QE 2D Data Visualisation Widgets Specification

Andrew Starritt

3rd November 2020

Contents

Introduction

This document describes in detail the QE2DDataVisualisation widgets (strictly speaking, we should call these widget classes, but the term “widget” is often used through-out this document) which are group of three EPICS aware widgets provided by the EPICS Qt, aka QE, Framework.

This document was created as a separate widget specification document. The main reason for this is ease of maintenance and avoiding editing large and unwieldy word documents.

Description

The QE2DDataVisualisation widgets allow the presentation of 2D data to the user using one of three widgets described below, each presenting the data in a different format. While there are three distinct widgets (as opposed to a single widget type that could morph into each format), each use the same data model by inheriting from a common abstract widget, QEAbstract2DData, also described below.

QEAbstract2DData

Description

The QEAbstract2DData widget is the base class for the QESpectrogram, QESurface and QEWaterfall widgets and itself directly inherits from the QEFrame widget. It could be used as the base class for other widgets providing some 2D data representation.

This class manages the PV data and its organisation into “rows” and “cols”. This widget can handle arrays of any numeric data type. It also provides a number of protected utility methods available to QESpectrogram, QESurface and QEWaterfall to extract the data and data attributes.

Depending on the value of the *dataFormat* property, the data is interpreted as either a 1D data array or a 2D data array.

When the *dataFormat* property is *array2D*, a width must be provided, either via the *widthVariable* (primary source) or via the *dataWidth* property (secondary source). The width defines the number of “columns” and is used to break the source data up into “rows”. The number of “rows” is calculated from the number of elements in the *dataVariable* process variable.

When the *dataFormat* property is *array1D*, a width is not required and any width value is ignored. The number of “columns” is just the number of elements in the *dataVariable* process variable, and the number of “rows” is defined by the *numberOfSets* property. In this mode, the QEAbstract2DData widget accumulates data sets on a FIFO bases (not dis-similar to the compress record in circular buffer mode)

up-to a maximum of *numberOfSets* "rows" of data. The accumulated data is then treated as a 2D array of data.

The widget provides no means to accumulate scalar data; such functionality would have to be provided by an IOC.

The widget triggers a display update each time it receives a *dataVariable* update irrespective of the duration since the last update. Any decimation and/or updating at a fixed interval functionalityis beyond the scope of this widget and must be provided by an IOC.

Properties

dataVariable : QString
This defines the process variable name that provides the data.

widthVariable : QString
This defines the process variable name that provides the data width (optional, as the value can be provided using the *dataWidth* property). A data width is only needed when the data format is *array2D*.

variableSubstitutions : QString
This defines the default substitutions that are applied to both variable names.

dataWidth : int
allowed range: >= 1
default value: 100
This provides a data width values, which is used only if the *widthVariable* is undefined.

dataFormat : enum
allowed values: array1D, array2D
default value: array2D

This controls how the data is interpreted.

numberOfSets : int
allowed range: 1 to 1024
default value: 40

This controls the number of data sets to be accumulated if/when the dataFormat is defined to be *array1D*.

autoScale : bool
default value: true

When set true, the widget will use the minimum and maximum values extract form the PV data sets to scale the widget when rendering the data to the user.

minimum : double
default value: 0.0

This defines the minimum value to be used when rendering the data if *autoScale* is set false. The widget ensures that *maximum* >= *minimum* + 0.001 at all times.

maximum : double

default value: 255.0

This defines the maximum value to be used when rendering the data if autoScale is set false.

QESpectrogram

Description

The QESpectrogram presents the data to the users by assigning a colour (grey scale or false colour) to each element of the 2D array of data. Each element is presented in a grid of "pixels". See the example in Figure 1 below.

When the *dataFormat* is *array1D*, the data accumulates downwards until full, and then scrolls up 1 row at a time as each new data set arrives, i.e. oldest data at the top, newest data at the bottom. If the *orientation* property is set to *Horizontal*, then this becomes: the data accumulates left-to-right until full, and then scrolls left 1 column at a time as each new data set arrives, i.e. oldest data on the left, newest data on the right.

As the mouse moves over the QESpectrogram widget, a readout message is displayed on the status bar showing the row, column and the data element value.

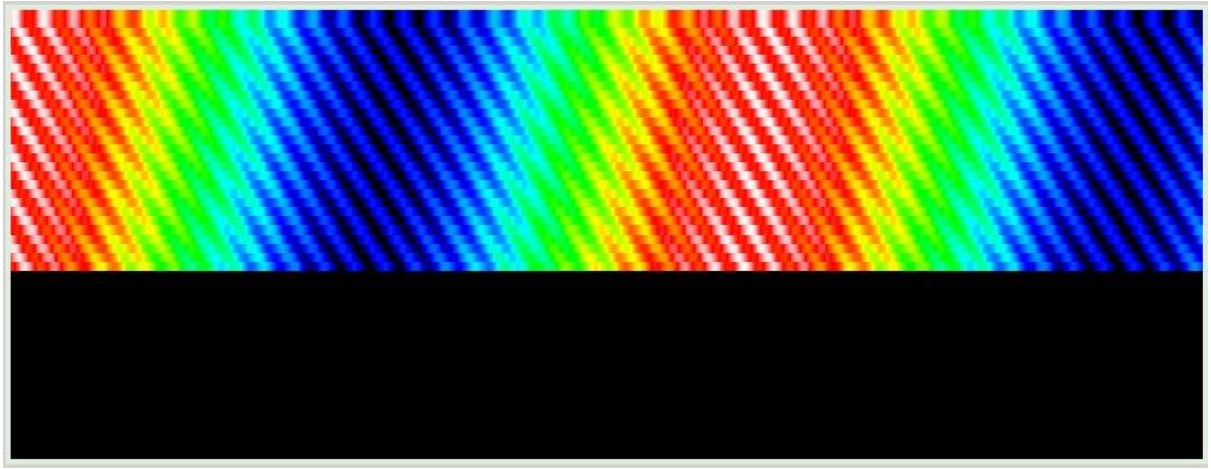

Figure 1 – QESpectrogram – data format: array1D, and is approx. 50% full

Note: this widget may be used to display a mono image; however it would have none of the additional functionality provided by the QEImage widget.

Properties

useFalseColour : bool

default value: true

The data may be displayed using a mono-chrome grey scale (when false) or using false colour when true. The false colour mapping is identical to that used by QEImage when *scaleWrap* property is set to 1.

The input data under-goes a linear mapping such that the minimum data value (as defined or as extracted from the data when *autoScale* set true) is mapped to 0, while the maximum data value (as defined or as extracted from the data when *autoScale* set true) is mapped to 255. Any values outside of this range are clamped to be in the range 0 to 255.

scaleWrap : int

allowed range: 1 to 10
default value: 1

When set to a value more than 1, say 3, the mapping is adjusted such that the range of values is a wrapped sequence of values in the range 0 .. 255, i.e.: 0 .. 224, 32 .. 224, 32 .. 255

This is perhaps more useful when using false colour and allows each colour to be used more than once; and while wrapping up-to to 10 times *is* allowed, more than 2 or 3 times is probably more than enough.

orientation : Qt::Orientation

allowed values: Horizontal, Vertical
default value: Horizontal

This controls the display orientation of the data “rows”.

QESurface

Description

The QESurface widget displays the data using Qt's QE3DSurface functionality. The widget is shown on the form as a push button, which launches a separate window displaying the data when clicked. See the example in Figure 2 below.

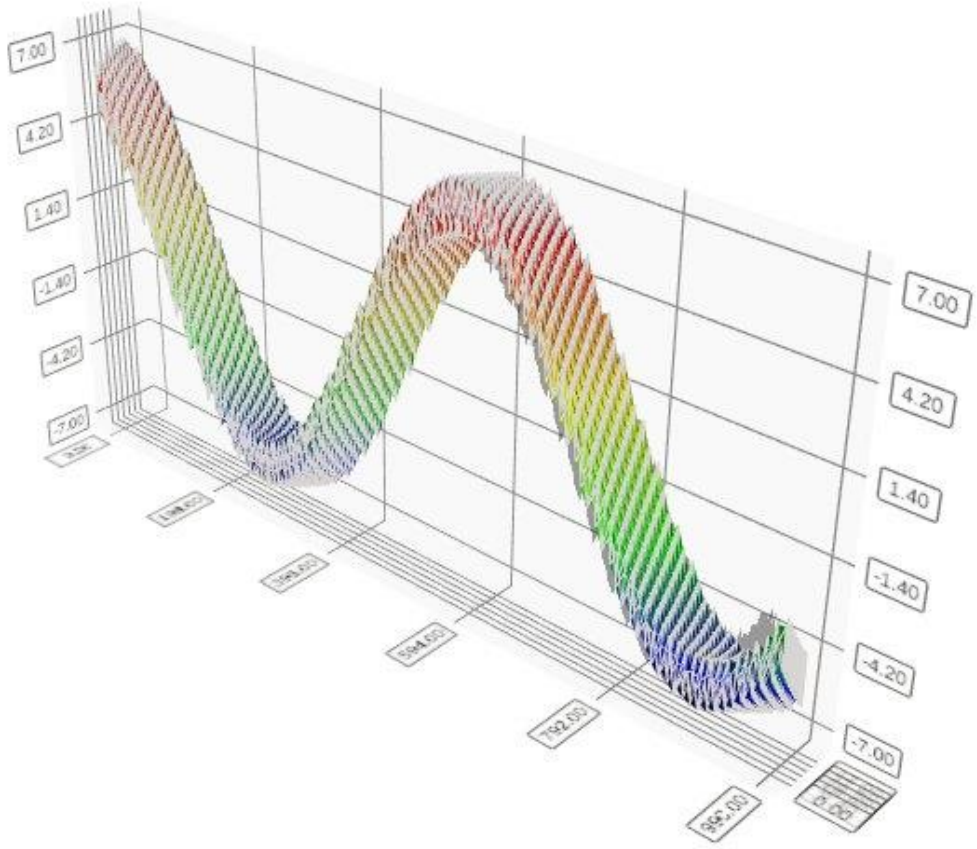

Figure 2 – QESurface – data format: array1D

To do: Figure out how to scale one or both of the horizontal axes.

Note: The QESurface widget is none functional in Qt4, in that it exists and will not break, but it does not display the surface. A debug message is output to this effect when the widget initialises.

Properties

text : QString

default value: "Show 3D Surface" or if using Qt4 "Requires Qt5"

This property defines the text on the button that launches a separate window that shows the 3D surface.

QEWaterfall

Description

The QEWaterfall presents the data to the users as a set of line traces, not dissimilar to the QEPlotter widget, except that each row is offset upwards and leftward. See the example in Figure 3 below.

When the *dataFormat* is *array1D*, the data accumulates downwards until full, and then scrolls backwards 1 row at a time as each new data set arrives, i.e. oldest data at the back, and newest data at the front.

As the mouse moves over the QEWaterfall widget, a readout message is displayed on the status bar showing the row, column and the data element value.

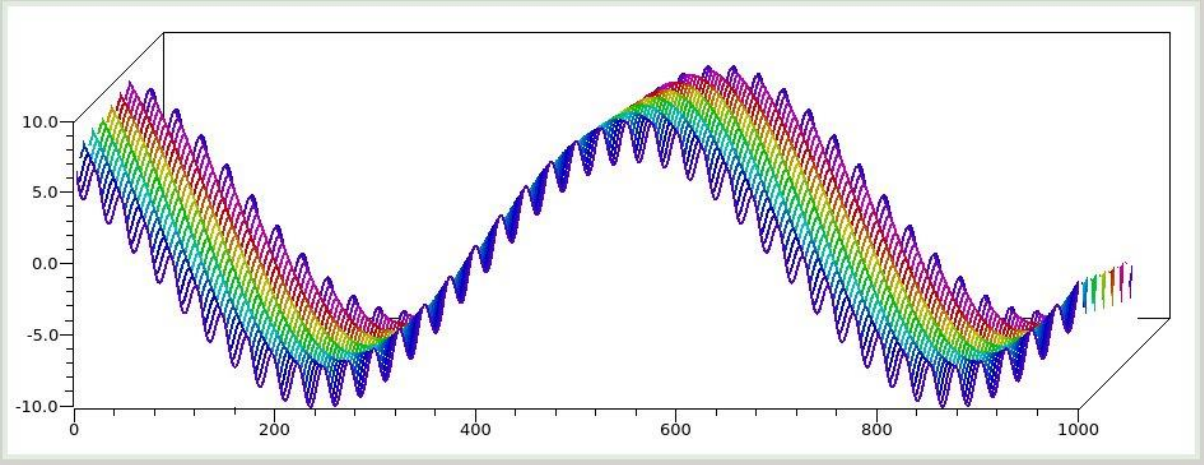

Figure 3: QEWaterfall

Properties

angle : int

allowed range: 0 – 90
default value: 30

This property specified the row/time axis from the vertical in degrees. In the example above, the angle was set to 45.

traceGap : int

allowed range: 1 – 40
default value: 5

This property defines the gap between traces in pixels. The *angle* and *traceGap* properties define the offset applied to each row, namely: *traceGap* * sin (*angle*), *traceGap* * cos (*angle*).

traceWidth : int

allowed range: 0 – 10
default value: 1

This property defines the trace, i.e. pen, width. A value of 0 is best guess, currently always 1.

traceColour : QColor

default value: dark blue

This property defines the colour of the trace.

mutableHue : bool

default value: false

When set true, the hue of the colour is advanced by 12 modulo 360 for each row of data, as illustrated in the example above.