PyQwt Documentation

Release 5.2.0

Gerard Vermeulen

August 02, 2009

CONTENTS

CHAPTER

ONE

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 is 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 in 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 you can operate on blocks, columns, rows, slices of colums, 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 PyQwt with NumPy

PyQwt is mostly used to write graphical user interface applications. However, the following snippet shows how to use PyQwt 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),
...  "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
```

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. [2] The statement:

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

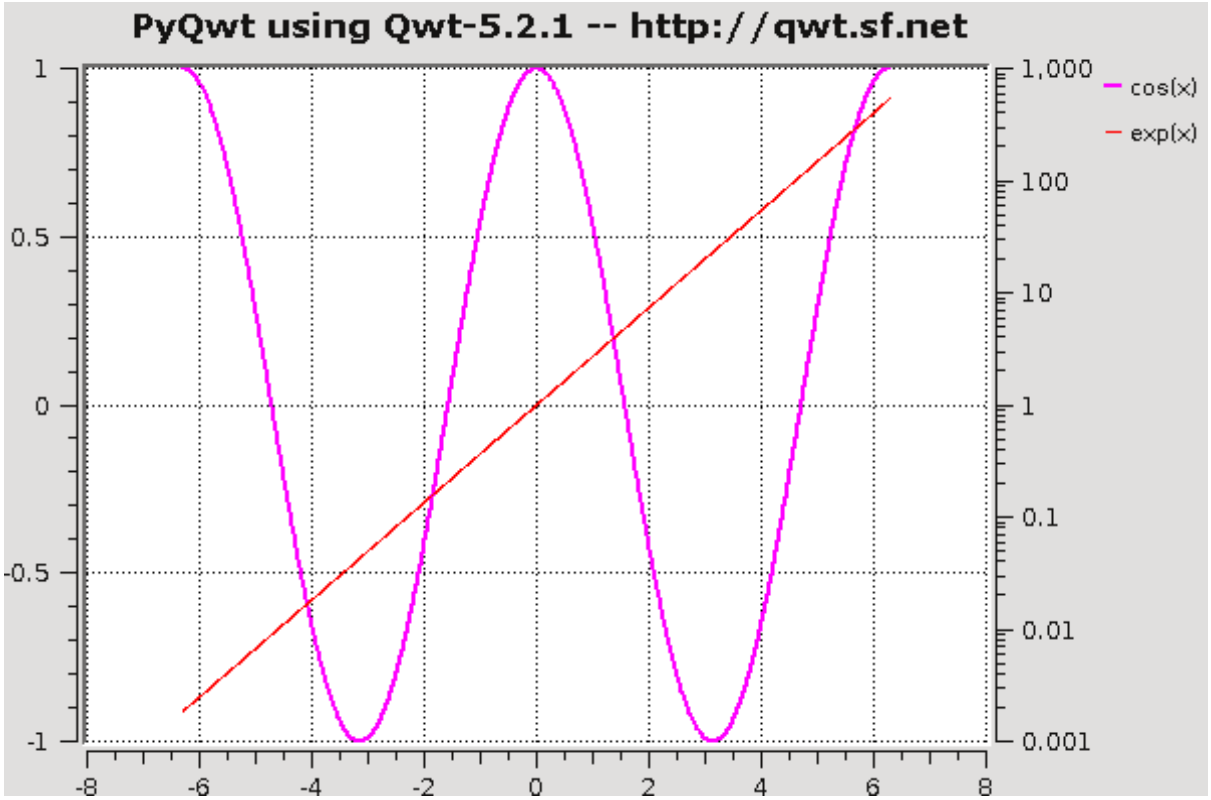
creates an array with elements increasing from -2*np.pi to 2*np.pi in steps of 0.01. The statement:

```
>>> 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:

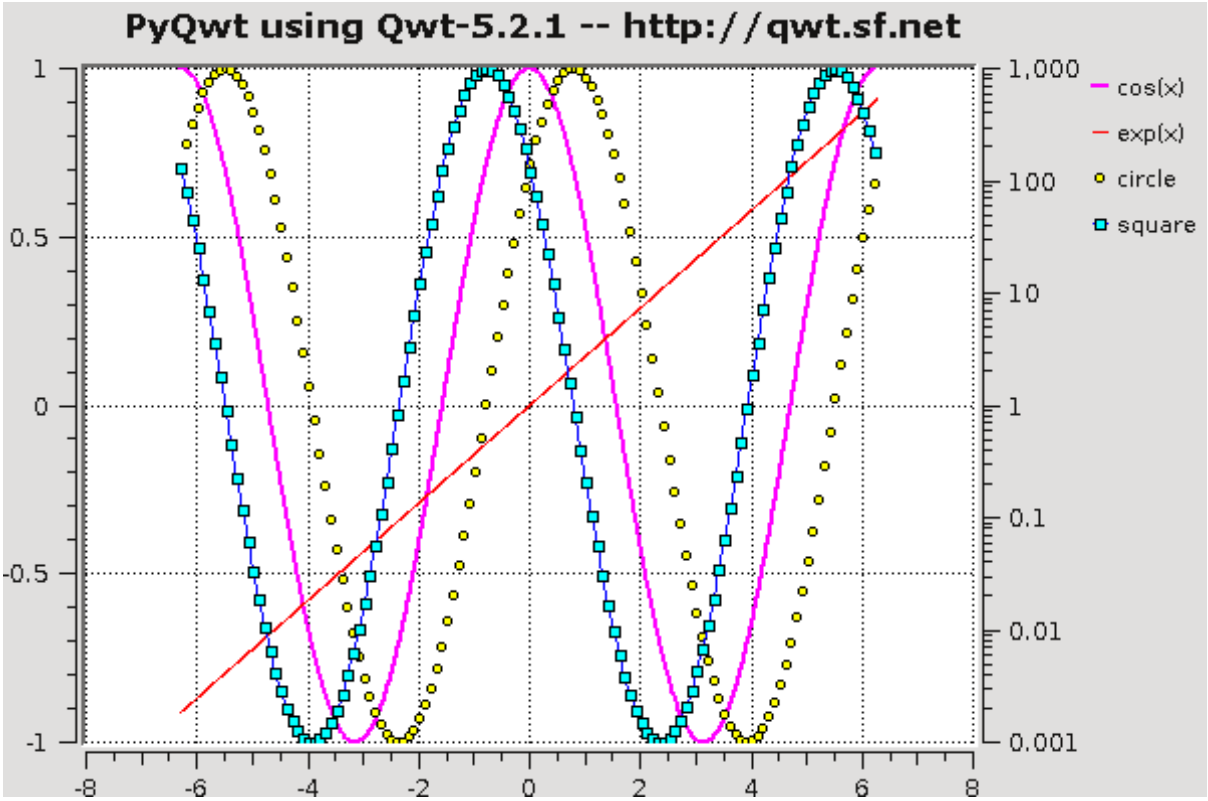
```
>>> p.plot(
...  Curve(x, np.cos(x-np.pi/4), Symbol(Circle, Yellow), "circle"),
...  Curve(x, np.cos(x+np.pi/4), Pen(Blue), Symbol(Square, Cyan),
...  "square"))
```

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

[2] PyQt-4.3.x and later support displaying Qt widgets from the Python command line interpreter.

```
>>> 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 and PyQwt3D have 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 garantee anonimity as much as possible.

CHAPTER

TWO

INSTALLATION

2.1 Source Code Installation

2.1.1 Build Prerequisites

Recommended build prerequisites for PyQwt-5.2.0 are:

1. Python, version 2.6.x and 2.5.x are supported.
2. Qt, version 4.5.x, 4.4.x, 4.3.x, and 3.3.x are supported.
3. SIP, version 4.8.x and 4.7.x (x > 3) are supported.
4. PyQt for Mac OS X, Windows, and/or X11, version 4.5.x, 4.4.x, 4.3.x, 3.18.x, and 3.17.x are supported.
5. optionally NumPy, version 1.3.x, 1.2.x, and 1.1.x are supported.
6. optionally Qwt, version 5.2.x, 5.1.x, and 5.0.x are supported.

The source package PyQwt-5.2.0.tar.gz contains a snapshot of the Qwt-5.2 subversion bug fix branch which may fix some bugs in Qwt-5.2.0. 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-5.2.0 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-5.2.0.tar.gz.
2. Invoke the following commands to build PyQwt-5.2.0 for Qt-4:

```
cd PyQwt-5.2.0
cd configure
python configure.py -Q ../qwt-5.2
make
make install
```

or invoke the commands to build PyQwt-5.2.0 for Qt-3:

```
cd PyQwt-5.2.0
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 PyQwt against the included Qwt subversion snapshot and install PyQwt. 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 into PyQwt
    -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 able to find the Qwt 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 able to find the Qwt library without
                        the -Q option)
    -j N, --jobs=N      concatenate the SIP generated code into N files
                        [default 1 per class] (to speed up make by 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/features.sip)
  -t EXTRA_SENSORY_PERCEPTION, --timelines=EXTRA_SENSORY_PERCEPTION
                      add a timeline option for SIP (normally one of the
                      timeline options in sip/timelines.sip)
  --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-5.2.0.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*.

2.2 Windows Binary Installer

Make sure that you have installed:

1. python-2.6.2.msi
2. numpy-1.3.0-win32-superpack-python2.6.exe
3. PyQt-Py2.6-gpl-4.5.4-1.exe

before installing PyQwt5.2.0-Python2.6-PyQt4.5.4-NumPy1.3.0-1.exe.

CHAPTER

THREE

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`()
FIXME.

class `QwtArrayInt`()
FIXME.

class `QwtArrayQwtDoubleInterval`()
FIXME.

class `QwtArrayQwtDoublePoint`()
FIXME.

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()
FIXME

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

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

`void` **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` **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` **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 **trackerText**(*QwtDoublePoint &point*)
is implemented as:

```
qwtText = plotPicker.trackerTextF(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()
FIXME: protected methods.

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 **QwtScaleArithmic**()
is fully implemented.

class **QwtScaleDiv**()

QwtScaleDiv(*const QwtDoubleInterval&, QwtValueList[NTickList]*)
is implemented as:

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

QwtScaleDiv(*double, double, QwtTickList[NTickList]*)
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(*int, int, double, double*)
does not exist in C++, but is provided by PyQwt.

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, numarray, or Numeric array containing data of type uint8 or uin32.

toNumarray(*image*)
Convert *image* to a 2D numarray array, where *image* must be a *QImage* with depth 8 or 32. The resulting 2D numarray 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 convertable 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.Qwt5 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.Qwt5.qplt

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.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),
...  '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=0.20000000000000001*)
Clone the plot into Grace for very high quality hard copy output.

Know bug: Grace does not scale the data correctly when Grace cannot 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.

CHAPTER

FOUR

COPYRIGHT

CHAPTER

FIVE

INDICES AND TABLES

- *Index*
- *Module Index*
- *Search Page*

MODULE INDEX

P

INDEX

S

T