
PyQwt Documentation

Release 5.2.1

Gerard Vermeulen

January 14, 2013

CONTENTS

1	Introduction	1
1.1	NumPy	1
1.2	Qwt	2
1.3	PyQwt with NumPy	2
1.4	Getting help	4
2	Installation	5
2.1	Source Code Installation	5
3	PyQwt Reference Guide	9
3.1	PyQt4.Qwt5	9
3.2	PyQt4.Qwt5.qplot	16
3.3	PyQt4.Qwt5.grace	18
4	Copyright	19
5	Indices and Tables	21
	Python Module Index	23
	Index	25

INTRODUCTION

PyQwt is a set of Python bindings for the [Qwt](#) library featuring fast plotting of Python lists and tuples and the powerful multi-dimensional arrays provided by [NumPy](#), the fundamental package for efficient scientific and engineering computing in Python.¹

1.1 NumPy

The [NumPy](#) package extends Python with multi-dimensional arrays and a complete set of ‘standard’ functions and operators to manipulate the arrays. NumPy turns Python into an ideal language experimental numerical and scientific computing (as powerful as APL, MatLab, IDL and others, but much more elegant).

If you do not have a mathematical background, you can think of a 1-dimensional array as a column in a spreadsheet. The spreadsheet lets you change whole columns element by element in one single statement. In a similar way, NumPy lets you change whole arrays element by element in one single statement as illustrated by the following snippet:

```
>>> import numpy as np
>>> x = np.arange(0.0, 10.0, 3.0)
>>> y = np.sin(x)
>>> x
array([ 0.,  3.,  6.,  9.])
>>> y
array([ 0.          ,  0.14112001, -0.2794155 ,  0.41211849])
>>> x*x
array([ 0.,  9., 36., 81.] )
```

The statement:

```
>>> np.arange(0.0, 10.0, 3.0)
```

returns a NumPy array of 4 equidistant points from 0 to 9 inclusive:

```
array([ 0.,  3.,  6.,  9.] )
```

The statements `y = np.sin(x)` and `x*x` show that NumPy arrays are manipulated element by element. All this has been coded in C, for a manifold speedup with respect to pure Python.

You can think of a 2-dimension array as a spreadsheet: in both cases you can operate on blocks, columns, rows, slices of columns, slices of rows or individual elements.

Want to learn more? Look at the [Tentative NumPy Tutorial](#) for a tutorial or at the [Guide to NumPy](#) for an advanced book.

¹ The older numerical Python extension packages, [numarray](#) and [Numeric](#) are deprecated.

1.2 Qwt

Qwt is a C++ library based on the [Qt GUI framework](#). The Qwt library contains widgets useful for writing technical, scientific, and financial programs. It includes the following widgets:

QwtCompass a very fancy QDial-like widget to display and control a direction.

QwtCounter a QSpinBox-like widget to display and control a bounded floating point value.

QwtDial a QDial-like widget to display and control a floating point value.

QwtKnob a potentiometer-like widget to display and control a bounded floating point value.

QwtPlot a widget to plot data in two dimensions.

QwtSlider a QSlider-like widget to display and control a bounded floating point value.

QwtThermo a thermometer-like widget to display a floating point value.

QwtWheel a wheel-like widget with its axis parallel to the computer screen to control a floating point value over a very large range in very small steps.

See the [Qwt manual](#) for a complete overview of the Qwt library.

1.3 PyQt with NumPy

PyQt is mostly used to write graphical user interface applications. However, the following snippet shows how to use PyQt in combination with NumPy from the command line interpreter. Line by line explanations follow the snippet:

```
>>> import numpy as np
>>> from PyQt4.Qt import *
>>> from PyQt4.Qwt5 import *
>>> from PyQt4.Qwt5.qplt import *
>>> application = QApplication([])
>>> x = np.arange(-2*np.pi, 2*np.pi, 0.01)
>>> p = Plot(
...     Curve(x, np.cos(x), Pen(Magenta, 2), "cos(x)"),
...     Curve(x, np.exp(x), Pen(Red), "exp(x)", Y2),
...     Axis(Y2, Log),
...     "PyQt using Qwt-%s -- http://qwt.sf.net" % QWT_VERSION_STR)
>>> QPixmap.grabWidget(p).save('cli-plot-1.png', 'PNG')
True
>>> x = x[0:-1:10]
>>> p.plot(
...     Curve(x, np.cos(x-np.pi/4), Symbol(Circle, Yellow), "circle"),
...     Curve(x, np.cos(x+np.pi/4), Pen(Blue), Symbol(Square, Cyan), "square"))
>>> QPixmap.grabWidget(p).save('cli-plot-2.png', 'PNG')
True
```

The statements:

```
>>> import numpy as np
>>> from PyQt4.Qt import *
>>> from PyQt4.Qwt5 import *
>>> from PyQt4.Qwt5.qplt import *
```

import numpy, PyQt4, Qwt5 and qplt. The statement:

```
>>> application = QApplication([])
```

initializes and starts the Qt library so that it handles mouse movements, mouse button presses, and keyboard key presses. The statement:

```
>>> x = np.arange(-2*np.pi, 2*np.pi, 0.01)
```

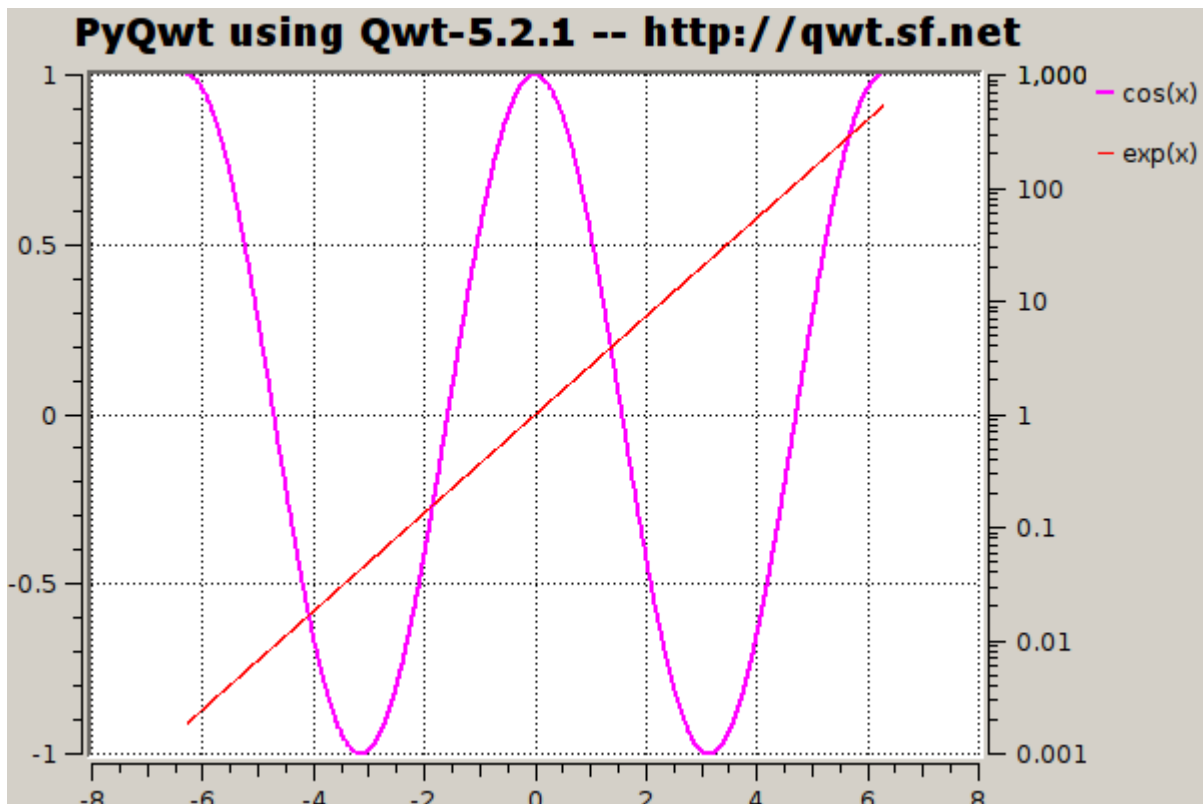
creates an array with elements increasing from -2π to 2π in steps of 0.01. The statement:

```
>>> p = Plot(
...     Curve(x, np.cos(x), Pen(Magenta, 2), "cos(x)"),
...     Curve(x, np.exp(x), Pen(Red), "exp(x)", Y2),
...     Axis(Y2, Log),
...     "PyQwt using Qwt-%s -- http://qwt.sf.net" % QWT_VERSION_STR)
```

creates and shows a plot widget with two curves and an additional right vertical logarithmic axis. The statement:

```
>>> QPixmap.grabWidget(p).save('cli-plot-1.png', 'PNG')
True
```

takes a snapshot of the plot widget and saves it into a file:



The statement:

```
>>> x = x[0:-1:10]
```

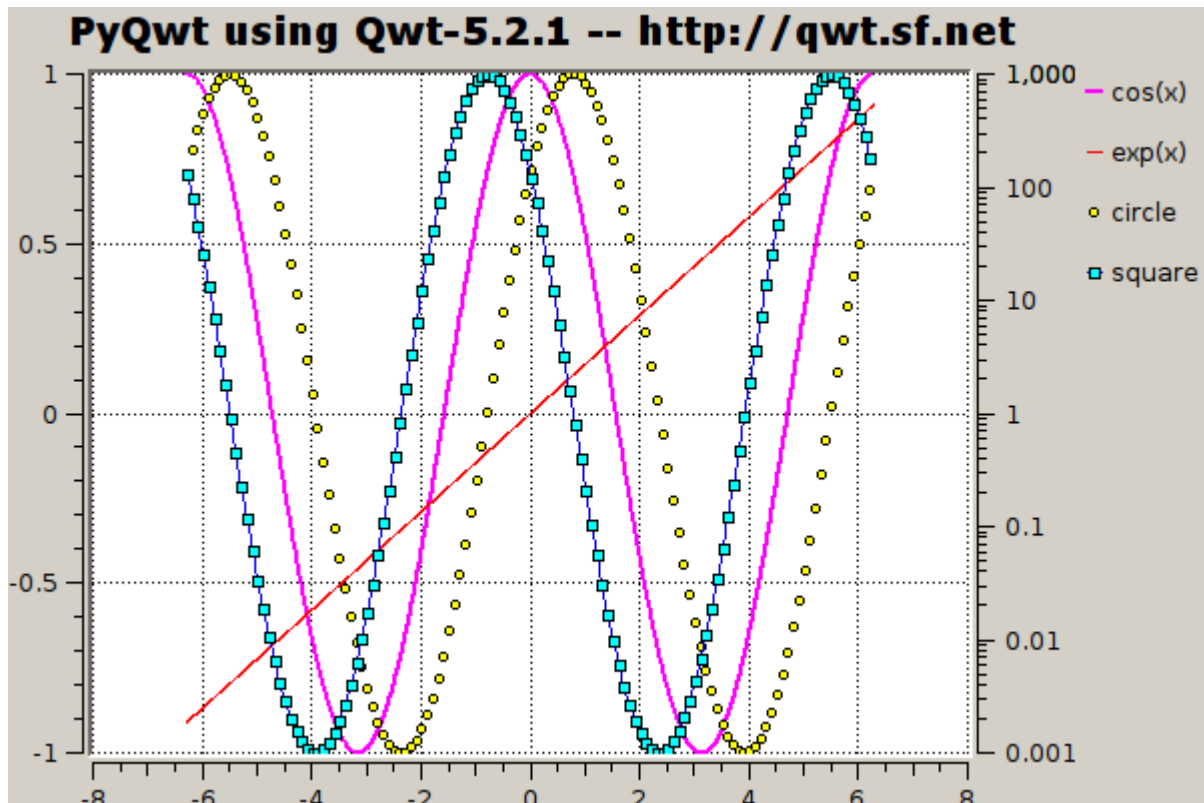
creates a new array from the old one by selecting every tenth element start from the index 0. The statement:

```
>>> p.plot(
...     Curve(x, np.cos(x-np.pi/4), Symbol(Circle, Yellow), "circle"),
...     Curve(x, np.cos(x+np.pi/4), Pen(Blue), Symbol(Square, Cyan),
...     "square"))
```

plots two new curves on the widget using the new array. The statement:

```
>>> QPixmap.grabWidget(p).save('cli-plot-2.png', 'PNG')
True
```

takes a snapshot of the plot widget and saves it into a file:



1.4 Getting help

PyQwt has a low volume mailing list to answer questions on installation problems and how to use the more advanced features. In particular, many of the more advanced examples using object oriented programming have been written to answer questions. Most questions help to improve PyQwt!

Please, [subscribe](#) to the mailing list before posting on the [mailing list](#).

The mailing list is a subscribers only list and mail from non-subscribers is deferred to filter spam (more than 95 % of the mail by non-subscribers is spam and mail by non-subscribers is rejected).

The mailing list is configured to guarantee anonymity as much as possible.

INSTALLATION

2.1 Source Code Installation

2.1.1 Build Prerequisites

Recommended build prerequisites for PyQwt-5.2.1 are:

1. [Python](#), version 3.1.x, 2.7.x, 2.6.x, and 2.5.x are supported.
2. [Qt](#), version 4.6.x, 4.5.x, 4.4.x, and 3.3.x are supported.
3. [SIP](#), version 4.10.x is supported.
4. [PyQt](#) for Mac OS X, Windows, and/or X11, version 4.7.x, and 3.18.x are supported.
5. optionally [NumPy](#), version 1.4.x, 1.3.x, and 1.2.x are supported. You will need to check out NumPy from subversion when you want to use NumPy with Python-3.x.
6. optionally [Qwt](#), version 5.2.x, 5.1.x, and 5.0.x are supported.

The source package [PyQwt-5.2.1.tar.gz](#) contains a snapshot of the Qwt-5.2 subversion bug fix branch which may fix some bugs in Qwt-5.2.1. I recommend to compile and link the bug fix branch statically into PyQwt.

To exploit the full power of PyQwt, you should install at least one of the numerical Python extensions:

- [NumPy](#)
- [numarray](#)
- [Numeric](#)

and built PyQwt with support for the numerical Python extension(s) of your choice. However, only NumPy is actively developed and numarray and Numeric are deprecated.

PyQwt-5.2.1 and recent versions of the numerical Python extensions support the [N-D array interface](#) protocol. Therefore, PyQwt supports those extensions, even if they have not been installed when PyQwt has been built. In this case, the functionality is somewhat reduced, since conversion from an QImage to a Numerical Python array is not supported.

2.1.2 Installation

The installation procedure consists of three steps:

1. Unpack PyQwt-5.2.1.tar.gz.
2. Invoke the following commands to build PyQwt-5.2.1 for Qt-4:

```
cd PyQwt-5.2.1
cd configure
python configure.py -Q ../qwt-5.2
```

```
make
make install
```

or invoke the commands to build PyQt-5.2.1 for Qt-3:

```
cd PyQt-5.2.1
cd configure
python configure.py -3 -Q ../qwt-5.2
make
make install
```

This assumes that the correct Python interpreter is on your path. Replace **make** by **nmake**, if you use Microsoft Visual C++. The commands build PyQt against the included Qwt subversion snapshot and install PyQt. Test the installation by playing with the example programs.

3. Fine tune (optional):

- to use a Qwt library already installed on your system invoke commands similar to:

```
python configure.py -I/usr/include/qwt -lqwt
make
make install
```

where the Qwt header files are assumed to be installed in `/usr/include/qwt`.

If the linker fails to find the qwt library, add:

```
-L /directory/with/qwt/library
```

to the **configure.py** options.

The `configure.py` script takes many options. The command:

```
python configure.py -h
```

displays a full list of the available options

Usage: `python configure.py [options]`

Each option takes at most one argument, but some options accumulate arguments when repeated. For example, invoke:

```
python configure.py -I . -I ..
```

to search the current `*`and`*` parent directories for headers.

Options:

`-h, --help` show this help message and exit

Common options:

```
-3, --qt3          build for Qt3 and PyQt [default Qt4]
-4, --qt4          build for Qt4 and PyQt4 [default Qt4]
-Q /sources/of/qwt, --qwt-sources=/sources/of/qwt
                  compile and link the Qwt source files in /sources/of/qwt statically i
-I /usr/lib/qt3/include/qwt, --extra-include-dirs=/usr/lib/qt3/include/qwt
                  add an extra directory to search for headers (the compiler must be ab
                  headers without the -Q option)
-L /usr/lib/qt3/lib, --extra-lib-dirs=/usr/lib/qt3/lib
                  add an extra directory to search for libraries (the linker must be ab
                  library without the -Q option)
-j N, --jobs=N     concatenate the SIP generated code into N files [default 1 per class]
                  running simultaneous jobs on multiprocessor systems)
```

Make options:

```
--debug          enable debugging symbols [default disabled]
--extra-cflags=EXTRA_CFLAG
```

```
        add an extra C compiler flag
--extra-cxxflags=EXTRA_CXXFLAG
        add an extra C++ compiler flag
-D HAS_EXTRA_SENSORY_PERCEPTION, --extra-defines=HAS_EXTRA_SENSORY_PERCEPTION
        add an extra preprocessor definition
-l extra_sensory_perception, --extra-libs=extra_sensory_perception
        add an extra library
--extra-lflags=EXTRA_LFLAG
        add an extra linker flag

SIP options:
-x EXTRA_SENSORY_PERCEPTION, --excluded-features=EXTRA_SENSORY_PERCEPTION
        add a feature for SIP to exclude (normally one of the features in sip
-t EXTRA_SENSORY_PERCEPTION, --timelines=EXTRA_SENSORY_PERCEPTION
        add a timeline option for SIP (normally one of the timeline options i
--sip-include-dirs=SIP_INCLUDE_DIR
        add an extra directory for SIP to search
--trace
        enable trace of the execution of the bindings [default disabled]

Detection options:
--disable-numarray  disable detection and use of numarray [default enabled]
--disable-numeric   disable detection and use of Numeric [default enabled]
--disable-numpy     disable detection and use of NumPy [default enabled]

Install options:
--module-install-path=MODULE_INSTALL_PATH
        specify the install directory for the Python modules
```

2.1.3 Troubleshooting and getting help

1. Check whether all development packages have been installed when **make** produces lots of errors on Linux.
2. If you fail to install PyQwt, unpack PyQwt-5.2.1.tar.gz into a clean directory and create two log files containing *stdout* *and* *stderr*:

```
python configure.py --your --options 2>&1 >configure.log
make 2>&1 >make.log
```

Send the log files to the [mailing list](#) after [subscribing](#) to the mailing list, because the mailing list is for subscribers only, see [Getting help](#).

PYQWT REFERENCE GUIDE

3.1 PyQt4.Qwt5

The reference should be used in conjunction with the [Qwt manual](#). Only the differences specific to the Python bindings are documented here.

In this chapter, **is not yet implemented** implies that the feature can be easily implemented if needed, **is not implemented** implies that the feature is not easily implemented, and **is not Pythonic** implies that the feature will not be implemented because it violates the Python philosophy (e.g. may use dangling pointers).

If a class is described as being **is fully implemented** then all non-private member functions and all public class variables have been implemented.

Undocumented classes have not yet been implemented or are still experimental.

3.1.1 Class reference

class QwtAbstractScale
is fully implemented.

class QwtAbstractScaleDraw
is fully implemented.

class QwtAbstractSlider
is fully implemented.

class QwtAlphaColorMap
is fully implemented.

class QwtArrayData
is fully implemented.

class QwtArrayDouble
is fully implemented. See *Template reference*.

class QwtArrayInt
is fully implemented. See *Template reference*.

class QwtArrayQwtDoubleInterval
is fully implemented. See *Template reference*.

class QwtArrayQwtDoublePoint
is fully implemented. See *Template reference*.

class QwtArrowButton
is fully implemented.

class QwtClipper
is fully implemented, but only available when PyQwt wraps Qwt-5.1.x.

class QwtColorMap
is fully implemented.

class QwtCompass
is fully implemented.

class QwtCompassMagnetNeedle
is fully implemented.

class QwtCompassRose
is fully implemented.

class QwtCompassWindArrow
is fully implemented.

class QwtCounter
is fully implemented.

class QwtCurveFitter
is fully implemented.

class QwtData
is fully implemented.

class QwtDial
is fully implemented.

class QwtDialNeedle
is fully implemented.

class QwtDialScaleDraw
is fully implemented.

class QwtDialSimpleNeedle
is fully implemented.

class QwtDoubleInterval
is fully implemented.

class QwtDoublePoint
is fully implemented, but only available when PyQt wraps Qt-3. When PyQt wraps Qt-4, replace this class with *QPointF* except in signals. For example, clicking in the canvas of the plot displayed by the following program:

```
#!/usr/bin/env python

import sys
from PyQt4 import Qt
import PyQt4.Qwt5 as Qwt

def aSlot(aQPointF):
    print 'aSlot gets:', aQPointF

# aSlot()

def make():
    demo = Qwt.QwtPlot()
    picker = Qwt.QwtPlotPicker(Qwt.QwtPlot.xBottom,
                              Qwt.QwtPlot.yLeft,
                              Qwt.QwtPicker.PointSelection,
                              Qwt.QwtPlotPicker.CrossRubberBand,
                              Qwt.QwtPicker.AlwaysOn,
                              demo.canvas())

    picker.connect(
        picker, Qt.SIGNAL('selected(const QwtDoublePoint&)'), aSlot)
    return demo
```

```
# make()

def main(args):
    app = Qt.QApplication(args)
    demo = make()
    demo.show()
    sys.exit(app.exec_())

# main()

if __name__ == '__main__':
    main(sys.argv)

# Local Variables: ***
# mode: python ***
# End: ***
```

shows that the signal returns an object of type *QPointF*:

```
aSlot gets: <PyQt4.QtCore.QPointF object at 0x2aaaaf73be20>
```

class QwtDoubleRange

is fully implemented.

class QwtDoubleRect

is fully implemented, but only available when PyQt wraps Qt-3.

When PyQt wraps Qt-4, replace this class with *QRectF* except in signals: see [QwtDoublePoint](#).

class QwtDoubleSize

is fully implemented, but only available when PyQt wraps Qt-3.

When PyQt wraps Qt-4, replace this class with *QSizeF* except in signals: see [QwtDoublePoint](#).

class QwtDynGridLayout

is fully implemented.

class QwtEventPattern

is fully implemented.

class QwtIntervalData

is fully implemented.

class QwtKnob

is fully implemented.

class QwtLegend

is fully implemented.

class QwtLegendItem

is fully implemented.

class QwtLegendItemManager

is fully implemented, but only available when PyQwt wraps Qwt-5.1.x.

class QwtLinearColorMap

is fully implemented.

class QwtLinearScaleEngine

is fully implemented.

class QwtLog10ScaleEngine

is fully implemented.

class QwtLegendMagnifier

is fully implemented, but only available when PyQwt wraps Qwt-5.1.x.

class QwtMetricsMap
is fully implemented.

class QwtPaintBuffer
is fully implemented when PyQt wraps Qt-3.

class QwtPainter
is fully implemented.

class QwtPanner
is fully implemented.

class QwtPicker
is fully implemented.

class QwtPickerClickPointMachine
is fully implemented.

class QwtPickerClickRectMachine
is fully implemented.

class QwtPickerDragPointMachine
is fully implemented.

class QwtPickerDragRectMachine
is fully implemented.

class QwtPickerMachine
is fully implemented.

class QwtPickerPolygonMachine
is fully implemented.

class QwtPlainTextEngine
is fully implemented.

class QwtPlot
is fully implemented, but:

`void QwtPlot::print (QPrinter& printer, const QwtPlotPrintFilter& filter)`
is implemented as:

```
plot.print_(printer, filter)
```

`void QwtPlot::print (QPainter* painter, const QRect& rect, const QwtPlotPrintFilter& filter)`
is implemented as:

```
plot.print_(painter, rect, filter)
```

class QwtPlotCanvas
is fully implemented.

class QwtPlotCurve
is fully implemented, but:

`void QwtPlotCurve::setData (double* x, double* y, int size)`
is implemented as:

```
curve.setData(x, y)
```

where *x* and *y* can be any combination of lists, tuples and Numerical Python arrays. The data is copied to C++ data types.

`void QwtPlotCurve::setRawData (double* x, double* y, int size)`
is not Pythonic.

class QwtPlotDict
is fully implemented. FIXME: is the auto delete feature dangerous?

class QwtPlotGrid
is fully implemented.

class QwtPlotItem
is fully implemented.

class QwtPlotLayout
is fully implemented.

class QwtPlotMagnifier
is fully implemented.

class QwtPlotMarker
is fully implemented.

class QwtPlotPanner
is fully implemented.

class QwtPlotPicker
is fully implemented, but:

`QwtText QwtPlotPicker::trackerText (QwtDoublePoint& point)`
is implemented as:

```
qwtText = plotPicker.trackerText (point)
```

where *point* is a `QwtDoublePoint` when PyQt wraps Qt-3 or a `QPointF` when PyQt wraps Qt-4.

class QwtPlotPrintFilter
is fully implemented.

class QwtPlotRasterItem
is fully implemented.

class QwtPlotScaleItem
is fully implemented, but only available when PyQwt wraps Qwt-5.1.x.

class QwtPlotSpectrogram
The protected member functions

```
QMap<double, QPolygonF> QwtPlotSpectrogram::renderContourLines (const Qwt-  
DoubleRect&,   
const   
QSize&) const
```

and

```
void QwtPlotSpectrogram::drawContourLines (QPainter*, const QwtScaleMap&, const   
QwtScaleMap&, const QMap<double,   
QPolygonF>&) const
```

are not yet implemented.

class QwtPlotSvgItem
is fully implemented.

class QwtPlotZoomer
is fully implemented.

class QwtPolygon
When PyQt wraps Qt-3, replace this class with `QPointArray` except in signals: see `QwtDoublePoint`.

When PyQt has been built for Qt-4, replace this class with `QPolygon` except in signals: see `QwtDoublePoint`.

class QwtPolygonFData
is fully implemented.

class QwtRasterData
is fully implemented.

class `QwtRect`
is fully implemented.

class `QwtRichTextEngine`
is fully implemented.

class `QwtRoundScaleDraw`
is fully implemented.

class `QwtScaleArithmetic`
is fully implemented.

class `QwtScaleDiv`

`QwtScaleDiv::QwtScaleDiv (const QwtDoubleInterval&, QwtValueList*)`
is implemented as:

```
scaleDiv = QwtScaleDiv(  
    qwtDoubleInterval, majorTicks, mediumTicks, minorTicks)
```

`QwtScaleDiv::QwtScaleDiv (double, double, QwtTickList*)`
is implemented as:

```
scaleDiv = QwtScaleDiv(  
    lower, upper, majorTicks, mediumTicks, minorTicks)
```

class `QwtScaleDraw`
is fully implemented.

class `QwtScaleEngine`
is fully implemented.

class `QwtScaleMap`
is fully implemented.

`QwtScaleMap::QwtScaleMap (int, int, double, double)`
does not exist in C++, but is provided by PyQt.

class `QwtScaleTransformation`
is fully implemented.

class `QwtScaleWidget`
is fully implemented.

class `QwtSimpleCompassRose`
is fully implemented.

class `QwtSlider`
is fully implemented.

class `QwtSpline`
is fully implemented.

class `QwtSplineCurveFitter`
is fully implemented.

class `QwtSymbol`
is fully implemented.

class `QwtText`
is fully implemented.

class `QwtTextEngine`
is fully implemented.

class `QwtTextLabel`
is fully implemented.

class `QwtThermo`
is fully implemented.

class `QwtWheel`
is fully implemented.

3.1.2 Function reference

`toImage` (*array*)
Convert *array* to a *QImage*, where *array* must be a 2D NumPy, ndarray, or Numeric array containing data of type uint8 or uint32.

`toNumarray` (*image*)
Convert *image* to a 2D ndarray array, where *image* must be a *QImage* with depth 8 or 32. The resulting 2D ndarray array contains data of type uint8 or uint32.

`toNumeric` (*image*)
Convert *image* to a 2D Numeric array, where *image* must be a *QImage* of depth 8 or 32. The resulting 2D Numeric array contains data of type uint8 or uint32.

`toNumpy` (*image*)
Convert *image* to a 2D NumPy array, where *image* must be a *QImage* of depth 8 or 32. The resulting 2D NumPy array contains data of type uint8 or uint32.

`to_na_array` (*image*)
Deprecated. Use `toNumarray()`.

`to_np_array` (*image*)
Deprecated. Use `toNumeric()`.

3.1.3 Template reference

PyQwt has a partial interface to the following *QwtArray<T>* templates:

1. `QwtArrayDouble` for *QwtArray<double>*
2. `QwtArrayInt` for *QwtArray<int>*
3. `QwtArrayQwtDoubleInterval` for *QwtArray<QwtDoubleInterval>*
4. `QwtArrayQwtDoublePoint` for *QwtArray<QwtDoublePoint>* when PyQt has been built against Qt-3 or for *QwtArray<QPointF>* when PyQt has been built against Qt-4.

Those classes have at least 3 constructors, taking *QwtArrayDouble* as an example:

1. `array = QwtArrayDouble()`
2. `array = QwtArrayDouble(int)`
3. `array = QwtArrayDouble(otherArray)`

QwtArrayDouble and *QwtArrayInt* have also a constructor which takes a sequence of items convertible to a C++ double and a C++ long. For instance:

- `array = QwtArrayDouble(numpy.array([0.0, 1.0]))`
- `array = QwtArrayInt(numpy.array([0, 1]))`

All those classes have 16 member functions, taking *QwtArrayDouble* as example:

1. `array = array.assign(otherArray)`
2. `item = array.at(index)`
3. `index = array.bsearch(item)`
4. `index = contains(item)`

```
5. array = otherArray.copy()
6. result = array.count()
7. array.detach()
8. array = array.duplicate(otherArray)
9. bool = array.fill(item, index=-1)
10. index = array.find(item, index=0)
11. bool = array.isEmpty()
12. bool = array.isNull()
13. bool = array.resize(index)
14. result = array.size()
15. array.sort()
16. bool = array.truncate(index)
```

Iterators are not yet implemented. However, the implementation of the special class methods `__getitem__`, `__len__` and `__setitem__` let you use those classes almost as a sequence. For instance:

```
>>> from PyQt4.Qt5 import *
>>> import numpy as np
>>> a = QwtArrayDouble(np.arange(10, 20, 4))
>>> for i in a:                                     # thanks to __getitem__
...     print i
...
10.0
14.0
18.0
>>> for i in range(len(a)):                         # thanks to __len__
...     print a[i]                                 # thanks to __getitem__
...
10.0
14.0
18.0
>>> for i in range(len(a)):                         # thanks to __len__
...     a[i] = 10+3*i                             # thanks to __setitem__
...
>>> for i in a:                                     # thanks to __getitem__
...     print i
...
10.0
13.0
16.0
```

3.2 PyQt4.Qt5.qplot

Provides a command line interpreter friendly layer over *QwtPlot*. An example of its use is:

```
>>> import numpy as np
>>> from PyQt4.Qt import *
>>> from PyQt4.Qt5 import *
>>> from PyQt4.Qt5.qplot import *
>>> application = QApplication([])
>>> x = np.arange(-2*np.pi, 2*np.pi, 0.01)
>>> p = Plot(
...     Curve(x, np.cos(x), Pen(Magenta, 2), 'cos(x)'),
...     Curve(x, np.exp(x), Pen(Red), 'exp(x)', Y2),
```

```

... Axis(Y2, Log),
... 'PyQwt using Qwt-%s -- http://qwt.sf.net' % QWT_VERSION_STR)
>>> QPixmap.grabWidget(p).save('cli-plot-1.png', 'PNG')
True
>>> x = x[0:-1:10]
>>> p.plot(
... Curve(x, np.cos(x-np.pi/4), Symbol(Circle, Yellow), 'circle'),
... Curve(x, np.cos(x+np.pi/4), Pen(Blue), Symbol(Square, Cyan), 'square'))
>>> QPixmap.grabWidget(p).save('cli-plot-2.png', 'PNG')
True

```

class Axis (*rest)

A command line interpreter friendly class.

The interpretation of the **rest* parameters is type dependent:

- *QwtPlot.Axis*: sets the orientation of the axis.
- *QwtScaleEngine*: sets the axis type (Lin or Log).
- *int* : sets the attributes of the axis.
- *string* or *QString*: sets the title of the axis.

class Curve (x, y, *rest)

A command line friendly layer over *QwtPlotCurve*.

Parameters:

- *x*: sequence of numbers
- *y*: sequence of numbers

The interpretation of the **rest* parameters is type dependent:

- *Axis*: attaches an axis to the curve.
- *Pen*: sets the pen to connect the data points.
- *Symbol*: sets the symbol to draw the data points.
- *str*, *QString*, or *QwtText*: sets the curve title.

class IPlot (*rest)

A QMainWindow widget with a Plot widget as central widget. It provides:

- 1.a toolbar for printing and piping into Grace.
- 2.a legend with control to toggle curves between hidden and shown.
- 3.mouse tracking to display the coordinates in the status bar.
- 4.an infinite stack of zoom regions.

The interpretation of the *rest* parameters is type dependent:

- *Axis*: enables the axis.
- *Curve*: adds a curve.
- *str* or *QString*: sets the title.
- *int*: sets a set of mouse events to the zoomer actions.
- (*int*, *int*): sets the size.

class Pen (*rest)

A command line friendly layer over *QPen*.

The interpretation of the **rest* parameters is type dependent:

- *Qt.PenStyle*: sets the pen style.

- *QColor* or *Qt.GlobalColor*: sets the pen color.
- *int*: sets the pen width.

class Plot (*rest)

A command line interpreter friendly layer over *QwtPlot*.

The interpretation of the *rest parameters is type dependent:

- *Axis*: enables the axis.
- *Curve*: adds a curve.
- *str* or *QString*: sets the title.
- *int*: sets a set of mouse events to the zoomer actions.
- (*int*, *int*): sets the size.
- *QWidget*: sets the parent widget

clearZoomStack ()

Force autoscaling and clear the zoom stack

formatCoordinates (x, y)

Format mouse coordinates as real world plot coordinates.

gracePlot (saveall=' ', pause=0.2)

Clone the plot into Grace for very high quality hard copy output.

Know bug: Grace does not scale the data correctly when Grace cannot keep up with gracePlot. This happens when it takes too long to load Grace in memory (exit the Grace process and try again) or when 'pause' is too short.

plot (*rest)

Plot additional curves and/or axes.

The interpretation of the *rest parameters is type dependent:

- *Axis*: enables the axis.
- *Curve*: adds a curve.

setZoomerMouseEventSet (index)

Attach the *Qwt.QwtPlotZoomer* actions to a set of mouse events.

toggleVisibility (plotItem)

Toggle the visibility of a plot item

class Symbol (*rest)

A command line friendly layer over *QwtSymbol*.

The interpretation of the *rest parameters is type dependent:

- *QColor* or *Qt.GlobalColor*: sets the symbol fill color.
- *QwtSymbol.Style*: sets symbol style.
- *int*: sets the symbol size.

3.3 PyQt4.Qwt5.grace

class GraceProcess (debug=None)

Provides a simple interface to a Grace subprocess.

COPYRIGHT

Copyright © 2001-2010 Gerard Vermeulen

Copyright © 2000 Mark Colclough

PyQwt is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

PyQwt is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with PyQwt; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

In addition, as a special exception, Gerard Vermeulen gives permission to link PyQwt dynamically with non-free versions of Qt and PyQt, and to distribute PyQwt in this form, provided that equally powerful versions of Qt and PyQt have been released under the terms of the GNU General Public License.

If PyQwt is dynamically linked with non-free versions of Qt and PyQt, PyQwt becomes a free plug-in for a non-free program.

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

p

`PyQt4.Qt5`, 9

`PyQt4.Qt5.grace`, 18

`PyQt4.Qt5.qplt`, 16

INDEX

A

Axis (class in PyQt4.Qwt5.qplot), 17

C

clearZoomStack() (Plot method), 18

Curve (class in PyQt4.Qwt5.qplot), 17

F

formatCoordinates() (Plot method), 18

G

gracePlot() (Plot method), 18

GraceProcess (class in PyQt4.Qwt5.grace), 18

I

IPlot (class in PyQt4.Qwt5.qplot), 17

P

Pen (class in PyQt4.Qwt5.qplot), 17

Plot (class in PyQt4.Qwt5.qplot), 18

plot() (Plot method), 18

PyQt4.Qwt5 (module), 9

PyQt4.Qwt5.grace (module), 18

PyQt4.Qwt5.qplot (module), 16

Q

QwtAbstractScale (class in PyQt4.Qwt5), 9

QwtAbstractScaleDraw (class in PyQt4.Qwt5), 9

QwtAbstractSlider (class in PyQt4.Qwt5), 9

QwtAlphaColorMap (class in PyQt4.Qwt5), 9

QwtArrayData (class in PyQt4.Qwt5), 9

QwtArrayDouble (class in PyQt4.Qwt5), 9

QwtArrayInt (class in PyQt4.Qwt5), 9

QwtArrayQwtDoubleInterval (class in PyQt4.Qwt5), 9

QwtArrayQwtDoublePoint (class in PyQt4.Qwt5), 9

QwtArrowButton (class in PyQt4.Qwt5), 9

QwtClipper (class in PyQt4.Qwt5), 9

QwtColorMap (class in PyQt4.Qwt5), 9

QwtCompass (class in PyQt4.Qwt5), 10

QwtCompassMagnetNeedle (class in PyQt4.Qwt5), 10

QwtCompassRose (class in PyQt4.Qwt5), 10

QwtCompassWindArrow (class in PyQt4.Qwt5), 10

QwtCounter (class in PyQt4.Qwt5), 10

QwtCurveFitter (class in PyQt4.Qwt5), 10

QwtData (class in PyQt4.Qwt5), 10

QwtDial (class in PyQt4.Qwt5), 10

QwtDialNeedle (class in PyQt4.Qwt5), 10

QwtDialScaleDraw (class in PyQt4.Qwt5), 10

QwtDialSimpleNeedle (class in PyQt4.Qwt5), 10

QwtDoubleInterval (class in PyQt4.Qwt5), 10

QwtDoublePoint (class in PyQt4.Qwt5), 10

QwtDoubleRange (class in PyQt4.Qwt5), 11

QwtDoubleRect (class in PyQt4.Qwt5), 11

QwtDoubleSize (class in PyQt4.Qwt5), 11

QwtDynGridLayout (class in PyQt4.Qwt5), 11

QwtEventPattern (class in PyQt4.Qwt5), 11

QwtIntervalData (class in PyQt4.Qwt5), 11

QwtKnob (class in PyQt4.Qwt5), 11

QwtLegend (class in PyQt4.Qwt5), 11

QwtLegendItem (class in PyQt4.Qwt5), 11

QwtLegendItemManager (class in PyQt4.Qwt5), 11

QwtLegendMagnifier (class in PyQt4.Qwt5), 11

QwtLinearColorMap (class in PyQt4.Qwt5), 11

QwtLinearScaleEngine (class in PyQt4.Qwt5), 11

QwtLog10ScaleEngine (class in PyQt4.Qwt5), 11

QwtMetricsMap (class in PyQt4.Qwt5), 11

QwtPaintBuffer (class in PyQt4.Qwt5), 12

QwtPainter (class in PyQt4.Qwt5), 12

QwtPanner (class in PyQt4.Qwt5), 12

QwtPicker (class in PyQt4.Qwt5), 12

QwtPickerClickPointMachine (class in PyQt4.Qwt5),
12

QwtPickerClickRectMachine (class in PyQt4.Qwt5),
12

QwtPickerDragPointMachine (class in PyQt4.Qwt5),
12

QwtPickerDragRectMachine (class in PyQt4.Qwt5), 12

QwtPickerMachine (class in PyQt4.Qwt5), 12

QwtPickerPolygonMachine (class in PyQt4.Qwt5), 12

QwtPlainTextEngine (class in PyQt4.Qwt5), 12

QwtPlot (class in PyQt4.Qwt5), 12

QwtPlot::print (C++ function), 12

QwtPlotCanvas (class in PyQt4.Qwt5), 12

QwtPlotCurve (class in PyQt4.Qwt5), 12

QwtPlotCurve::setData (C++ function), 12

QwtPlotCurve::setRawData (C++ function), 12

QwtPlotDict (class in PyQt4.Qwt5), 12

QwtPlotGrid (class in PyQt4.Qwt5), 12

QwtPlotItem (class in PyQt4.Qwt5), 13

QwtPlotLayout (class in PyQt4.Qwt5), 13

QwtPlotMagnifier (class in PyQt4.Qwt5), 13

QwtPlotMarker (class in PyQt4.Qwt5), 13
QwtPlotPanner (class in PyQt4.Qwt5), 13
QwtPlotPicker (class in PyQt4.Qwt5), 13
QwtPlotPicker::trackerText (C++ function), 13
QwtPlotPrintFilter (class in PyQt4.Qwt5), 13
QwtPlotRasterItem (class in PyQt4.Qwt5), 13
QwtPlotScaleItem (class in PyQt4.Qwt5), 13
QwtPlotSpectrogram::drawContourLines (C++ function), 13
QwtPlotSpectrogram::renderContourLines (C++ function), 13
QwtPlotSpectrogram (class in PyQt4.Qwt5), 13
QwtPlotSvgItem (class in PyQt4.Qwt5), 13
QwtPlotZoomer (class in PyQt4.Qwt5), 13
QwtPolygon (class in PyQt4.Qwt5), 13
QwtPolygonFData (class in PyQt4.Qwt5), 13
QwtRasterData (class in PyQt4.Qwt5), 13
QwtRect (class in PyQt4.Qwt5), 14
QwtRichTextEngine (class in PyQt4.Qwt5), 14
QwtRoundScaleDraw (class in PyQt4.Qwt5), 14
QwtScaleArithmetic (class in PyQt4.Qwt5), 14
QwtScaleDiv (class in PyQt4.Qwt5), 14
QwtScaleDiv::QwtScaleDiv (C++ function), 14
QwtScaleDraw (class in PyQt4.Qwt5), 14
QwtScaleEngine (class in PyQt4.Qwt5), 14
QwtScaleMap (class in PyQt4.Qwt5), 14
QwtScaleMap::QwtScaleMap (C++ function), 14
QwtScaleTransformation (class in PyQt4.Qwt5), 14
QwtScaleWidget (class in PyQt4.Qwt5), 14
QwtSimpleCompassRose (class in PyQt4.Qwt5), 14
QwtSlider (class in PyQt4.Qwt5), 14
QwtSpline (class in PyQt4.Qwt5), 14
QwtSplineCurveFitter (class in PyQt4.Qwt5), 14
QwtSymbol (class in PyQt4.Qwt5), 14
QwtText (class in PyQt4.Qwt5), 14
QwtTextEngine (class in PyQt4.Qwt5), 14
QwtTextLabel (class in PyQt4.Qwt5), 14
QwtThermo (class in PyQt4.Qwt5), 14
QwtWheel (class in PyQt4.Qwt5), 15

S

setZoomerMouseEventSet() (Plot method), 18
Symbol (class in PyQt4.Qwt5.qplot), 18

T

to_na_array() (in module PyQt4.Qwt5), 15
to_np_array() (in module PyQt4.Qwt5), 15
toggleVisibility() (Plot method), 18
toImage() (in module PyQt4.Qwt5), 15
toNumarray() (in module PyQt4.Qwt5), 15
toNumeric() (in module PyQt4.Qwt5), 15
toNumpy() (in module PyQt4.Qwt5), 15