can select this option, modify the GUI, close designer and see the changes with no further action required.
- **Refresh Current Form**
  This is a diagnostic option to restart an individual GUI.
- **Set Passwords...**
  Display the user level passwords and allow them to be modified. Refer to 'User levels' (page 30) for details on user levels. User level passwords will be saved when QEGui closes. This option is available if the 'Edit' menu is enabled and the 'Edit' menu is intended to only be enabled when a GUI system is being designed. If this model changes, for example if some GUI files are read only and the user is free to edit and create others using the 'Edit' menu then another mechanism for controlling passwords may be required. Note, the QEGui user level passwords are not intended to be highly secure and are not intended to provide protection from malicious activity. As well as starting QEGui with the –e parameter the user can also view passwords in the QEGui settings file.

Menu bar and tool button customisation

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

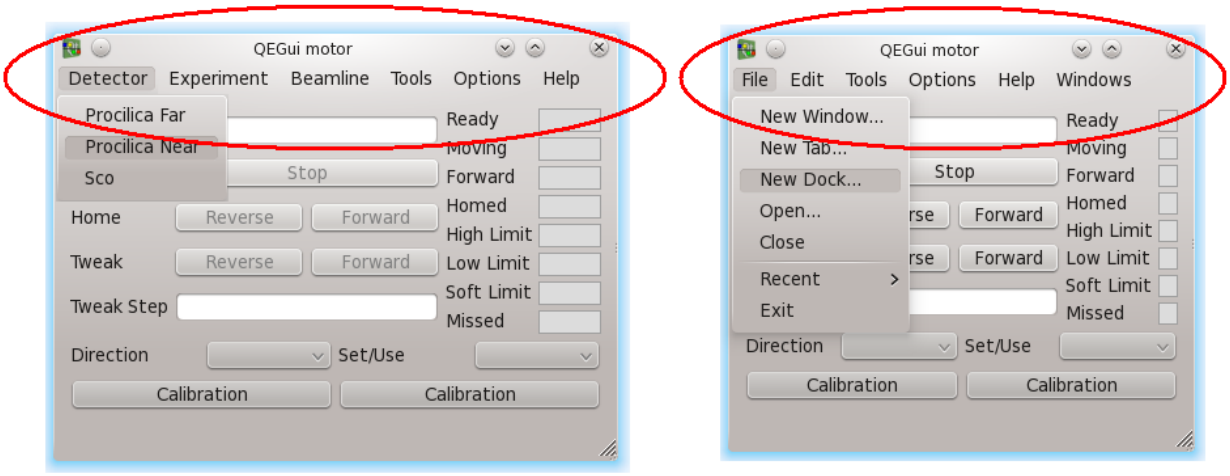

Figure 5 Menu bar customisation

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

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

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

Customisation is carried out in two general steps:

1) **Define a customisation file containing one or more named sets of customisations.**
   A set of customisations defines what buttons appear in what toolbars, what menus are required in the menu bar, what items are in the menus, and what the buttons and menu items do.
   The customisation file is loaded when QEGui starts. QEGui loads any customisation file specified on the command line with the –w parameter. If none is specified, QEGui attempts to load QEGuiCustomisation.xml in the current directory. See 'Command format:' (page 8) for details.
2) **Request a set of customisations by name when opening QEGui windows.**
   This occurs when QEGui starts, or when opening new GUIs. New GUIs can be opened from a menu item or button defined by the customisation itself, or by a QEPushButton widget in a GUI.

The QEGui –n command line parameter is used to specify the customisation set name used for all the windows started at startup; that is, all the user interface files specified on the qegui command line.
The QEGui –d command line parameter is used to specify the default customisation set name used if no name is specified in any other way. (If this is not specified, the final default is QEGui_Default)
See 'Command format:' (page 8) for details.

Customisation file format

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

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

```
<QEWindowCustomisation>

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

          <Item Name="menu item name" UserLevelEnabled="Scientist"|"Engineer">
            <Dock>
              <UiFile>file.ui</UiFile>
              <MacroSubstitutions>NAM1=v1,NAM2=v2,...</MacroSubstitutions>
              <CreationOption>[LeftDock|RightDock|TopDock|BottomDock|LeftDockTabbed
              |RightDockTabbed|TopDockTabbed|BottomDockTabbed|FloatingDock]</Creati
              onOption>
              <Hidden/>
            </Dock>
          </Item>

          <Item Name="menu item name">
            <Dock>
              <Title>Dock Title</Title>
            </Dock>
          </Item>

          <Item Name="menu item name">
```

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

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

</Menu>

<Placeholder Name="name"/>

<Button Name="name" Icon="icon.png" Toolbar="name"
Location="Left|right|Top|Bottom">
      <Window>
          <UiFile>file.ui</UiFile>
          <Title>Window Title</Title>
          <MacroSubstitutions>NAM1=v1,NAM2=v2,...</MacroSubstitutions>
          <CustomisationName>name</CustomisationName>
          <CreationOption>[Open|NewTab|NewWindow]</CreationOption>
      </Window>
</Button>

<Button Name="name2" Icon="icon2.png" Toolbar="name"
        Location="Left|right|Top|Bottom">
      <BuiltIn Name="name" />
</Button>

<Button Name="name3" Icon="icon3.png" Toolbar="name"
        Location="Left|right|Top|Bottom">
  <Program Name="name">
    <Arguments>arguments</Arguments>
  </Program>
</Button>

</Customisation>

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

...
```

```
</QEWindowCustomisation>
```

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

Tag name	Element description	Attributes (*Mandatory)	Child element tags (*Mandatory)
QEWindowCustomisation	A single element with this tag is expected in each customisation file.		CustomisationIncludeFile Customisation
Customisation	A named set of window customisations.	**Name*:** Used to identify a set of customisations.	Menu Placeholder Toolbar IncludeCustomisation
CustomisationIncludeFile	Name of XML customisation file to include.	**Name*:** Customisation file name	
IncludeCustomisation	Named set of window customisations to add to the current set being defined	**Name*:** Name of customisation set to include.	
Menu	A menu, or sub menu in the window menu bar.	**Name*:** Menu name	Menu Item
Item	A menu item on a menu in the window menu bar.	**Name:** Menu item name **UserLevelVisible:** User level at which this item will be visible. "User" (default) "Scientist" or "Engineer" **UserLevelEnabled:** User level at which this item will be enabled. "User" (default) "Scientist" or "Engineer"	Window Dock BuiltIn Program Separator Checkable
Separator	Element presence signals a menu item is to be preceded by a separator. Element contents ignored		
Checkable	Item is displayed checkable if this tag is present. The value contains the macro substitution to use determine the checked state of this item.		

Tag name	Element description	Attributes (*Mandatory)	Child element tags (*Mandatory)
Window	A GUI to open (in a main window)		UiFile* Title MacroSubstitutions CustomisationName CreationOption
Dock	Linked to an existing dock containing a control provided to the application by a QE widget. The menu item can hide or show the dock.		Title*
Dock	A GUI to open (in a dock of the current main window)		UiFile* MacroSubstitutions CreationOption Hidden
UiFile	GUI file name (.ui file)		
Title	Used to specify a window title for a new GUI, or to locate a dock containing a component hosted by a QE widget (by matching its title) When defining a title for a new GUI this overrides any title extracted from the .ui file being presented. Note, if a title is specified for a dock it is only used for locating an existing dock. It should not be used in conjunction with a .ui file in dock specifications.		
MacroSubstitutions	Macro substitutions to be passed on to all GUI components.		
CustomisationName	Name of customisation set to be applied to the window		
Hidden	If this element is present dock is to be created hidden. Element contents are ignored.		

Tag name	Element description	Attributes (*Mandatory)	Child element tags (*Mandatory)
CreationOption	Defines how new GUI is presented. If opening the GUI in a window the options are: • Open • NewTab • NewWindow If opening the GUI in a dock the options are: • LeftDock • RightDock • TopDock • BottomDock • LeftDockTabbed • RightDockTabbed • TopDockTabbed • BottomDockTabbed • FloatingDock		
BuiltIn	Name of function built in to the application, or built into a QE widget. If a WidgetName element is not provided, the function is expected to be built in to the application.	**Name*:** Function name	WidgetName
WidgetName	Name of the QE widget being displayed in a GUI that will receive a request for a built in function.		
Program	Command line command.	**Name*:** Program name. For example, firefox	Arguments
Arguments	Command line arguments		
Placeholder	Defines a location that the application can locate by name. For example, QEGui will search for a placeholder called 'Windows' and if found add a 'windows' menu.	**Name*:** placeholder identifier.	

Tag name	Element description	Attributes (*Mandatory)	Child element tags (*Mandatory)
Button	Main window toolbar button. Toolbars will be added to match requirements of the button.	**Name*:** Button title **Icon:** filename of image to use as icon **Toolbar:** name of toolbar (default is "Toolbar") **Group:** Toolbar group name **Location:**Toolbar placement –Left, Right, Top, Bottom **UserLevelVisible:** User level at which this item will be visible. "User" (default) "Scientist" or "Engineer" **UserLevelEnabled:** User level at which this item will be enabled. "User" (default) "Scientist" or "Engineer"	Window BuiltIn Program

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

```
<QEWindowCustomisation>

<Customisation Name="EXIT_ONLY">

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

</Customisation>

<Customisation Name="MAIN">

<Menu Name="Detector">

<Item Name="Procilica Far">
<Window>
<Checkable>DET=01</Checkable>
```

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

<Item Name="Procilica Near">
<Window>
<Checkable>DET=01</Checkable>
<Title>Procilica Near</Title>
<UiFile>detectorOverview.ui</UiFile>
<MacroSubstitutions>DET=02</MacroSubstitutions>
<CustomisationName>IMAGING</CustomisationName>
</Window>
</Item>

</Menu>

</Customisation>

<Customisation Name="IMAGING">

<Menu Name="Imaging">

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

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

</Menu>

<Button Name="btn1" Icon="icon1.png" Toolbar="name"
Location="Left">
<Window>
<UiFile imageStatistics.ui</UiFile>
<CreationOption>NewWindow</CreationOption>
</Window>
```

```
</Button>
<Button Name="About..." Icon="icon2.png" Toolbar="name"
 Location="Top"UserLevelEnabled="Scientist">
<BuiltIn Name="About..." />
</Button>
<Button Name="Firefox" Icon="icon3.png" Toolbar="name"
 Location="Left">
<Program Name="firefox">
<Arguments>www.google.com</Arguments>
</Program>
</Button>

</Customisation>

</QEWindowCustomisation>
```

Understanding customisations and fixing errors

While allowing customisation sub-sets to be in separate files provides a convenient way of defining commonly used sets, a deeply nested group of customisation files can be hard to follow when there is a problem. The QEGui 'Help->About...' dialog contains a log of the customisation files loaded at start-up to help diagnose problems. This log can be extensive. If you want to search through the log, copy and paste it from the dialog window to any text editor.

If there are errors in the customisation files, there may not be a 'Help->About...' menu item! If there are any errors found processing the customisation files, a message is written to the console and also presented in a message box before starting the application to note this and the log of the customisation file processing is written to customisationErrors.log.

Default customisations

The default menus in QEGui are implemented using the same customisation mechanism described in this document. On start-up, QEGui loads an internal customisation file which defines a window customisation set named `QEGui_Default`. The set of customisations named `QEGui_Default` is applied if no other named set is specified.
Because the default menus are defined using the customisation mechanism, the customisation set named `QEGui_Default`or parts of it such as `QEGui_Default_Help` or `QEGui_Default_Windows` can be redefined or included within another customisation set if required.
If customisation sets named `QEGui_Default`, `QEGui_Default_Help`, `QEGui_Default_Windows` etc are defined in any customisation file loaded, these will override the internal customisation set loaded when QEGui starts.
If you want to create a variation of the default set you might like to start with the original which can be found in the QEGui source code file QEGuiCustomisationDefault.xml.
If you want to include the entire default customisations within your own customisation set, add `<IncludeCustomisation Name="QEGui_Default"/>` to your customisation set.

If you want to include parts of the default customisations within your own customisation set, add any of the following to your customisation set:

```
<IncludeCustomisation Name="QEGui_Default_File"/>
<IncludeCustomisation Name="QEGui_Default_Edit"/>
<IncludeCustomisation Name="QEGui_Default_Tools"/>
<IncludeCustomisation Name="QEGui_Default_Options"/>
<IncludeCustomisation Name="QEGui_Default_Help"/>
<IncludeCustomisation Name="QEGui_Default_Windows"/>
```

For example, if you want your own custom menus, but the default 'Help' and 'Windows' your customisation set definition would be in the following form:

```
<Customisation Name="MyCustomisation">

    ...all your own custom menus here...

    <IncludeCustomisation Name="QEGui_Default_Help"/>
    <IncludeCustomisation Name="QEGui_Default_Windows"/>
</Customisation>
```

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

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

Window customisation from a QE button widgets

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

Toolbar buttons

Toolbars are not defined directly in customisation files. Toolbars are created as required when referenced in a button definition. In the following button definition a top toolbar called Toolbar1 will be created to hold the 'About...' button:

```
<Button Name="About..." Icon="about.png" Toolbar="Toolbar1"
 Location="Top">
<BuiltIn Name="About..." />
</Button>
```

Since multiple toolbars of the same name cannot be created, if a toolbar has already been created of the required name a button is added to it regardless of the buttons preferred location for that tool bar.

Separators in menus

Note, separators in menus are considered an attribute of the menu item following the separator, as in the following example:

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

```
</Item>
```

Checkable menu items

Menu items may be made checkable by including a `<Checkable>` tag. The value for this tag is a macro substitution which will determine if the item is actually checked. If the macro is defined correctly when the menu item is created the item is checked. The following example is a fragment for customisations used to customise a QEGui main window to manage a range of cameras. Each camera requires different GUIs and menu options. As each option in the menu is selected the window is re-opened with a different set of customisations to suit the camera selected. Each set of customisations, however, includes this menu.

In the following example a customisation set called DET_SELECT is used to define a menu where each menu item (re)opens the QEGui main window with customisations to suit a selected camera. Each of the specialised customisations would include this DET_SELECT customisation set at the same point in the menu bar. As the menu bar changes to suit the camera, the user would see the same camera selection menu appear in the same place, with the current camera selected.

```
<Customisation Name="DET_SELECT">
<Menu Name="Detector">
<Item Name="Camera 1">
<Checkable>CAM=01</Checkable>
<Window>
<CustomisationName>CAM1</CustomisationName>
<MacroSubstitutions>CAM=01</MacroSubstitutions>
<Title>DetectorSystem - Camera 1</Title>
<CreationOption>Open</CreationOption>
</Window>
</Item>
<Item Name="Camera 2">
<Checkable>CAM=02</Checkable>
<Window>
<CustomisationName>CAM2</CustomisationName>
<MacroSubstitutions>CAM=02</MacroSubstitutions>
<Title>DetectorSystem - Camera 2</Title>
<CreationOption>Open</CreationOption>
</Window>
</Item>
</Menu>
</Customisation>
```

Note, the above example is a bit recursive; in the intended use the same menu is re-created by an item from the menu. When re-created, the checked state will change as determined by the new macro substitutions specified by the item.

Built-in functions

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

The QEGui application provides the following built-in functions:

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

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

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

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

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

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

Repeating sections of a set of window customisations

Identical menus may be repeated in multiple sets of customisations. For example, two sets of customisations may be defined for various GUIs, but both require the same 'Tools' menu. Even though only two customisation names are required when starting the GUIs, a third may be defined for inclusion in the others as follows:

```
<QEWindowCustomisation>

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

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

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

</QEWindowCustomisation>
```

Menus associated with a Push Button

The **Error! Reference source not found.** widget (**Error! Reference source not found.**, page **Error! Bookmark not defined.**) also provide a menu capability.

Tricks and tips (FAQ)

GUI titles

The QEGui application displays a title for each main window. Where possible, this title relates to the GUI currently being displayed. The GUI designer can leave the system to automatically generate this title based on the GUI file names, or use several mechanisms to specify window titles.

Regardless of the how the title is generated current macro substitutions are applied to the title before using it as the GUI title.

The following rules are used to determine the window title.

- If no GUI is open, the application title will be used as the window title. By default application title is 'QEGui'. This can be replaced, however, using the –t switch when starting QEGui.
- If a GUI is open, and there is no GUI title specified through any of the mechanisms below, the window title will consist of the application title and the GUI .ui filename prefix in the form 'QEGui *prefix*'.
- If the windowTitle property of the top level widget in a user interface file is present and not simply the default title for that widget type, it is used as the GUI title. Some widgets have a default value for the windowTitle property which is ignored. For example the default windowTitle property for a QForm widget is 'Form'. This default title will be ignored when choosing a GUI title. Figure 6 shows a windowTitle property, that includes macros, being edited in Designer, with the same user interface being displayed by QEGui with the appropriate macro substitution.
- If a GUI is specified through the application menu customisation system, and a title is specified with a <title> tag, this is used as the GUI title. See 'Menu bar and tool button customisation' (page 15) for details.

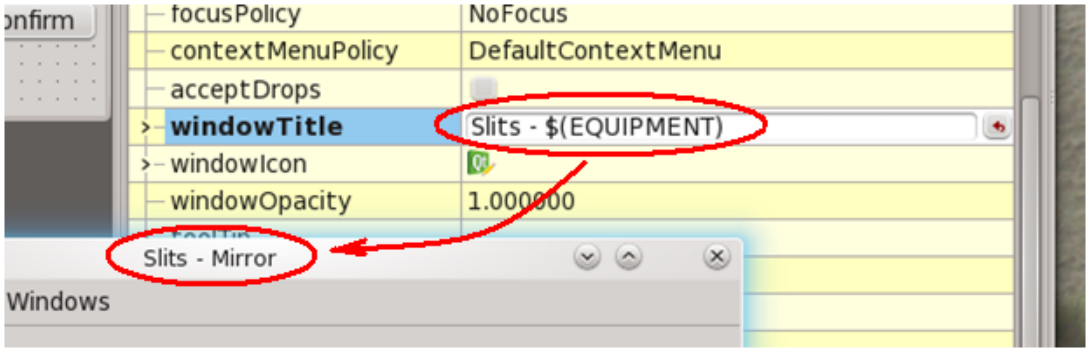

Figure 6 windowTitle Property in designer with actual translated window title on form in foreground

User levels

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

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

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

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

User levels control the behaviour of QE widgets, the menu bar items and tool bar buttons in the QEGui application, and are available programmatically to user written code.

The following guidelines should be considered when applying user levels.

- **Avoid hiding things based on user level.**"I'm sure it was there yesterday..." It may not be clear to a user that something they recall is no longer visible because of a user level change. While hiding widgets can free up space, just disabling them may leave the GUI more familiar.
- **Avoid significant changes to a GUI layout based on user level changes.** Confusion will be caused if changes in visibility, position, or size related properties cause the GUI to become unfamiliar. Enabling and disabling buttons or menu options to provide access to other user level based GUIs means the GUIs remain familiar as the user level changes.
- **Manage user level related widgets as a group.** This is much easier to maintain, and is clearer to the user. For example, rather than managing a set of engineering widgets separately, place them in an appropriately titled group box and enable and disable only the group box based on user level.
- **Consider user level based sub-forms.** Design sub-forms containing information required for the different user levels and include them laid out in single a top level GUI each controlled by user level. If forms are laid out in the top level GUI in order of user level ('User', 'Scientist', then 'Engineer') the top level GUI will gracefully expand and contract as the user level changes.

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

While the user level can be set and read programmatically using the ContainerProfile class, it is intended to be set using the QELogin widget and acted on by other QE widgets and by the QEGui application. The QELogin widget imposes a hierarchy to the user levels, requesting passwords when increasing user

levels but allowing the user level to be reduced without authority. Refer to '**Error! Reference source not found.**' (page **Error! Bookmark not defined.**) for details of how passwords are set.

The user levels are used to control individual QE widget behaviour and QEGui menu bar and tool bar button behaviour.

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

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

For the QEGui application, customised menu bar items and tool bar buttons can be defined to be enabled or be visible only at a specified user level. Refer to 'Menu bar and tool button customisation' (page 15) for details.

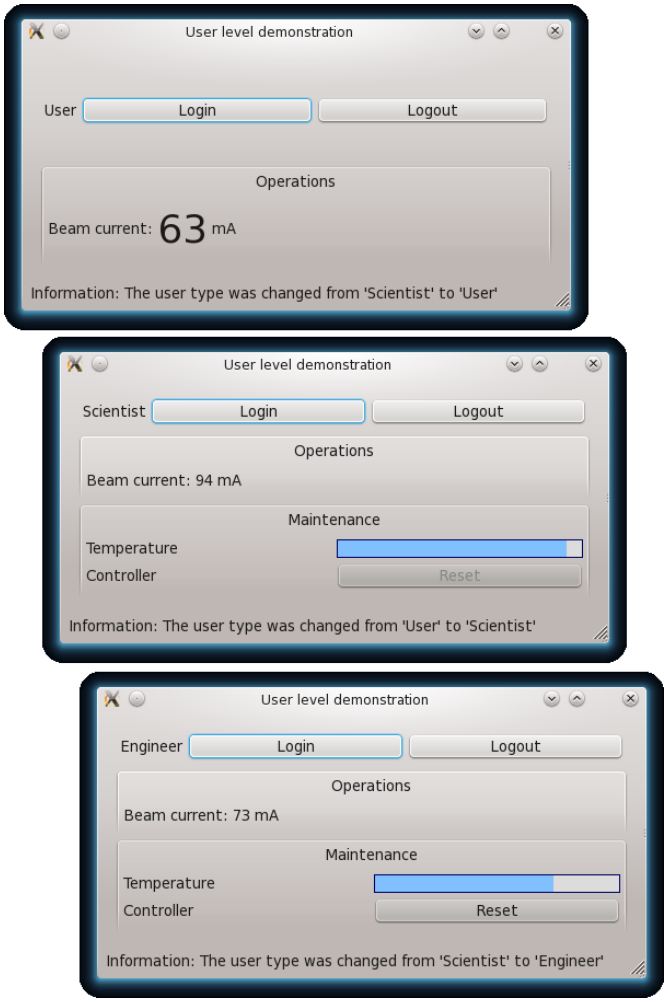

Figure 7 User level example

Logging

Several QE widgets generate log messages. These can be caught and displayed by a QELog widget, or a user application. This section describes the overall QE framework message logging system. Refer to '**Error! Reference source not found.**' (page **Error! Bookmark not defined.**) for a description of the QELog widget.

Log messages have three attributes:

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

Simplest use:

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

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

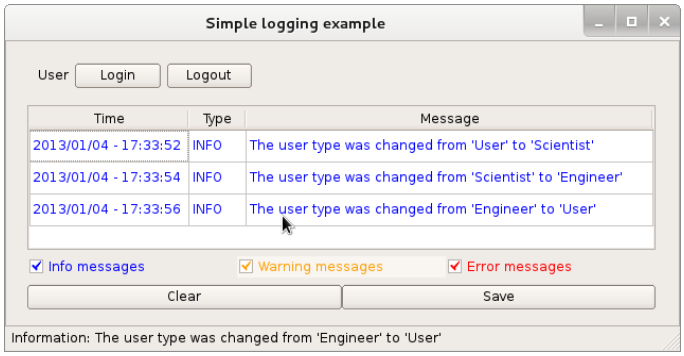

Figure 8 Simple logging example

Complex use:

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

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

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

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

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

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

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

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

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

application itself also uses a UserMessage class to catch and present the same messages on its status bar.

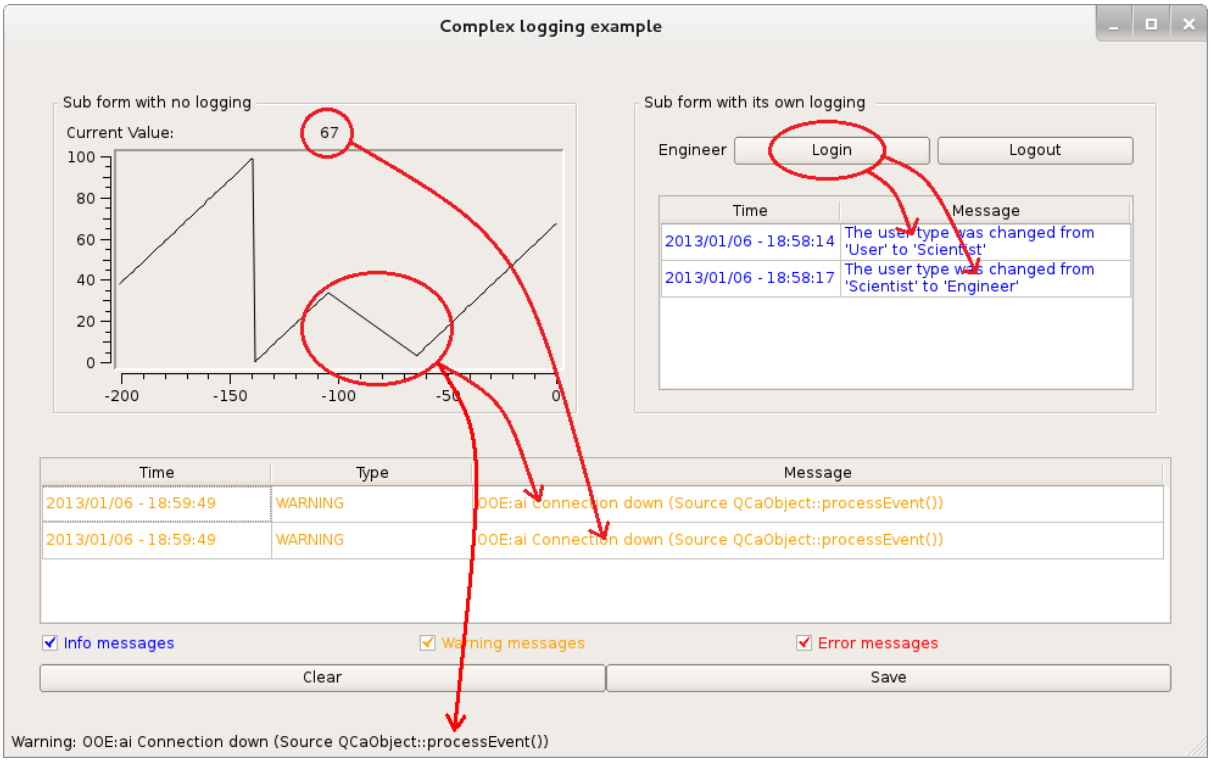

Figure 9 Complex logging example

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

Finding files

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

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

In the GEQui application, the –p switch is used to specify a path list which is published in the ContainerProfile class. In the QEForm widget, macro substitutions can be used to modify file names. Refer to 'File location rules' (page11) for details on how the QEGui application and QEForm widgets search for a user interface file given absolute and relative file paths.

Sub form file names

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

- No path should be specified for sub forms in the same directory as the parent form.
- A relative path should be given for sub forms in a directory under the parent form directory.
- Paths to directories containing generic sub forms can be added to the–p switch.

Refer to 'File location rules' (page11) details on how QEGui searches for a user interface file given absolute and relative file paths.

Who is in charge of the size of my form?

You choose!

When designing forms you get to choose if anything in the form is fixed size, can grow, shrink, or is managed by a Qt 'layout'. This freedom can cause its own problems if you don't consider resizing when designing GUIs.

To ensure your GUIs behave well you should consider the following:

- Have you decided on a consistent policy regarding how your GUIs will be resized (or if they will be resizable at all)?
- There is a lot of great Qt documentation of the relevant widget properties. Are you are well aware of how to use the following properties?
    - sizePolicy
    - minimumSize
    - maximumSize
    - geometry
    - layoutColumnStretch
    - layoutRowStretch
- Are you familiar with Qt layouts? Layouts are a very powerful tool for arranging widgets on a form. They are dynamic and manage the position and size of widgets when a form is resized.
- Even when a form is designed to be fixed size, you should use layouts to help arrange widgets consistently and to make future modifications easier.
- Are you aware a layout can be imposed directly on container widgets such as QForm, QFrame, QGroupBox, etc? You should never drop a layout into these widgets as the top level layout manager. Figure 10 (page 37) gives correct and incorrect examples of top level layouts.
- Are you aware of how QEGui helps with GUI resizing? The QEGui application will look at the top level widget of a form when opening it and decide if it should help in managing the size of the GUI. If the top level widget is a QScrollArea, or if the top level widget has a 'layout' set, then QEGui does not interfere and lets the GUI look after resizing itself. If the top level widget is not a scroll area, or the top level widget does not have a 'layout' the QEGui will load the GUI within a QScrollArea widget of its own.

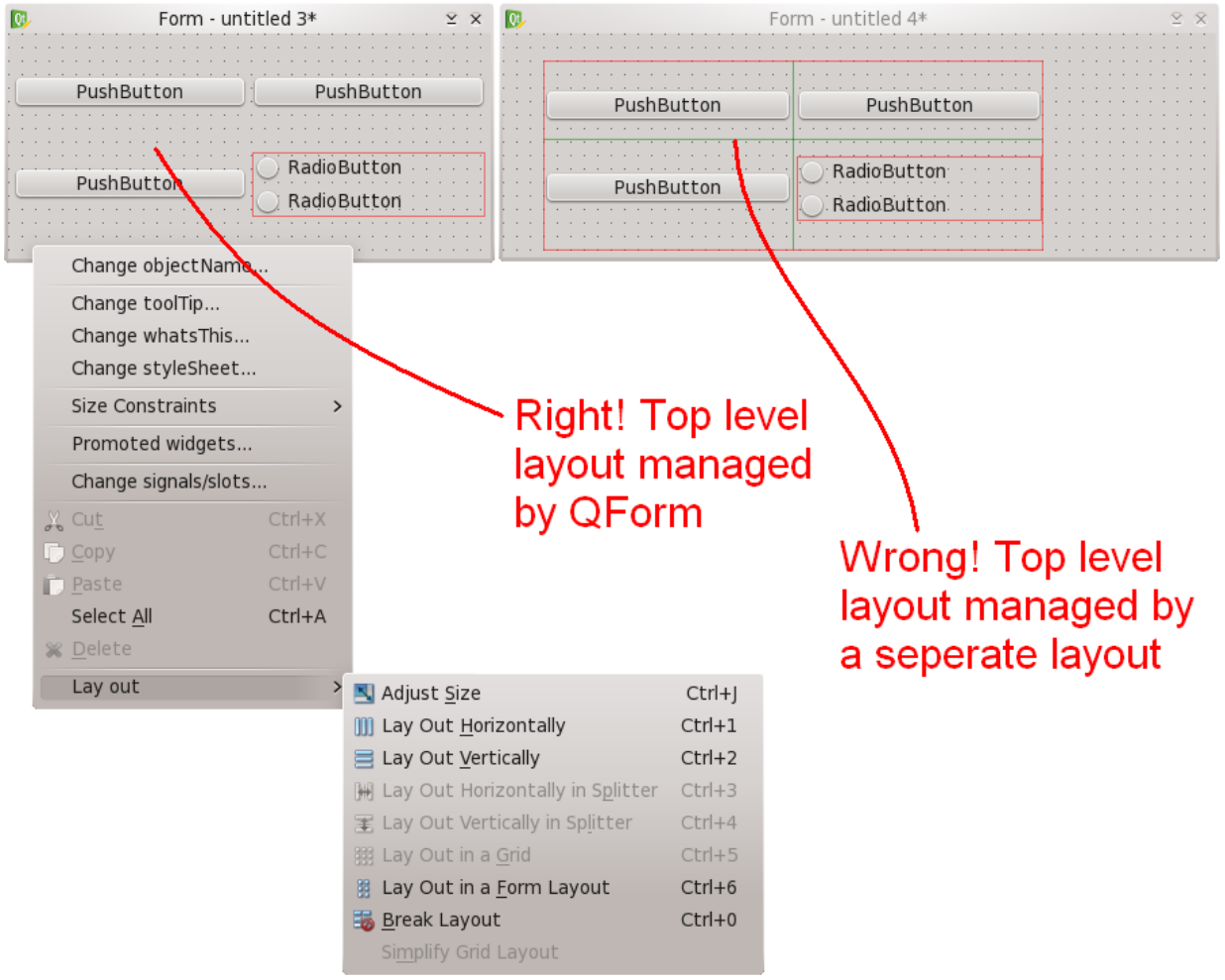

Figure 10 Layout use and misuse

Sub form resizing

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

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

Refer to **Error! Reference source not found.** (page**Error! Bookmark not defined.**) for complete details about the QEForm widget

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

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

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

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

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

What top level form to use

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

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

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

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

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

GUI based on a QScrollArea won't scroll in QEGui

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

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

How does a user interact with an updating QE widget

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

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

Widgets disappear when escape is pressed!

A QDialog widget has been used as the top level form. Use a QWidget instead. A QDialog will work as a QEGui form, but a feature of a QDialog is that the escape key causes it to close. The main task of the QEGui application is to load .ui files. Apart from a small amount of introspection to determine if the loaded form will be managing its own resizing, QEGui does not know or care what the top level widget is in the .ui file it is loading. You may have used a QDialog widget as the top level form simply because this was the default Qt's designer offered. Refer to 'What top level form to use' (page 38) for selecting the best top level widget.

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

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

To avoid this problem, set 'displayAlarmStateOption' to 'Never' QE widgets that are displaying a field unrelated to the alarm state.

A QEPlot widget is not displaying updates

For QE Framework version 2.3.5 and prior, if the time on the machine running the QEPlot widget is ahead of the time on the machine generating the data by more than the time span, the QEPlot widget will not display the data. For example, if the QEPlot widget is displaying the last minute's data and an update arrives for data two minutes ago (from a machine with the time set two minutes behind), the QEPlot widget will discard the update along with any other values earlier than the time span being presented.

Following QE Framework version 2.3.5 QEPlot only uses the data timestamp as-is within reasonable limits. If necessary the timestamp is adjusted to stay within 100mS into the future and 500mS into the past. This should cater for typical limitations in machine time synchronisation and occasional network latencies.

Droppable widgets as scratch pads and customisable GUIs

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

Note, The QEGui application provides a built in scratch pad form (refer to 'Built in forms' (page 13) for details). The QEScratchPad widget used within this built in forms is also available to be added into any GUI. It provides a tabular area for dropping variable names and adds a description column.

Dynamic titles for frames, group boxes and labels

QESubstitutedLabel, QEFrame, and QEGroupBox are light wrappers around QLabel, QFrame, and QGroupBox respectively which allow you to use macro substitution in the text of what would otherwise be static text.

Viewing PSI's caQtDM MEDM conversion widgets within QEGui

caQtDM, produced by The Paul Scherrer Institute, provides a set of widgets for implementing MEDM like GUIs. The package contains a conversion tool to generate Qt .ui files from MEDM screens, widgets to implement the components within the screens and a GUI viewer to view the converted screens. The converted screens are simply Qt .ui files and can be opened by any application that can open Qt .ui files. caQtDM widgets are not however, in themselves, EPICS aware. The viewer must supply them with EPICS data. The QEGui application can open and view .ui files containing caQtDM widgets, but must use tools provided by the caQtDM libraries to ensure caQtDM widgets are activated and supplied with EPICS data.

Building the caQtDM activation tools into the QEGui application is optional.

Adding GUIs as windows and docks

QEGui can open GUIs in the following ways:

- A new main window
- In place of the GUI in the current main window
- A new tab in the current main window
- A dock of the current main window

New GUIs can be opened by the user from standard buttons in the 'File' menu, from custom menu items, or from buttons within a GUI.

If the 'File' menu is available the user can open a GUI any of the ways listed above.

Custom menu items can be added to open one or more GUIs in a window. For example a single custom menu item can open a new main window containing several tabbed GUIs with several other GUIs in docks.

The GUIs can be opened in the main window where they are tabbed if more than one is specified, or as docks to the main window.

Refer to 'Menu bar and tool button customisation' (page 15) for details on adding custom menubar items to create new windows and docks.

Understanding complex customisation files

QEGui menus and toolbar buttons can be customised to suit the requirements of the GUI solution. XML files are used to define the customisations. To allow common sub-sets of customisations customisation XML files can be included in other customisation. Complex arrangements can be hard to diagnose. To make this task easier, a log is generated as customisation files are loaded at start-up. The log shows what customisation files included what, and in what files each named set of customisations is defined. This log is available in the Help->About dialog as shown in Figure 11. The text in the log can be selected and copied to a text editor to make searching easier.

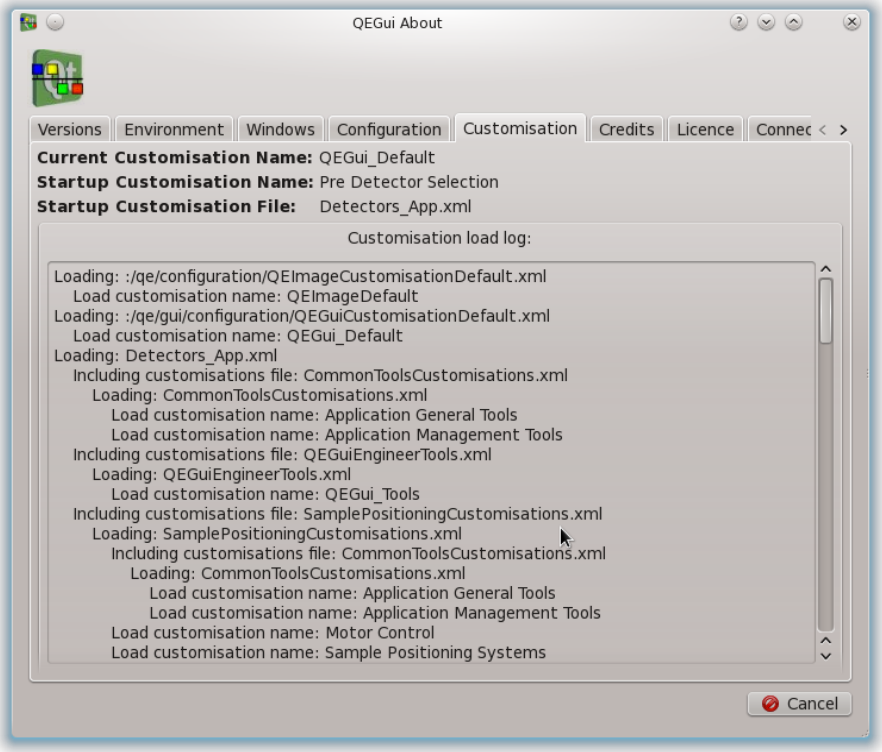

Figure 11 Customisation log

Using a signal to set QE widgets visible (or not)

QWidgets provide a set of slots for showing or hiding the widgets. They include setVisible() and setHidden(). These slots can be used in Designer or programmatically when designing GUIs and applications for a wide range of reasons. The QE framework also shows or hides widgets according to

the user level. This functionality can conflict with the designer's need to show or hide a widget. To manage this QE widgets all have a setManagedVisible() slot.

The QE widgets will be hidden if either hidden through the setManagedVisible() slot or due to an inappropriate user level. Otherwise the QE widgets will be visible.

Calling setManagedVisible() is effective even while the QE widget is hidden due to an inappropriate user level. When the user level is changed to a level allowing the display of the widget, the most recent call to setManagedVisible() will be in effect.

Applying a stylesheet

Like any other application QEGui can be started with a style sheet to customise aspects of the GUIs.

Add `-stylesheet stylesheet.qss` to the qegui command line

Note, if your style sheet specifies colours for a specific widget type, that is what you get, even if the widget is disabled. It may be useful to specify a colour for when the widget is disabled so the normal 'disabled' look still applies like so:

```
QEPushButton                    {color:blue}
QEPushButton:!enabled           {color:grey}
```

You may like to apply different styles to QE push buttons depending on what the button does. For example, red text if the button writes to a variable, green text if the button opens an associated display. But they are all of type QEPushbutton, so how do you distinguish them in the stylesheet file? QE push buttons examine what they are doing and set a dynamic property based on their action. This dynamic property can be referenced in the style sheet like so:

```
QEPushButton[StyleOption="PV"]      {color:red}
QEPushButton[StyleOption="UI"]      {color:green}
QEPushButton                        {color:blue}
```

Refer to '**Error! Reference source not found.**' (page **Error! Bookmark not defined.**) for details.

Setting a window background colour

When editing the palette for a widget you change the 'Window' Color Role hoping that will change the widget's background colour, but nothing happens.

The problem is the widget property 'autoFillBackground' is false by default. Set this to true and the window background colour you selected will be applied.

Setting a background image for a form

Here are three methods for defining a background image for a GUI:

- Base the GUI on a QEFrame. You can then set the QEFrame's 'pixmap' and 'scaledContents' properties to specify a background image in the same way you would on a QLabel.
- Define an image in the widget's 'styleSheet' property like so:

```
QWidget#Form{background: url(austin.png) no-repeat }
```

The above can be varied to tile an image, centre it, set its origin, etc. Refer to Qt's style sheet reference for more details. One thing you don't appear to be able to do using this method is to scale the image to fit the form. To avoid the need for this the form should be set to a suitable fixed size.

- Add a QLabel as a background to the entire form and set its 'pixmap' property. You can also set the 'scaledContents' property to scale the image as QLabel size changes. Other widgets can then be placed over the QLabel as required. This method does not cope well with Qt's layout management. When you specify a layout the layout management will move the QLabel you are using as a background to fit in with your other widgets. Like the previous method, this method works best for fixed sized forms where the control widgets remain where they are placed over the relevant part of the background image.

User Level and Alarm State have no effect while in 'Designer'

QE widgets are 'live' in Designer, which is great – up to a point.

To allow the designer to define the behaviour of QE widget without coming into conflict with the widget modifying its own behaviour due to user level and alarm state, some 'live' functionality is disabled when presented within Designer (including Designer preview).

For example, a control might be set to be hidden or disabled unless User Level is 'Engineer', or a variable may display with a red background if in alarm. It would be awkward to define the default background of a QELabel while the widget is setting its own background to red because of an alarm state. Likewise, having a widget disappear while editing the instant you set it to disappear when in user mode is not helpful.

Generally, QE widgets will be 'live' in Designer in that they will present their core data such as text in a QELabel, or an image in a QEImage, but will not modify properties with which the developer will need to interact, such as background colour.

To easily view full QE widget behaviour while editing in Designer, have the .ui file open in qegui at the same time, starting qegui with the -e (edit) switch. Whenever you want to see what the full behaviour of a GUI will be just save the form. The 'edit' switch will ensure qegui automatically reloads the form when the file is saved.

Starting QEGui where you left off

Every time you exit QEGui it saves the current configuration of windows to a configuration called 'ExitSave'. You can restart QEGui where you left off by using the '-r' restore parameter as follows:

```
qegui –r ExitSave
```

Alternatively, you can select 'Restore Configuration...' from the 'Options' menu and restore the configuration names 'ExitSave'

QE Widgets - General

QE widgets enable the design of control system user interfaces.

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

EPICS enabled standard Qt widgets:

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

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

Any QE widget which can write to one or more EPICS variables will have an indication when the write-access to its variable(s) is not allowed for some reason. If this happens, the cursor will be changed to a "forbidden" cursor from a default cursor when moving within the widget.

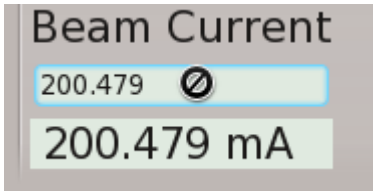

Figure 12 "Write Forbidden" cursor

Control System widgets

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

Common QE Widget properties

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

variableName and variableSubstitutions

All EPICS aware widgets have one or more variable name properties.

Since release 3.7.1, the variableName (and other properties that specify a PV name) property now comprise of a protocol specification and variable name. The allowed protocols are Channel Access and PV Access. The latter is only allowed when using EPICS base 7 or later. The protocols are defined by use of the "ca:// " and "pva:// " prefixes respectively. When no protocol is specified, the Channel Access protocol is used. In later releases, the default may become selectable via an environment variable.

The protocol is derived from the characters preceding the first ":// " within the variableName property. Therefore, if referencing a PV name that actually contains the character sequence ":// ", you must explicitly prefix the PV name with the required protocol specification. Examples:

protocol	PV name	variableName property	Note
Channel Access	BR01IP13:PRESSURE	BR01IP13:PRESSURE	CA selected by default
Channel Access	BR01IP13:PRESSURE	ca://BR01IP13:PRESSURE	
PV Access	BR01IP13:PRESSURE	pva://BR01IP13:PRESSURE	Invalid if base not 7 or later
Invalid	BR01IP13:PRESSURE	xyz://BR01IP13:PRESSURE	Invalid
Channel Access	CRAZY://NAME	ca://CRAZY://NAME	The protocol prefix must be used in this case.

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

Macro substitutions follow the usual EPICS conventions, i.e. they are specified by a comma separated key=value list and called by as $(key).

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

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

Figure 13 shows a form containing a QELabel. The variable name includes macros SECTOR, DEVICE and MONITOR. Default substitutions are provided for MONITOR. This is not adequate to derive a complete variable name.

Figure 14 shows a form using the form from Figure 13 as a sub form. Additional macro definitions for SECTOR and DEVICE are provided with the sub-form file name. When the sub form is loaded, the QELabel in the sub form can now derive a complete variable name (SR01BCM01:CURRENT_MONITOR). While complete, this is not actually functional – the correct sector is SR11.

Figure 15 shows the form from Figure 14 opened by the QEGui application with the following parameters:

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

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

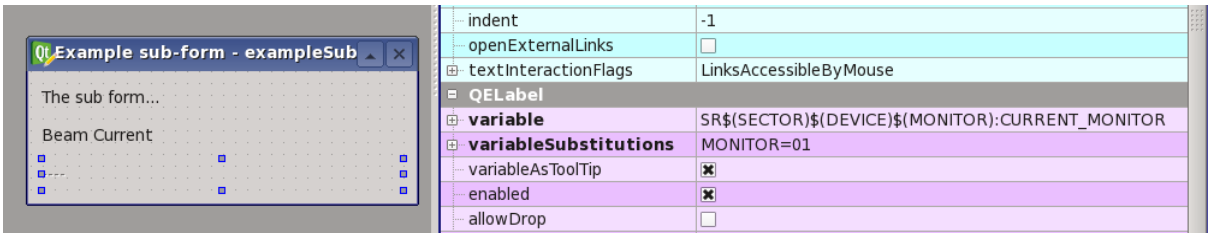

Figure 13 Sub form with macro substitution for part of the variable name

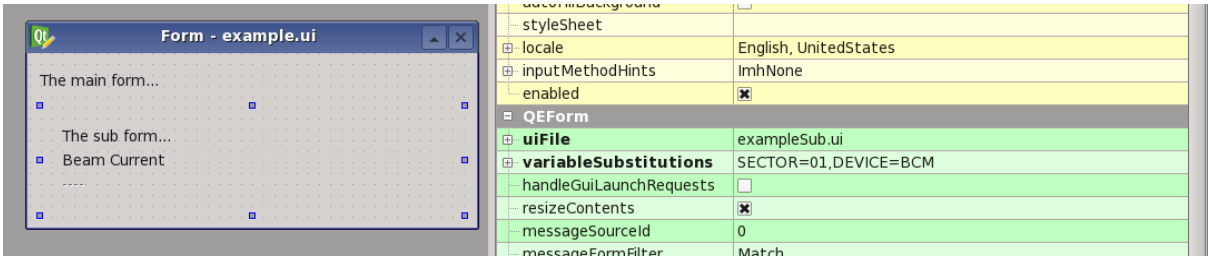

Figure 14 Main form containing sub form with all macro substitutions satisfied (but one is incorrect)

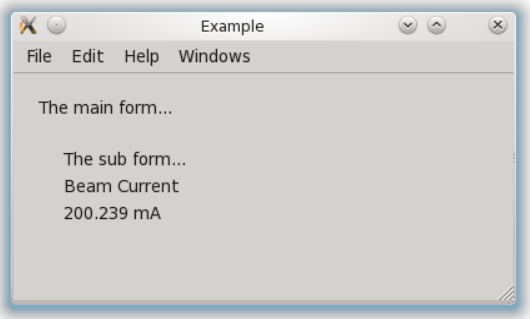

Figure 15 QEGui displaying form and sub forms with all macro substitutions satisfied correctly

elementsRequired

All single variable widgets have an elements required property. This is used to select how many elements of an array PV are read and subscribed for. The default value is 0 which means subscribe for all elements, as opposed to “literally” subscribe for zero elements. This feature useful, for example, to allow a QESimpleShape widget to display the status of a large waveform, without reading the whole waveform.

When this property is defined the widget ensures that: arrayIndex is < elementsRequired.

arrayIndex

All single variable widgets have an array index property. These is used to select a single element from an array PV (such as the .VAL field of a waveform record) used by the widget. The default is 0.

Note: for control widgets, the nominated element of the currently held array data is updated, and then the whole array data is written to the PV, therefore the subscribe property should be set true. Also note there is no mutex/interlock for the case when two independent widgets, maybe hosted on different qegui and/or hosts, write to different elements of an array PV. A potential race condition exists when two writes occur contemporaneously. If such a condition arises, it is a case of last write will 'undo' the update to the other element. If such a situation is unacceptable, then a more robust design should be implement on the PV (EPICS IOC) host. Such a design is beyond the scope of this document.

Note: Previously arrayIndex was considered a string formatting property, but now should be considered a PV name qualifier property. However, QELabel still honours the arrayAction property.

variableAsTooltip

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

subscribe

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

enabled

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

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

allowDrop

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

visible

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

Note, this property is part of the QE framework and is not the QWidget 'visible'property. The following differences apply:

- When this property is false the widget will still be visible in Qt Designer.
- This property remains set or cleared even when the widget is being hidden due to an inappropriate user level. When the user level no longer hides the widget, this property again determines the widget visibility.
- This property can be set through the setManagedVisibility() slot. Using this slot avoids conflict with the operation of the user level mechanism. When this property is set through the setManagedVisibility() slot it remains in the state set, regardless of whether the widget is being hidden due to the user level.

messageSourceId

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

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

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

Refer to Logging(page33) for further details.

userLevelUserStyle, userLevelScientistStyle, userLevelEngineerStyle

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

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

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

Refer to User levels (page 30) for details regarding user levels.

userLevelVisibility

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

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

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

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

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

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

Refer to User levels (page 30) for details regarding user levels.

userLevelEnabled

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

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

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

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

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

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

Refer to User levels (page 30) for details regarding user levels.

displayAlarmStateOption

Widgets may indicate the current alarm state of a variable as well as present the variable data.

Typically the background colour of the widget is set to indicate the alarm state.

This property determines when the alarm state will be presented.

If ‘Always’(default) the widget will always indicate the alarm state of any variable data is displaying, including ‘No Alarm’.

If ‘Never’ the widget will not display the alarm state of any variable data is displaying.

If ‘WhenInAlarm’ the widget will indicate the alarm state of any variable data is displaying when in an alarm state such as EPICS ‘Major’ or ‘Minor’, but will not affect the presentation of the widget when the alarm state is ‘No Alarm’.

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

Note: The default background style colours used for QELabels and other widgets that display textual data are pale-green (NO_ALARM), yellow (MINOR), pale-red (MAJOR, and white (INVALID). For graphical widgets such as QESimpleShape and the like, the default colours are green (NO_ALARM), yellow (MINOR), red (MAJOR, and white (INVALID).
These colours may be re-defined either programmatically (by a bespoke display manager or own plugin) by calling the QECaAlarmInfoColorNamesManager::setStyleColorNames and the QECaAlarmInfoColorNamesManager::setColorNames functions (out of QCaAlarmInfo.h); or be definig the following environment variables: QE_STYLE_COLOR_NAMES and/or QE_COLOR_NAMES.
See the environment variable documentation for details.

String formatting properties

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

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

precision

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

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

useDbPrecision

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

If false, the 'precision' property is used.

leadingZero

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

trailingZeros

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

addUnits

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

localEnumeration

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

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

Format is:

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

Where:

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

Rules are:

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

Examples:

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

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

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

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

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

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

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

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

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

format

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

- Default — Format as best appropriate for the data type.
- Floating — Format as a floating point number
- Integer — Format as an integer

- UnsignedInteger Format as an unsigned integer
- Time Format as a time, source value interpreted as seconds.
- LocalEnumeration Format as a selection from the 'localEnumeration' property

radix

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

notation

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

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

arrayAction

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

- **Append** Interpret each element in the array as an unsigned integer and append string representations of each element from the array with a space in between each. For example, an array of three numbers 10, 11 and 12 will be formatted as '10 11 12'.
- **Ascii** Interpret each element from the array as a character in a string. Trailing zeros and carriage returns are ignored. All other non printing characters except line feeds spaces are translated to '?'. For example an array of three characters 'a' 'b' 'c' will be formatted as 'abc'.
- **Index** Interpret the element selected by setArrayIndex() as an constrained integer. For example, if arrayIndex property is 1, an array of three numbers 10, 11 and 12 will be formatted as '11'.

Appendix A

GNU Free Documentation Licence

GNU Free Documentation License
Version 1.3, 3 November 2008

0. PREAMBLE

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

This License is a kind of "copyleft", which means that derivativeworks of the document must themselves be free in the same sense. Itcomplements the GNU General Public License, which is a copyleftlicense designed for free software.

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

1. APPLICABILITY AND DEFINITIONS

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

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

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

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

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

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

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

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

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

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

the Document to the public.

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

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

2. VERBATIM COPYING

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

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

3. COPYING IN QUANTITY

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

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

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

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

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

4. MODIFICATIONS

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

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

the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section.
You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.

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

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

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

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

5. COMBINING DOCUMENTS

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

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

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

6. COLLECTIONS OF DOCUMENTS

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

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

7. AGGREGATION WITH INDEPENDENT WORKS

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

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

8. TRANSLATION

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

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

9. TERMINATION

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

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

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

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

10. FUTURE REVISIONS OF THIS LICENSE

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

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

11. RELICENSING

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

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