
PythonQwt Manual

Release 0.12.1

Pierre Raybaut

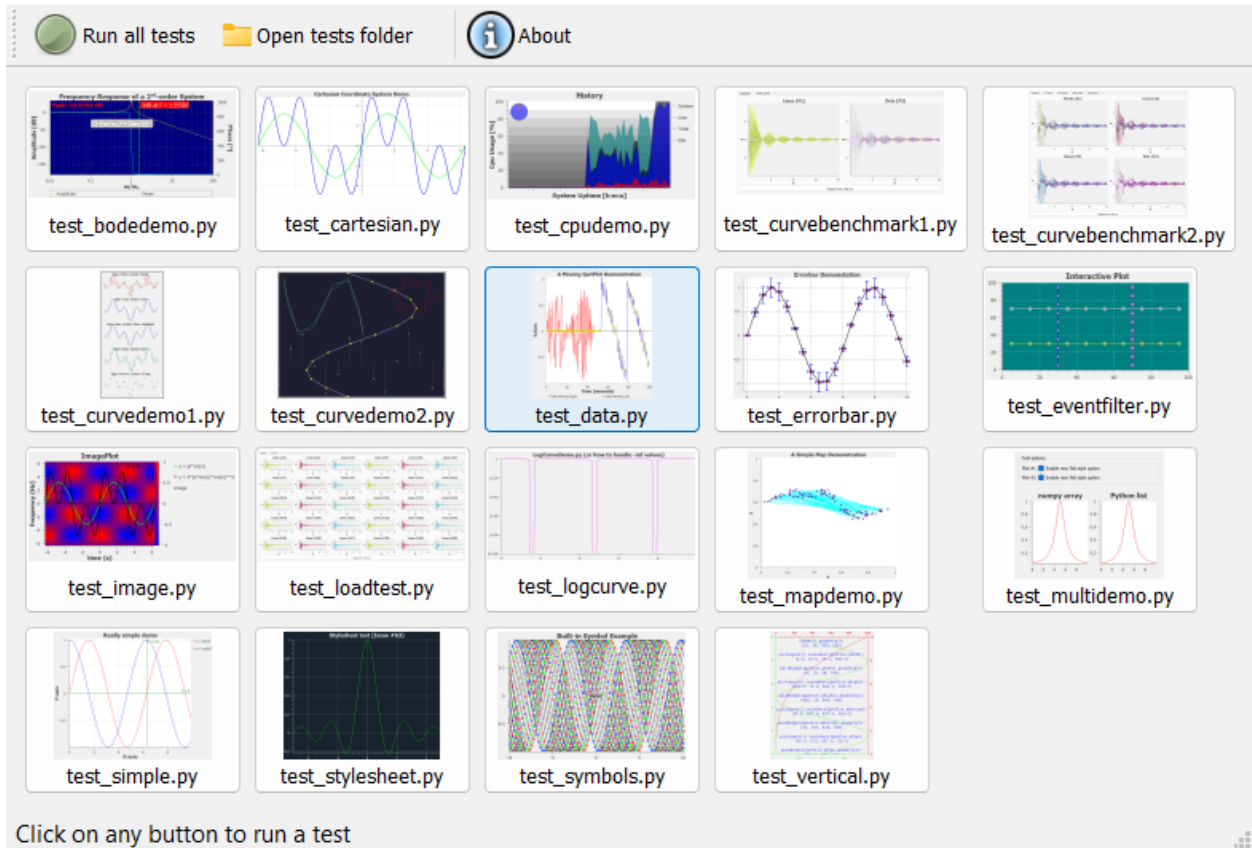
Feb 29, 2024

CONTENTS

1	Purpose and Motivation	3
2	Overview	5
3	Installation	7
3.1	Dependencies	7
3.2	Installation	7
3.3	Help and support	7
4	Examples	9
4.1	The test launcher	9
4.2	Tests	10
5	Reference	75
5.1	Plot widget fundamentals	75
5.2	Plot items	100
5.3	Additional plot features	119
5.4	Scales	132
5.5	QwtSymbol	163
5.6	Text widgets	172
5.7	Text engines	180
5.8	NumPy array to QImage	184
5.9	QwtGraphic	184
5.10	QwtInterval	189
5.11	QwtPlotDirectPainter	192
5.12	QwtPlotLayout	194
5.13	Plotting series item	200
5.14	Coordinate transformations	204
	Python Module Index	209
	Index	211

The PythonQwt package is a 2D-data plotting library using Qt graphical user interfaces for the Python programming language.

It consists of a single Python package named *qwt* which is a pure Python implementation of Qwt C++ library with some limitations.



External resources:

- Python Package Index: [PyPI](#)
- Project page on GitHub: [GitHubPage](#)
- Bug reports and feature requests: [GitHub](#)

Contents:

PURPOSE AND MOTIVATION

The `PythonQwt` project was initiated to solve -at least temporarily- the obsolescence issue of *PyQwt* (the Python-Qwt C++ bindings library) which is no longer maintained. The idea was to translate the original Qwt C++ code to Python and then to optimize some parts of the code by writing new modules based on NumPy and other libraries.

OVERVIEW

The `PythonQwt` package consists of a single Python package named `qwt` and of a few other files (examples, doc, ...):

- The subpackage `qwt.tests` contains the `PythonQwt` unit tests:
 - 75% were directly adapted from `Qwt/C++` demos (Bode demo, cartesian demo, etc.).
 - 25% were written specifically for `PythonQwt`.
 - The test launcher is an exclusive `PythonQwt` feature.

The `qwt` package is a pure Python implementation of `Qwt C++` library with the following limitations.

The following `Qwt` classes won't be reimplemented in `qwt` because more powerful features already exist in `guiqwt` (and in the forthcoming `plotpy` library, resulting of the merge of `guidata` and `guiqwt`): `QwtPlotZoomer`, `QwtCounter`, `QwtEventPattern`, `QwtPicker`, `QwtPlotPicker`.

Only the following plot items are currently implemented in `qwt` (the only plot items needed by `guiqwt` and `plotpy`): `QwtPlotItem` (base class), `QwtPlotItem`, `QwtPlotMarker`, `QwtPlotSeriesItem` and `QwtPlotCurve`.

The `HistogramItem` object implemented in `PyQwt's` `HistogramDemo.py` is not available here (a similar item is already implemented in `guiqwt/plotpy`). As a consequence, the following classes are not implemented: `QwtPlotHistogram`, `QwtIntervalSeriesData`, `QwtIntervalSample`.

The following data structure objects are not implemented as they seemed irrelevant with Python and NumPy: `QwtC-PointerData` (as a consequence, method `QwtPlot.setRawSamples` is not implemented), `QwtSyntheticPointData`.

The following sample data type objects are not implemented as they seemed quite specific: `QwtSetSample`, `QwtOHLC-Sample`. For similar reasons, the `QwtPointPolar` class and the following sample iterator objects are not implemented: `QwtSetSeriesData`, `QwtTradingChartData` and `QwtPoint3DSeriesData`.

The following classes are not implemented because they seem inappropriate in the Python/NumPy context: `QwtArray-SeriesData`, `QwtPointSeriesData`, `QwtAbstractSeriesStore`.

Threads:

- Multiple threads for graphic rendering is implemented in `Qwt C++` code thanks to the `QtConcurrent` and `QFuture` Qt features which are currently not supported by `PyQt`.
- **As a consequence the following API is not supported in `PythonQwt`:**
 - `QwtPlotItem.renderThreadCount`
 - `QwtPlotItem.setRenderThreadCount`
 - option `numThreads` in `QwtPointMapper.toImage`

The `QwtClipper` class is not implemented yet (and it will probably be very difficult or even impossible to implement it in pure Python without performance issues). As a consequence, when zooming in a plot curve, the entire curve is still painted (in other words, when working with large amount of data, there is no performance gain when zooming in).

The curve fitter feature is not implemented because powerful curve fitting features are already implemented in *guiqwt/plotpy*.

Other API compatibility issues with *Qwt*:

- *QwtPlotCurve.MinimizeMemory* option was removed as this option has no sense in PythonQwt (the polyline plotting is not taking more memory than the array data that is already there).
- *QwtPlotCurve.Fitted* option was removed as this option is not supported at the moment.

INSTALLATION

3.1 Dependencies

Requirements:

- Python 3.8 or higher
- PyQt5 5.15, PyQt6 or PySide6
- QtPy 1.9 or higher
- NumPy 1.21 or higher
- Sphinx for documentation generation
- pytest, pytest-cov, pytest-qt, coverage for unit testing

3.2 Installation

From the source package:

python -m build

3.3 Help and support

External resources:

- Bug reports and feature requests: [GitHub](#)

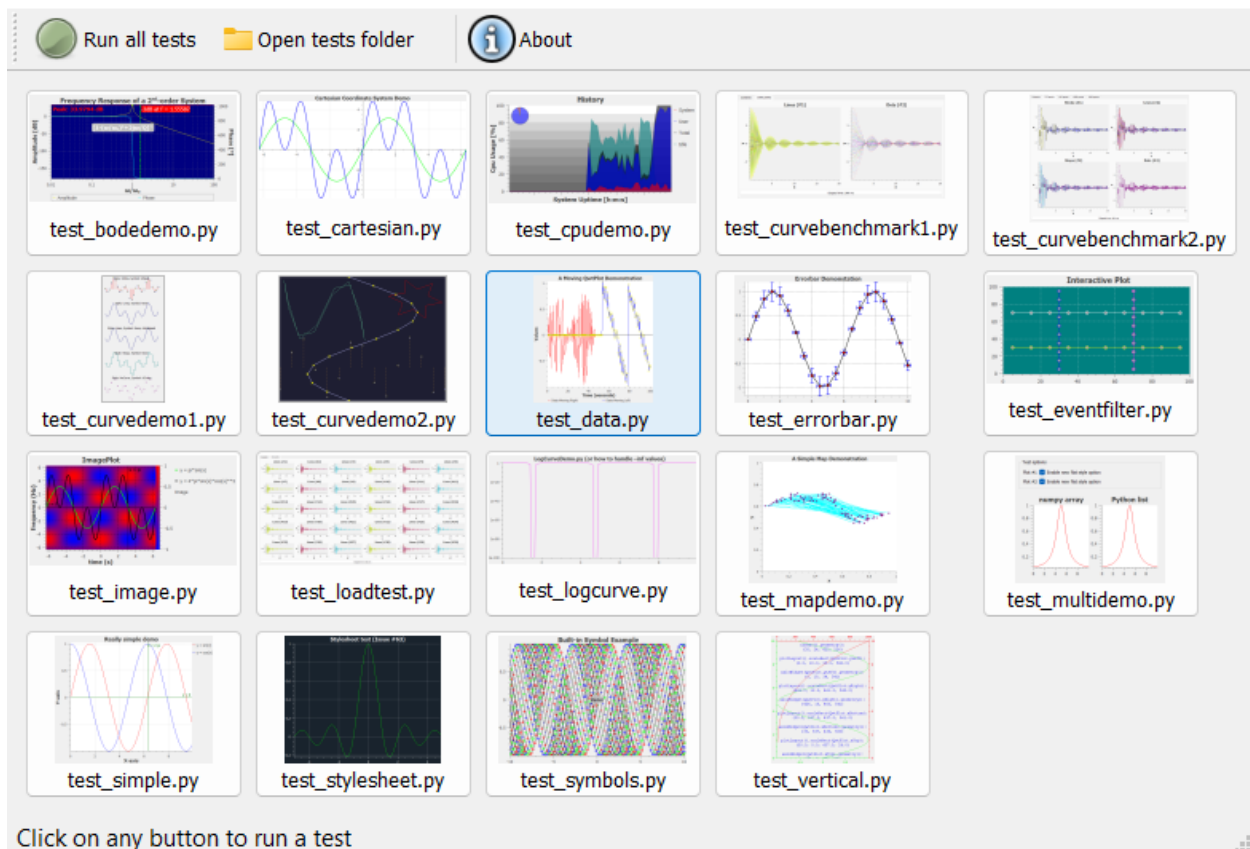
EXAMPLES

4.1 The test launcher

A lot of examples are available in the `qwt.test` module

```
from qwt import tests
tests.run()
```

The two lines above execute the PythonQwt test launcher:



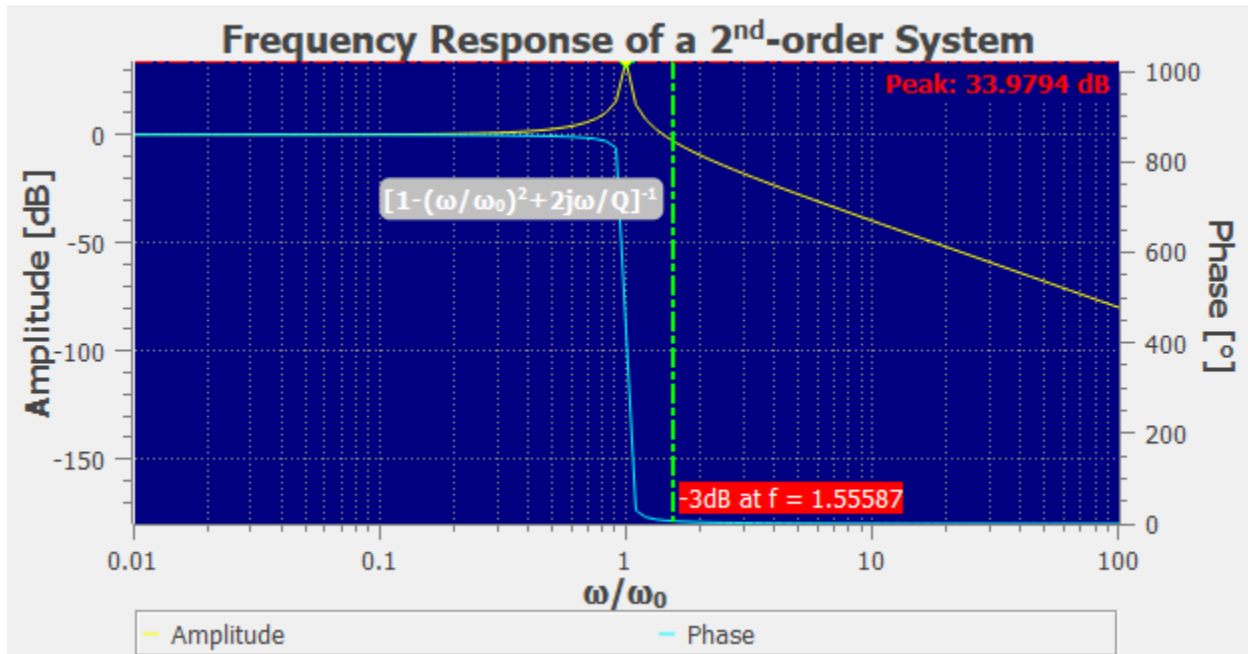
GUI-based test launcher can be executed from the command line thanks to the PythonQwt test script.

Unit tests may be executed from the command line thanks to the console-based script `PythonQwt-tests`:
`PythonQwt-tests --mode unattended.`

4.2 Tests

Here are some examples from the *qwt.test* module:

4.2.1 Bode demo



```
import os

import numpy as np
from qtpy.QtCore import Qt
from qtpy.QtGui import QFont, QIcon, QPageLayout, QPen, QPixmap
from qtpy.QtPrintSupport import QPrintDialog, QPrinter
from qtpy.QtWidgets import (
    QFrame,
    QHBoxLayout,
    QLabel,
    QMainWindow,
    QToolBar,
    QToolButton,
    QWidget,
)

from qwt import (
    QwtLegend,
    QwtLogScaleEngine,
    QwtPlot,
    QwtPlotCurve,
    QwtPlotGrid,
    QwtPlotMarker,
```

(continues on next page)

(continued from previous page)

```

    QwtPlotRenderer,
    QwtSymbol,
    QwtText,
)
from qwt.tests import utils

print_xpm = [
    "32 32 12 1",
    "a c #ffffff",
    "h c #ffff00",
    "c c #ffffff",
    "f c #dcdcdc",
    "b c #c0c0c0",
    "j c #a0a0a4",
    "e c #808080",
    "g c #808000",
    "d c #585858",
    "i c #00ff00",
    "# c #000000",
    ". c None",
    ".....",
    ".....",
    ".....###.....",
    ".....#abb###.....",
    ".....#aabbbb###.....",
    ".....#ddaaabbbb###.....",
    ".....#dddddaabbbb###.....",
    ".....#defdddddabbbb###.....",
    ".....#deaaabbbdddddabbbb###.....",
    ".....#deaaaaaabbdddddabbbb",
    "....#deaaabbbaaa#ddeddfggaaad#",
    "...#deaaaaaaaaddeeeafgggfd#",
    "..#deaaabbbaaa#ddeeeabbbgfdd#",
    ".#deeefaaaaaaddeeeabbbhbbadd#",
    "#aabbbeefaaa#ddeeeabbbbbadd#",
    "#bbaabbbbee#ddeeeabbiibadd#",
    "#bbbbbaabbeeeeeeabbbbbadd#",
    "#bjbbbbbbaabbbbeabbbbbadd#",
    "#bjjjbbbbbbaaeabbbbbadd#",
    "#bjaaajjjbbbbbbaabbbadd#",
    "#bbbbbaajjjbbbbbbaaadd#",
    "#bjbbbbbbaajjjbbbbbddddd#.",
    "#bjjjbbbbbbaajjjbbddddd#..",
    "#bjaaajjjbbbbbjaajjbdddd#...",
    "#bbbbbaajjjbbbjbaabddddd#...",
    "###bbbbbaajjjbbbbbddddd#....",
    "...###bbbbbaajbbbbbddddd#.....",
    ".....###bbbbbbjbbbbbddddd#.....",
    ".....###bbbbbbbbbdd#.....",
    ".....###bbbbbbd#.....",
    ".....###bbb#.....",
    ".....###....."
]

```

(continues on next page)

(continued from previous page)

```

]

class BodePlot(QwtPlot):
    def __init__(self, *args):
        QwtPlot.__init__(self, *args)

        self.setTitle("Frequency Response of a 2nd-order System")
        self.setCanvasBackground(Qt.darkBlue)

        # legend
        legend = QwtLegend()
        legend.setFrameStyle(QFrame.Box | QFrame.Sunken)
        self.insertLegend(legend, QwtPlot.BottomLegend)

        # grid
        QwtPlotGrid.make(plot=self, enableminor=(True, False), color=Qt.darkGray)

        # axes
        self.enableAxis(QwtPlot.yRight)
        self.setAxisTitle(QwtPlot.xBottom, "\u03c9/\u03c9<sub>0</sub>")
        self.setAxisTitle(QwtPlot.yLeft, "Amplitude [dB]")
        self.setAxisTitle(QwtPlot.yRight, "Phase [\u00b0]")

        self.setAxisMaxMajor(QwtPlot.xBottom, 6)
        self.setAxisMaxMinor(QwtPlot.xBottom, 10)
        self.setAxisScaleEngine(QwtPlot.xBottom, QwtLogScaleEngine())

        # curves
        self.curve1 = QwtPlotCurve.make(
            title="Amplitude", linecolor=Qt.yellow, plot=self, antialiased=True
        )
        self.curve2 = QwtPlotCurve.make(
            title="Phase", linecolor=Qt.cyan, plot=self, antialiased=True
        )
        self.dB3Marker = QwtPlotMarker.make(
            label=QwtText.make(color=Qt.white, brush=Qt.red, weight=QFont.Light),
            linestyle=QwtPlotMarker.VLine,
            align=Qt.AlignRight | Qt.AlignBottom,
            color=Qt.green,
            width=2,
            style=Qt.DashDotLine,
            plot=self,
        )
        self.peakMarker = QwtPlotMarker.make(
            label=QwtText.make(
                color=Qt.red, brush=self.canvasBackground(), weight=QFont.Bold
            ),
            symbol=QwtSymbol.make(QwtSymbol.Diamond, Qt.yellow, Qt.green, (7, 7)),
            linestyle=QwtPlotMarker.HLine,
            align=Qt.AlignRight | Qt.AlignBottom,
            color=Qt.red,

```

(continues on next page)

(continued from previous page)

```

        width=2,
        style=Qt.DashDotLine,
        plot=self,
    )
    QwtPlotMarker.make(
        xvalue=0.1,
        yvalue=-20.0,
        align=Qt.AlignRight | Qt.AlignBottom,
        label=QwtText.make(
            "[1-(\u03c9/\u03c9<sub>0</sub>)<sup>2</sup>+2j\u03c9/Q]"
            "<sup>-1</sup>",
            color=Qt.white,
            borderradius=2,
            borderpen=QPen(Qt.lightGray, 5),
            brush=Qt.lightGray,
            weight=QFont.Bold,
        ),
        plot=self,
    )

    self.setDamp(0.01)

def showData(self, frequency, amplitude, phase):
    self.curve1.setData(frequency, amplitude)
    self.curve2.setData(frequency, phase)

def showPeak(self, frequency, amplitude):
    self.peakMarker.setValue(frequency, amplitude)
    label = self.peakMarker.label()
    label.setText("Peak: %4g dB" % amplitude)
    self.peakMarker.setLabel(label)

def show3dB(self, frequency):
    self.dB3Marker.setValue(frequency, 0.0)
    label = self.dB3Marker.label()
    label.setText("-3dB at f = %4g" % frequency)
    self.dB3Marker.setLabel(label)

def setDamp(self, d):
    self.damping = d
    # Numerical Python: f, g, a and p are NumPy arrays!
    f = np.exp(np.log(10.0) * np.arange(-2, 2.02, 0.04))
    g = 1.0 / (1.0 - f * f + 2j * self.damping * f)
    a = 20.0 * np.log10(abs(g))
    p = 180 * np.arctan2(g.imag, g.real) / np.pi
    # for show3dB
    i3 = np.argmax(np.where(np.less(a, -3.0), a, -100.0))
    f3 = f[i3] - (a[i3] + 3.0) * (f[i3] - f[i3 - 1]) / (a[i3] - a[i3 - 1])
    # for showPeak
    imax = np.argmax(a)

    self.showPeak(f[imax], a[imax])

```

(continues on next page)

(continued from previous page)

```

        self.show3dB(f3)
        self.showData(f, a, p)

        self.replot()

FNAME_PDF = "bode.pdf"

class BodeDemo(QMainWindow):
    def __init__(self, *args):
        QMainWindow.__init__(self, *args)

        self.plot = BodePlot(self)
        self.plot.setContentsMargins(5, 5, 5, 0)

        self.setContextMenuPolicy(Qt.NoContextMenu)

        self.setCentralWidget(self.plot)

        toolBar = QToolBar(self)
        self.addToolBar(toolBar)

        btnPrint = QToolButton(toolBar)
        btnPrint.setText("Print")
        btnPrint.setIcon(QIcon(QPixmap(print_xpm)))
        btnPrint.setToolButtonStyle(Qt.ToolButtonTextUnderIcon)
        toolBar.addWidget(btnPrint)
        btnPrint.clicked.connect(self.print_)

        btnExport = QToolButton(toolBar)
        btnExport.setText("Export")
        btnExport.setIcon(QIcon(QPixmap(print_xpm)))
        btnExport.setToolButtonStyle(Qt.ToolButtonTextUnderIcon)
        toolBar.addWidget(btnExport)
        btnExport.clicked.connect(self.exportDocument)

        toolBar.addSeparator()

        dampBox = QWidget(toolBar)
        dampLayout = QHBoxLayout(dampBox)
        dampLayout.setSpacing(0)
        dampLayout.addWidget(QWidget(dampBox), 10) # spacer
        dampLayout.addWidget(QLabel("Damping Factor", dampBox), 0)
        dampLayout.addSpacing(10)

        toolBar.addWidget(dampBox)

        self.statusBar()

        self.showInfo()

```

(continues on next page)

(continued from previous page)

```

    if utils.TestEnvironment().unattended:
        self.print_(unattended=True)

def print_(self, unattended=False):
    try:
        mode = QPrinter.HighResolution
        printer = QPrinter(mode)
    except AttributeError:
        # Some PySide6 / PyQt6 versions do not have this attribute on Linux
        printer = QPrinter()

    printer.setCreator("Bode example")
    printer.setPageOrientation(QPageLayout.Landscape)
    try:
        printer.setColorMode(QPrinter.Color)
    except AttributeError:
        pass

    docName = str(self.plot.title().text())
    if not docName:
        docName.replace("\n", " -- ")
        printer.setDocName(docName)

    dialog = QPrintDialog(printer)
    if unattended:
        # Configure QPrinter object to print to PDF file
        printer.setPrinterName("")
        printer.setOutputFileName(FNAME_PDF)
        dialog.accept()
        ok = True
    else:
        ok = dialog.exec_()
    if ok:
        renderer = QwtPlotRenderer()
        renderer.renderTo(self.plot, printer)

def exportDocument(self):
    renderer = QwtPlotRenderer(self.plot)
    renderer.exportTo(self.plot, "bode")

def showInfo(self, text=""):
    self.statusBar().showMessage(text)

def moved(self, point):
    info = "Freq=%g, Ampl=%g, Phase=%g" % (
        self.plot.invTransform(QwtPlot.xBottom, point.x()),
        self.plot.invTransform(QwtPlot.yLeft, point.y()),
        self.plot.invTransform(QwtPlot.yRight, point.y()),
    )
    self.showInfo(info)

def selected(self, _):

```

(continues on next page)

(continued from previous page)

```

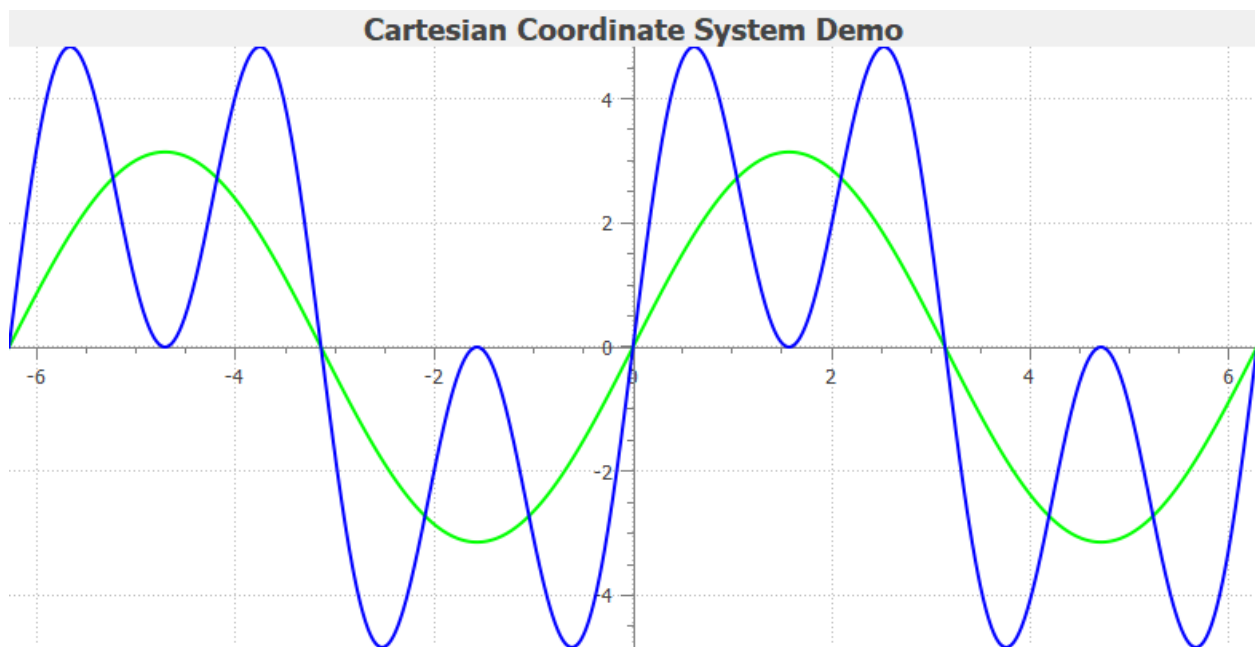
        self.showInfo()

def test_bodedemo():
    """Bode demo"""
    utils.test_widget(BodeDemo, (640, 480))
    if os.path.isfile(FNAME_PDF):
        os.remove(FNAME_PDF)

if __name__ == "__main__":
    test_bodedemo()

```

4.2.2 Cartesian demo



```

import numpy as np

from qtpy.QtCore import Qt

from qwt import QwtPlot, QwtScaleDraw, QwtPlotGrid, QwtPlotCurve, QwtPlotItem
from qwt.tests import utils

class CartesianAxis(QwtPlotItem):
    """Supports a coordinate system similar to
    http://en.wikipedia.org/wiki/Image:Cartesian-coordinate-system.svg"""

    def __init__(self, masterAxis, slaveAxis):

```

(continues on next page)

(continued from previous page)

```

        """Valid input values for masterAxis and slaveAxis are QwtPlot.yLeft,
        QwtPlot.yRight, QwtPlot.xBottom, and QwtPlot.xTop. When masterAxis is
        an x-axis, slaveAxis must be an y-axis; and vice versa."""
    QwtPlotItem.__init__(self)
    self.__axis = masterAxis
    if masterAxis in (QwtPlot.yLeft, QwtPlot.yRight):
        self.setAxes(slaveAxis, masterAxis)
    else:
        self.setAxes(masterAxis, slaveAxis)
    self.scaleDraw = QwtScaleDraw()
    self.scaleDraw.setAlignment(
        (
            QwtScaleDraw.LeftScale,
            QwtScaleDraw.RightScale,
            QwtScaleDraw.BottomScale,
            QwtScaleDraw.TopScale,
        )[masterAxis]
    )

def draw(self, painter, xMap, yMap, rect):
    """Draw an axis on the plot canvas"""
    xtr = xMap.transform
    ytr = yMap.transform
    if self.__axis in (QwtPlot.yLeft, QwtPlot.yRight):
        self.scaleDraw.move(round(xtr(0.0)), yMap.p2())
        self.scaleDraw.setLength(yMap.p1() - yMap.p2())
    elif self.__axis in (QwtPlot.xBottom, QwtPlot.xTop):
        self.scaleDraw.move(xMap.p1(), round(ytr(0.0)))
        self.scaleDraw.setLength(xMap.p2() - xMap.p1())
    self.scaleDraw.setScaleDiv(self.plot().axisScaleDiv(self.__axis))
    self.scaleDraw.draw(painter, self.plot().palette())

class CartesianPlot(QwtPlot):
    """Creates a coordinate system similar system
    http://en.wikipedia.org/wiki/Image:Cartesian-coordinate-system.svg"""

    def __init__(self, *args):
        QwtPlot.__init__(self, *args)
        self.setTitle("Cartesian Coordinate System Demo")
        # create a plot with a white canvas
        self.setCanvasBackground(Qt.white)
        # set plot layout
        self.plotLayout().setCanvasMargin(0)
        self.plotLayout().setAlignCanvasToScales(True)
        # attach a grid
        QwtPlotGrid.make(self, color=Qt.lightGray, width=0, style=Qt.DotLine, z=-1)
        # attach a x-axis
        xaxis = CartesianAxis(QwtPlot.xBottom, QwtPlot.yLeft)
        xaxis.attach(self)
        self.enableAxis(QwtPlot.xBottom, False)
        # attach a y-axis

```

(continues on next page)

(continued from previous page)

```

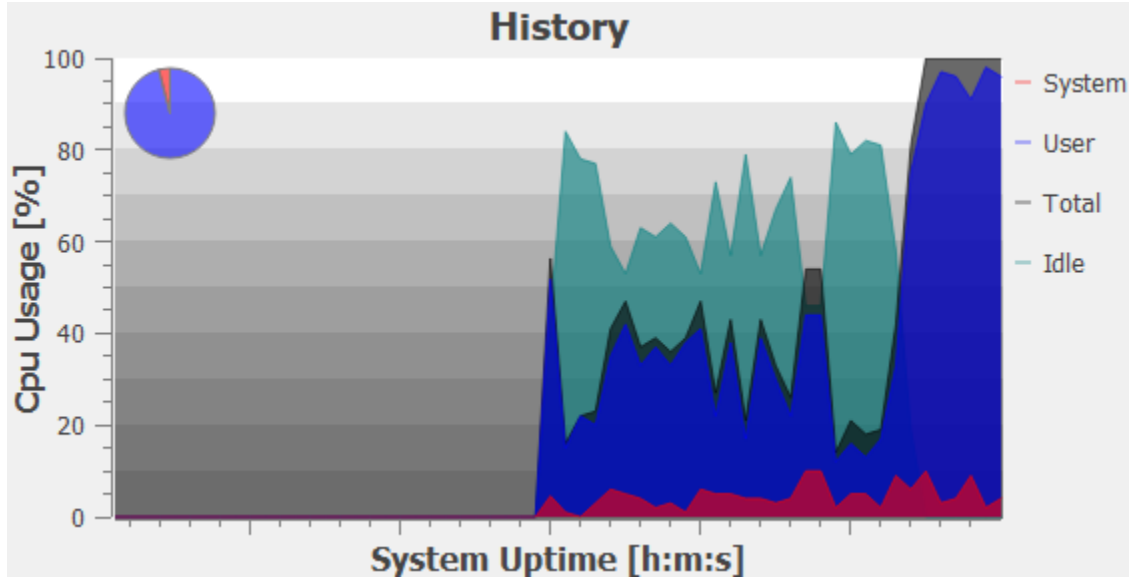
yaxis = CartesianAxis(QwtPlot.yLeft, QwtPlot.xBottom)
yaxis.attach(self)
self.enableAxis(QwtPlot.yLeft, False)
# calculate 3 NumPy arrays
x = np.arange(-2 * np.pi, 2 * np.pi, 0.01)
# attach a curve
QwtPlotCurve.make(
    x,
    np.pi * np.sin(x),
    title="y = pi*sin(x)",
    linecolor=Qt.green,
    linewidth=2,
    plot=self,
    antialiased=True,
)
# attach another curve
QwtPlotCurve.make(
    x,
    4 * np.pi * np.cos(x) * np.cos(x) * np.sin(x),
    title="y = 4*pi*sin(x)*cos(x)**2",
    linecolor=Qt.blue,
    linewidth=2,
    plot=self,
    antialiased=True,
)
self.replot()

def test_cartesian():
    """Cartesian plot test"""
    utils.test_widget(CartesianPlot, (800, 480))

if __name__ == "__main__":
    test_cartesian()

```

4.2.3 CPU plot demo



```
import os
import numpy as np

from qtpy.QtWidgets import QWidget, QVBoxLayout, QLabel
from qtpy.QtGui import QColor, QBrush
from qtpy.QtCore import QRect, QTime, Qt

from qwt import (
    QwtPlot,
    QwtPlotMarker,
    QwtScaleDraw,
    QwtLegend,
    QwtPlotCurve,
    QwtPlotItem,
    QwtLegendData,
    QwtText,
)
from qwt.tests import utils

class CpuStat:
    User = 0
    Nice = 1
    System = 2
    Idle = 3
    counter = 0
    dummyValues = (
        (103726, 0, 23484, 819556),
        (103783, 0, 23489, 819604),
        (103798, 0, 23490, 819688),
        (103820, 0, 23490, 819766),
```

(continues on next page)

(continued from previous page)

```
(103840, 0, 23493, 819843),
(103875, 0, 23499, 819902),
(103917, 0, 23504, 819955),
(103950, 0, 23508, 820018),
(103987, 0, 23510, 820079),
(104020, 0, 23513, 820143),
(104058, 0, 23514, 820204),
(104099, 0, 23520, 820257),
(104121, 0, 23525, 820330),
(104159, 0, 23530, 820387),
(104176, 0, 23534, 820466),
(104215, 0, 23538, 820523),
(104245, 0, 23541, 820590),
(104267, 0, 23545, 820664),
(104311, 0, 23555, 820710),
(104355, 0, 23565, 820756),
(104367, 0, 23567, 820842),
(104383, 0, 23572, 820921),
(104396, 0, 23577, 821003),
(104413, 0, 23579, 821084),
(104446, 0, 23588, 821142),
(104521, 0, 23594, 821161),
(104611, 0, 23604, 821161),
(104708, 0, 23607, 821161),
(104804, 0, 23611, 821161),
(104895, 0, 23620, 821161),
(104993, 0, 23622, 821161),
(105089, 0, 23626, 821161),
(105185, 0, 23630, 821161),
(105281, 0, 23634, 821161),
(105379, 0, 23636, 821161),
(105472, 0, 23643, 821161),
(105569, 0, 23646, 821161),
(105666, 0, 23649, 821161),
(105763, 0, 23652, 821161),
(105828, 0, 23661, 821187),
(105904, 0, 23666, 821206),
(105999, 0, 23671, 821206),
(106094, 0, 23676, 821206),
(106184, 0, 23686, 821206),
(106273, 0, 23692, 821211),
(106306, 0, 23700, 821270),
(106341, 0, 23703, 821332),
(106392, 0, 23709, 821375),
(106423, 0, 23715, 821438),
(106472, 0, 23721, 821483),
(106531, 0, 23727, 821517),
(106562, 0, 23732, 821582),
(106597, 0, 23736, 821643),
(106633, 0, 23737, 821706),
(106666, 0, 23742, 821768),
(106697, 0, 23744, 821835),
```

(continues on next page)

(continued from previous page)

```
(106730, 0, 23748, 821898),
(106765, 0, 23751, 821960),
(106799, 0, 23754, 822023),
(106831, 0, 23758, 822087),
(106862, 0, 23761, 822153),
(106899, 0, 23763, 822214),
(106932, 0, 23766, 822278),
(106965, 0, 23768, 822343),
(107009, 0, 23771, 822396),
(107040, 0, 23775, 822461),
(107092, 0, 23780, 822504),
(107143, 0, 23787, 822546),
(107200, 0, 23795, 822581),
(107250, 0, 23803, 822623),
(107277, 0, 23810, 822689),
(107286, 0, 23810, 822780),
(107313, 0, 23817, 822846),
(107325, 0, 23818, 822933),
(107332, 0, 23818, 823026),
(107344, 0, 23821, 823111),
(107357, 0, 23821, 823198),
(107368, 0, 23823, 823284),
(107375, 0, 23824, 823377),
(107386, 0, 23825, 823465),
(107396, 0, 23826, 823554),
(107422, 0, 23830, 823624),
(107434, 0, 23831, 823711),
(107456, 0, 23835, 823785),
(107468, 0, 23838, 823870),
(107487, 0, 23840, 823949),
(107515, 0, 23843, 824018),
(107528, 0, 23846, 824102),
(107535, 0, 23851, 824190),
(107548, 0, 23853, 824275),
(107562, 0, 23857, 824357),
(107656, 0, 23863, 824357),
(107751, 0, 23868, 824357),
(107849, 0, 23870, 824357),
(107944, 0, 23875, 824357),
(108043, 0, 23876, 824357),
(108137, 0, 23882, 824357),
(108230, 0, 23889, 824357),
(108317, 0, 23902, 824357),
(108412, 0, 23907, 824357),
(108511, 0, 23908, 824357),
(108608, 0, 23911, 824357),
(108704, 0, 23915, 824357),
(108801, 0, 23918, 824357),
(108891, 0, 23928, 824357),
(108987, 0, 23932, 824357),
(109072, 0, 23943, 824361),
(109079, 0, 23943, 824454),
```

(continues on next page)

(continued from previous page)

```

(109086, 0, 23944, 824546),
(109098, 0, 23950, 824628),
(109108, 0, 23955, 824713),
(109115, 0, 23957, 824804),
(109122, 0, 23958, 824896),
(109132, 0, 23959, 824985),
(109142, 0, 23961, 825073),
(109146, 0, 23962, 825168),
(109153, 0, 23964, 825259),
(109162, 0, 23966, 825348),
(109168, 0, 23969, 825439),
(109176, 0, 23971, 825529),
(109185, 0, 23974, 825617),
(109193, 0, 23977, 825706),
(109198, 0, 23978, 825800),
(109206, 0, 23978, 825892),
(109212, 0, 23981, 825983),
(109219, 0, 23981, 826076),
(109225, 0, 23981, 826170),
(109232, 0, 23984, 826260),
(109242, 0, 23984, 826350),
(109255, 0, 23986, 826435),
(109268, 0, 23987, 826521),
(109283, 0, 23990, 826603),
(109288, 0, 23991, 826697),
(109295, 0, 23993, 826788),
(109308, 0, 23994, 826874),
(109322, 0, 24009, 826945),
(109328, 0, 24011, 827037),
(109338, 0, 24012, 827126),
(109347, 0, 24012, 827217),
(109354, 0, 24017, 827305),
(109367, 0, 24017, 827392),
(109371, 0, 24019, 827486),
)

def __init__(self):
    self.procValues = self.__lookup()

def statistic(self):
    values = self.__lookup()
    userDelta = 0.0
    for i in [CpuStat.User, CpuStat.Nice]:
        userDelta += values[i] - self.procValues[i]
    systemDelta = values[CpuStat.System] - self.procValues[CpuStat.System]
    totalDelta = 0.0
    for i in range(len(self.procValues)):
        totalDelta += values[i] - self.procValues[i]
    self.procValues = values
    return 100.0 * userDelta / totalDelta, 100.0 * systemDelta / totalDelta

def upTime(self):

```

(continues on next page)

(continued from previous page)

```

    result = QTime()
    for item in self.procValues:
        result = result.addSecs(int(0.01 * item))
    return result

def __lookup(self):
    if os.path.exists("/proc/stat"):
        for line in open("/proc/stat"):
            words = line.split()
            if words[0] == "cpu" and len(words) >= 5:
                return [float(w) for w in words[1:]]
    else:
        result = CpuStat.dummyValues[CpuStat.counter]
        CpuStat.counter += 1
        CpuStat.counter %= len(CpuStat.dummyValues)
    return result

class CpuPieMarker(QwtPlotMarker):
    def __init__(self, *args):
        QwtPlotMarker.__init__(self, *args)
        self.setZ(1000.0)
        self.setRenderHint(QwtPlotItem.RenderAntialiased, True)

    def rtti(self):
        return QwtPlotItem.Rtti_PlotUserItem

    def draw(self, painter, xMap, yMap, rect):
        margin = 5
        pieRect = QRect()
        pieRect.setX(rect.x() + margin)
        pieRect.setY(rect.y() + margin)
        pieRect.setHeight(int(yMap.transform(80.0)))
        pieRect.setWidth(pieRect.height())

        angle = 3 * 5760 / 4
        for key in ["User", "System", "Idle"]:
            curve = self.plot().cpuPlotCurve(key)
            if curve.dataSize():
                value = int(5760 * curve.sample(0).y() / 100.0)
                painter.save()
                painter.setBrush(QBrush(curve.pen().color(), Qt.SolidPattern))
                painter.drawPie(pieRect, int(-angle), int(-value))
                painter.restore()
                angle += value

class TimeScaleDraw(QwtScaleDraw):
    def __init__(self, baseTime, *args):
        QwtScaleDraw.__init__(self, *args)
        self.baseTime = baseTime

```

(continues on next page)

(continued from previous page)

```

def label(self, value):
    upTime = self.baseTime.addSecs(int(value))
    return QwtText(upTime.toString())

class Background(QwtPlotItem):
    def __init__(self):
        QwtPlotItem.__init__(self)
        self.setZ(0.0)

    def rtti(self):
        return QwtPlotItem.Rtti_PlotUserItem

    def draw(self, painter, xMap, yMap, rect):
        c = QColor(Qt.white)
        r = QRect(rect)

        for i in range(100, 0, -10):
            r.setBottom(int(yMap.transform(i - 10)))
            r.setTop(int(yMap.transform(i)))
            painter.fillRect(r, c)
            c = c.darker(110)

class CpuCurve(QwtPlotCurve):
    def __init__(self, *args):
        QwtPlotCurve.__init__(self, *args)
        self.setRenderHint(QwtPlotItem.RenderAntialiased)

    def setColor(self, color):
        c = QColor(color)
        c.setAlpha(150)

        self.setPen(c)
        self.setBrush(c)

class CpuPlot(QwtPlot):
    HISTORY = 60

    def __init__(self, unattended=False):
        QwtPlot.__init__(self)

        self.curves = {}
        self.data = {}
        self.timeData = 1.0 * np.arange(self.HISTORY - 1, -1, -1)
        self.cpuStat = CpuStat()

        self.setAutoReplot(False)

        self.plotLayout().setAlignCanvasToScales(True)

```

(continues on next page)

(continued from previous page)

```

legend = QwtLegend()
legend.setDefaultItemMode(QwtLegendData.Checkable)
self.insertLegend(legend, QwtPlot.RightLegend)

self.setAxisTitle(QwtPlot.xBottom, "System Uptime [h:m:s]")
self.setAxisScaleDraw(QwtPlot.xBottom, TimeScaleDraw(self.cpuStat.upTime()))
self.setAxisScale(QwtPlot.xBottom, 0, self.HISTORY)
self.setAxisLabelRotation(QwtPlot.xBottom, -50.0)
self.setAxisLabelAlignment(QwtPlot.xBottom, Qt.AlignLeft | Qt.AlignBottom)

self.setAxisTitle(QwtPlot.yLeft, "Cpu Usage [%]")
self.setAxisScale(QwtPlot.yLeft, 0, 100)

background = Background()
background.attach(self)

pie = CpuPieMarker()
pie.attach(self)

curve = CpuCurve("System")
curve.setColor(Qt.red)
curve.attach(self)
self.curves["System"] = curve
self.data["System"] = np.zeros(self.HISTORY, float)

curve = CpuCurve("User")
curve.setColor(Qt.blue)
curve.setZ(curve.z() - 1.0)
curve.attach(self)
self.curves["User"] = curve
self.data["User"] = np.zeros(self.HISTORY, float)

curve = CpuCurve("Total")
curve.setColor(Qt.black)
curve.setZ(curve.z() - 2.0)
curve.attach(self)
self.curves["Total"] = curve
self.data["Total"] = np.zeros(self.HISTORY, float)

curve = CpuCurve("Idle")
curve.setColor(Qt.darkCyan)
curve.setZ(curve.z() - 3.0)
curve.attach(self)
self.curves["Idle"] = curve
self.data["Idle"] = np.zeros(self.HISTORY, float)

self.showCurve(self.curves["System"], True)
self.showCurve(self.curves["User"], True)
self.showCurve(self.curves["Total"], False or unattended)
self.showCurve(self.curves["Idle"], False or unattended)

self.startTimer(20 if unattended else 1000)

```

(continues on next page)

```

        legend.checked.connect(self.showCurve)
        self.replot()

    def timerEvent(self, e):
        for data in self.data.values():
            data[1:] = data[0:-1]
        self.data["User"][0], self.data["System"][0] = self.cpuStat.statistic()
        self.data["Total"][0] = self.data["User"][0] + self.data["System"][0]
        self.data["Idle"][0] = 100.0 - self.data["Total"][0]

        self.timeData += 1.0

        self.setAxisScale(QwtPlot.xBottom, self.timeData[-1], self.timeData[0])
        for key in self.curves.keys():
            self.curves[key].setData(self.timeData, self.data[key])

        self.replot()

    def showCurve(self, item, on, index=None):
        item.setVisible(on)
        self.legend().legendWidget(item).setChecked(on)
        self.replot()

    def cpuPlotCurve(self, key):
        return self.curves[key]

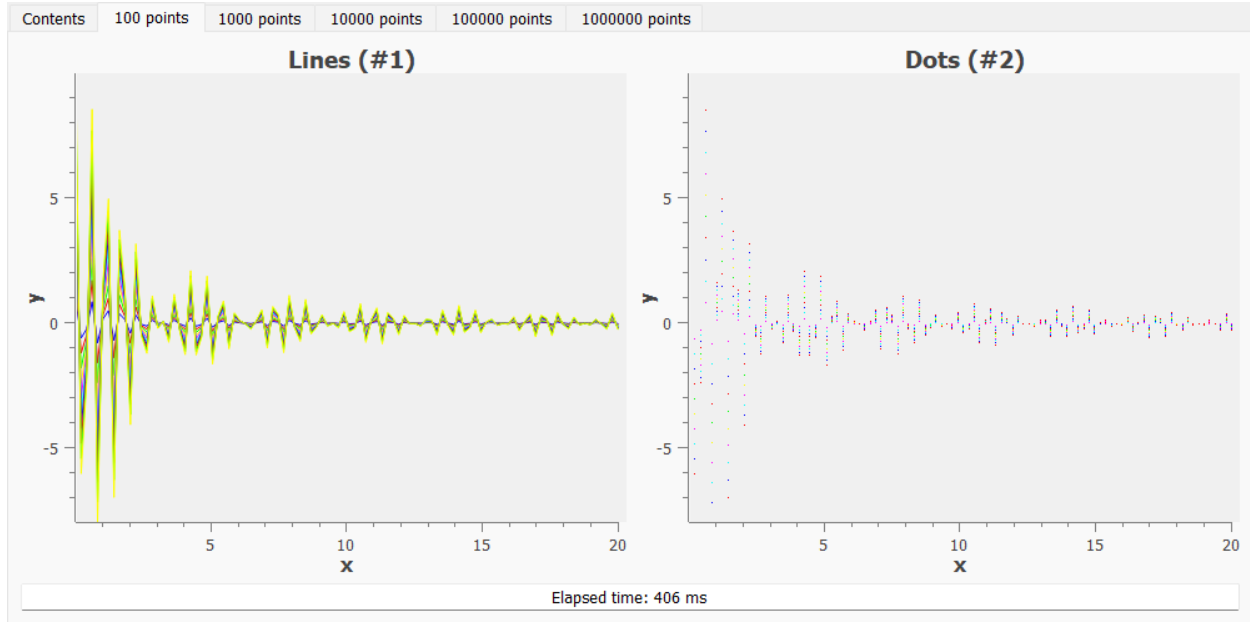
class CpuDemo(QWidget):
    def __init__(self, parent=None, unattended=False):
        super(CpuDemo, self).__init__(parent)
        layout = QVBoxLayout()
        self.setLayout(layout)
        plot = CpuPlot(unattended=unattended)
        plot.setTitle("History")
        layout.addWidget(plot)
        label = QLabel("Press the legend to en/disable a curve")
        layout.addWidget(label)

def test_cpudemo():
    """CPU demo"""
    utils.test_widget(CpuDemo, (600, 400))

if __name__ == "__main__":
    test_cpudemo()

```

4.2.4 Curve benchmark demo 1



```
import time

import numpy as np
from qtpy.QtCore import Qt
from qtpy.QtWidgets import (
    QApplication,
    QGridLayout,
    QLineEdit,
    QMainWindow,
    QTabWidget,
    QTextEdit,
    QWidget,
)

from qwt import QwtPlot, QwtPlotCurve
from qwt.tests import utils

COLOR_INDEX = None

def get_curve_color():
    global COLOR_INDEX
    colors = (Qt.blue, Qt.red, Qt.green, Qt.yellow, Qt.magenta, Qt.cyan)
    if COLOR_INDEX is None:
        COLOR_INDEX = 0
    else:
        COLOR_INDEX = (COLOR_INDEX + 1) % len(colors)
    return colors[COLOR_INDEX]
```

(continues on next page)

(continued from previous page)

```

PLOT_ID = 0

class BMPlot(QwtPlot):
    def __init__(self, title, xdata, ydata, style, symbol=None, *args):
        super(BMPlot, self).__init__(*args)
        global PLOT_ID
        self.setMinimumSize(200, 150)
        PLOT_ID += 1
        self.setTitle("%s (%d)" % (title, PLOT_ID))
        self.setAxisTitle(QwtPlot.xBottom, "x")
        self.setAxisTitle(QwtPlot.yLeft, "y")
        self.curve_nb = 0
        for idx in range(1, 11):
            self.curve_nb += 1
            QwtPlotCurve.make(
                xdata,
                ydata * idx,
                style=style,
                symbol=symbol,
                linecolor=get_curve_color(),
                antialiased=True,
                plot=self,
            )
        self.replot()

class BMWidget(QWidget):
    def __init__(self, nbc, points, *args, **kwargs):
        super(BMWidget, self).__init__()
        self.plot_nb = 0
        self.curve_nb = 0
        self.setup(nbc, points, *args, **kwargs)

    def params(self, *args, **kwargs):
        if kwargs.get("only_lines", False):
            return ("Lines", None,)
        else:
            return (
                ("Lines", None),
                ("Dots", None),
            )

    def setup(self, nbc, points, *args, **kwargs):
        x = np.linspace(0.001, 20.0, int(points))
        y = (np.sin(x) / x) * np.cos(20 * x)
        layout = QGridLayout()
        col, row = 0, 0
        for style, symbol in self.params(*args, **kwargs):
            plot = BMPlot(style, x, y, getattr(QwtPlotCurve, style), symbol=symbol)
            layout.addWidget(plot, row, col)

```

(continues on next page)

(continued from previous page)

```

        self.plot_nb += 1
        self.curve_nb += plot.curve_nb
        col += 1
        if col >= nbcol:
            row += 1
            col = 0
        self.text = QLineEdit()
        self.text.setReadOnly(True)
        self.text.setAlignment(Qt.AlignCenter)
        self.text.setText("Rendering plot...")
        layout.addWidget(self.text, row + 1, 0, 1, nbcol)
        self.setLayout(layout)

class BMText(QTextEdit):
    def __init__(self, parent=None, title=None):
        super(BMText, self).__init__(parent)
        self.setReadOnly(True)
        library = "PythonQwt"
        wintitle = self.parent().windowTitle()
        if not wintitle:
            wintitle = "Benchmark"
        if title is None:
            title = "%s example" % wintitle
        self.parent().setWindowTitle("%s [%s]" % (wintitle, library))
        self.setText(
            """\
<b>%s:</b><br>
(base plotting library: %s)<br><br>
Click on each tab to test if plotting performance is acceptable in terms of
GUI response time (switch between tabs, resize main windows, ...).<br>
<br><br>
<b>Benchmarks results:</b>
"""
            % (title, library)
        )

class CurveBenchmark1(QMainWindow):
    TITLE = "Curve benchmark"
    SIZE = (1000, 500)

    def __init__(self, max_n=1000000, parent=None, unattended=False, **kwargs):
        super(CurveBenchmark1, self).__init__(parent=parent)
        title = self.TITLE
        if kwargs.get("only_lines", False):
            title = "%s (%s)" % (title, "only lines")
        self.setWindowTitle(title)
        self.tabs = QTabWidget()
        self.setCentralWidget(self.tabs)
        self.text = BMText(self)
        self.tabs.addTab(self.text, "Contents")

```

(continues on next page)

(continued from previous page)

```

self.resize(*self.SIZE)

# Force window to show up and refresh (for test purpose only)
self.show()
QApplication.processEvents()

t0g = time.time()
self.run_benchmark(max_n, unattended, **kwargs)
dt = time.time() - t0g
self.text.append("<br><br><u>Total elapsed time</u>: %d ms" % (dt * 1e3))
self.tabs.setCurrentIndex(1 if unattended else 0)

def process_iteration(self, title, description, widget, t0):
    self.tabs.addTab(widget, title)
    self.tabs.setCurrentWidget(widget)

    # Force widget to refresh (for test purpose only)
    QApplication.processEvents()

    time_str = "Elapsed time: %d ms" % ((time.time() - t0) * 1000)
    widget.text.setText(time_str)
    self.text.append("<br><i>%s:</i><br>%s" % (description, time_str))
    print("[%s] %s" % (utils.get_lib_versions(), time_str))

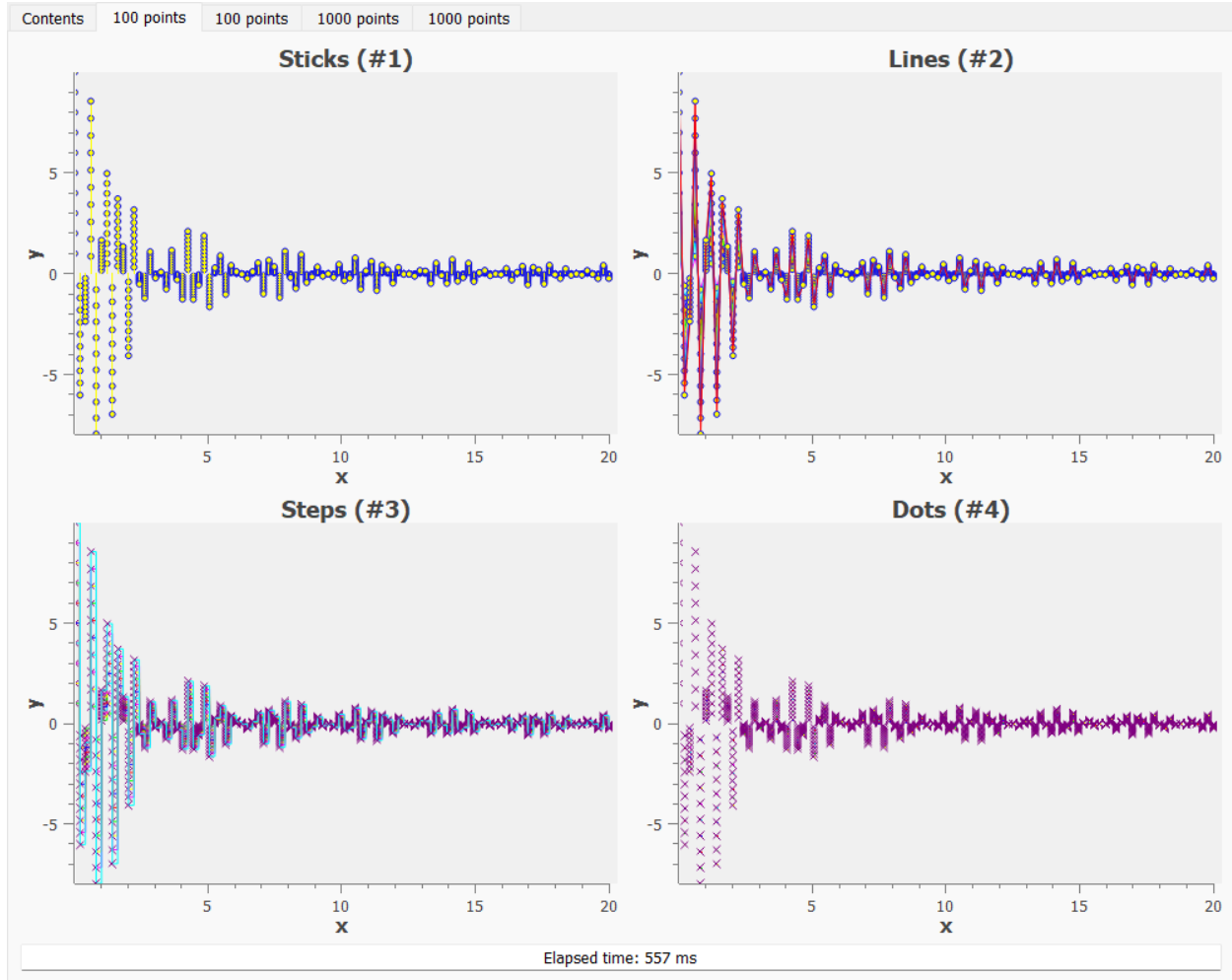
def run_benchmark(self, max_n, unattended, **kwargs):
    max_n = 1000 if unattended else max_n
    iterations = 0 if unattended else 4
    for idx in range(iterations, -1, -1):
        points = int(max_n / 10**idx)
        t0 = time.time()
        widget = BMWidget(2, points, **kwargs)
        title = "%d points" % points
        description = "%d plots with %d curves of %d points" % (
            widget.plot_nb,
            widget.curve_nb,
            points,
        )
        self.process_iteration(title, description, widget, t0)

def test_curvebenchmark1():
    """Curve benchmark example"""
    utils.test_widget(CurveBenchmark1, options=False)

if __name__ == "__main__":
    test_curvebenchmark1()

```

4.2.5 Curve benchmark demo 2



```
import time

from qtpy.QtCore import Qt

from qwt import QwtSymbol
from qwt.tests import test_curvebenchmark1 as cb
from qwt.tests import utils

class CSWidget(cb.BMWidget):
    def params(self, *args, **kwargs):
        (symbols,) = args
        symb1 = QwtSymbol.make(
            QwtSymbol.Ellipse, brush=Qt.yellow, pen=Qt.blue, size=(5, 5)
        )
        symb2 = QwtSymbol.make(QwtSymbol.XCross, pen=Qt.darkMagenta, size=(5, 5))
        if symbols:
            if kwargs.get("only_lines", False):
```

(continues on next page)

(continued from previous page)

```

        return (
            ("Lines", symb1),
            ("Lines", symb1),
            ("Lines", symb2),
            ("Lines", symb2),
        )
    else:
        return (
            ("Sticks", symb1),
            ("Lines", symb1),
            ("Steps", symb2),
            ("Dots", symb2),
        )
    else:
        if kwargs.get("only_lines", False):
            return (
                ("Lines", None),
                ("Lines", None),
                ("Lines", None),
                ("Lines", None),
            )
        else:
            return (
                ("Sticks", None),
                ("Lines", None),
                ("Steps", None),
                ("Dots", None),
            )

class CurveBenchmark2(cb.CurveBenchmark1):
    TITLE = "Curve styles"
    SIZE = (1000, 800)

    def __init__(self, max_n=1000, parent=None, unattended=False, **kwargs):
        super(CurveBenchmark2, self).__init__(
            max_n=max_n, parent=parent, unattended=unattended, **kwargs
        )

    def run_benchmark(self, max_n, unattended, **kwargs):
        for points, symbols in zip(
            (max_n / 10, max_n / 10, max_n, max_n), (True, False) * 2
        ):
            t0 = time.time()
            symtext = "with%s symbols" % (" if symbols else "out")
            widget = CSWidget(2, points, symbols, **kwargs)
            title = "%d points" % points
            description = "%d plots with %d curves of %d points, %s" % (
                widget.plot_nb,
                widget.curve_nb,
                points,
                symtext,
            )

```

(continues on next page)

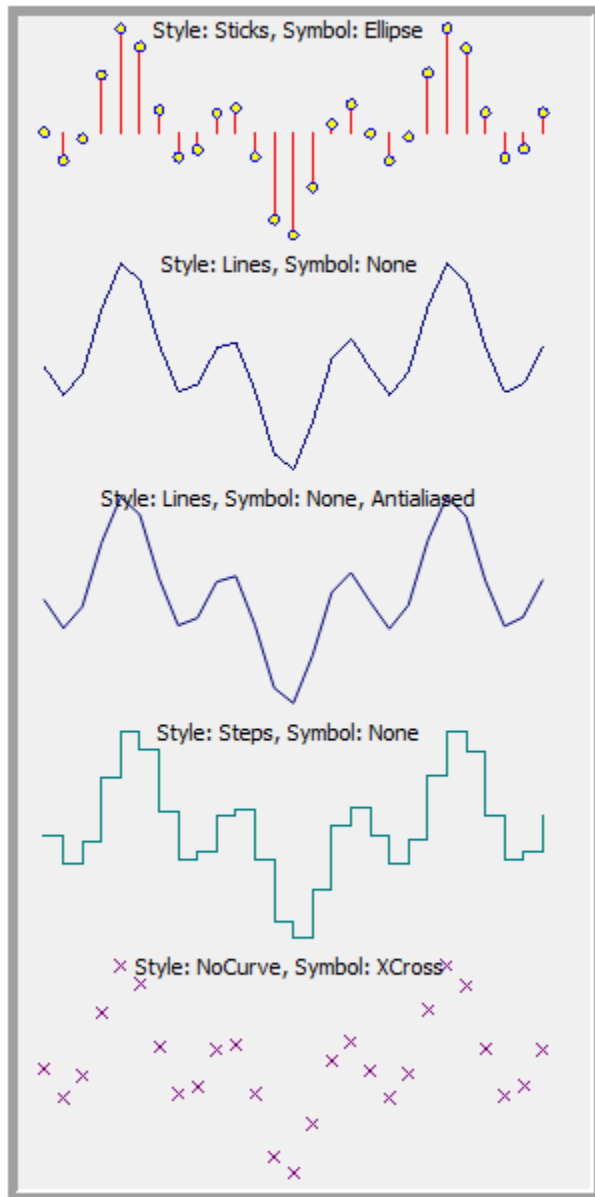
(continued from previous page)

```
        )
        self.process_iteration(title, description, widget, t0)

def test_curvebenchmark2():
    """Curve styles benchmark example"""
    utils.test_widget(CurveBenchmark2, options=False)

if __name__ == "__main__":
    test_curvebenchmark2()
```

4.2.6 Curve demo 1



```
import numpy as np

from qtpy.QtWidgets import QFrame
from qtpy.QtGui import QPen, QBrush, QFont, QPainter
from qtpy.QtCore import QSize, Qt

from qwt import QwtSymbol, QwtPlotCurve, QwtPlotItem, QwtScaleMap
from qwt.tests import utils

class CurveDemo1(QFrame):
```

(continues on next page)

(continued from previous page)

```

def __init__(self, *args):
    QFrame.__init__(self, *args)

    self.xMap = QwtScaleMap()
    self.xMap.setScaleInterval(-0.5, 10.5)
    self.yMap = QwtScaleMap()
    self.yMap.setScaleInterval(-1.1, 1.1)

    # frame style
    self.setFrameStyle(QFrame.Box | QFrame.Raised)
    self.setLineWidth(2)
    self.setMidLineWidth(3)

    # calculate values
    self.x = np.arange(0, 10.0, 10.0 / 27)
    self.y = np.sin(self.x) * np.cos(2 * self.x)

    # make curves with different styles
    self.curves = []
    self.titles = []
    # curve 1
    self.titles.append("Style: Sticks, Symbol: Ellipse")
    curve = QwtPlotCurve()
    curve.setPen(QPen(Qt.red))
    curve.setStyle(QwtPlotCurve.Sticks)
    curve.setSymbol(
        QwtSymbol(QwtSymbol.Ellipse, QBrush(Qt.yellow), QPen(Qt.blue), QSize(5, 5))
    )
    self.curves.append(curve)
    # curve 2
    self.titles.append("Style: Lines, Symbol: None")
    curve = QwtPlotCurve()
    curve.setPen(QPen(Qt.darkBlue))
    curve.setStyle(QwtPlotCurve.Lines)
    self.curves.append(curve)
    # curve 3
    self.titles.append("Style: Lines, Symbol: None, Antialiased")
    curve = QwtPlotCurve()
    curve.setPen(QPen(Qt.darkBlue))
    curve.setStyle(QwtPlotCurve.Lines)
    curve.setRenderHint(QwtPlotItem.RenderAntialiased)
    self.curves.append(curve)
    # curve 4
    self.titles.append("Style: Steps, Symbol: None")
    curve = QwtPlotCurve()
    curve.setPen(QPen(Qt.darkCyan))
    curve.setStyle(QwtPlotCurve.Steps)
    self.curves.append(curve)
    # curve 5
    self.titles.append("Style: NoCurve, Symbol: XCross")
    curve = QwtPlotCurve()
    curve.setStyle(QwtPlotCurve.NoCurve)

```

(continues on next page)

(continued from previous page)

```

        curve.setSymbol(
            QwtSymbol(QwtSymbol.XCross, QBrush(), QPen(Qt.darkMagenta), QSize(5, 5))
        )
        self.curves.append(curve)

        # attach data, using Numeric
        for curve in self.curves:
            curve.setData(self.x, self.y)

    def shiftDown(self, rect, offset):
        rect.translate(0, offset)

    def paintEvent(self, event):
        QFrame.paintEvent(self, event)
        painter = QPainter(self)
        painter.setClipRect(self.contentsRect())
        self.drawContents(painter)

    def drawContents(self, painter):
        # draw curves
        r = self.contentsRect()
        dy = int(r.height() / len(self.curves))
        r.setHeight(dy)
        for curve in self.curves:
            self.xMap.setPaintInterval(r.left(), r.right())
            self.yMap.setPaintInterval(r.top(), r.bottom())
            painter.setRenderHint(
                QPainter.Antialiasing,
                curve.testRenderHint(QwtPlotItem.RenderAntialiased),
            )
            curve.draw(painter, self.xMap, self.yMap, r)
            self.shiftDown(r, dy)

        # draw titles
        r = self.contentsRect()
        r.setHeight(dy)
        painter.setFont(QFont("Helvetica", 8))
        painter.setPen(Qt.black)
        for title in self.titles:
            painter.drawText(
                0,
                r.top(),
                r.width(),
                painter.fontMetrics().height(),
                Qt.AlignTop | Qt.AlignHCenter,
                title,
            )
            self.shiftDown(r, dy)

def test_curvedemo1():
    """Curve demo 1"""
    utils.test_widget(CurveDemo1, size=(300, 600), options=False)

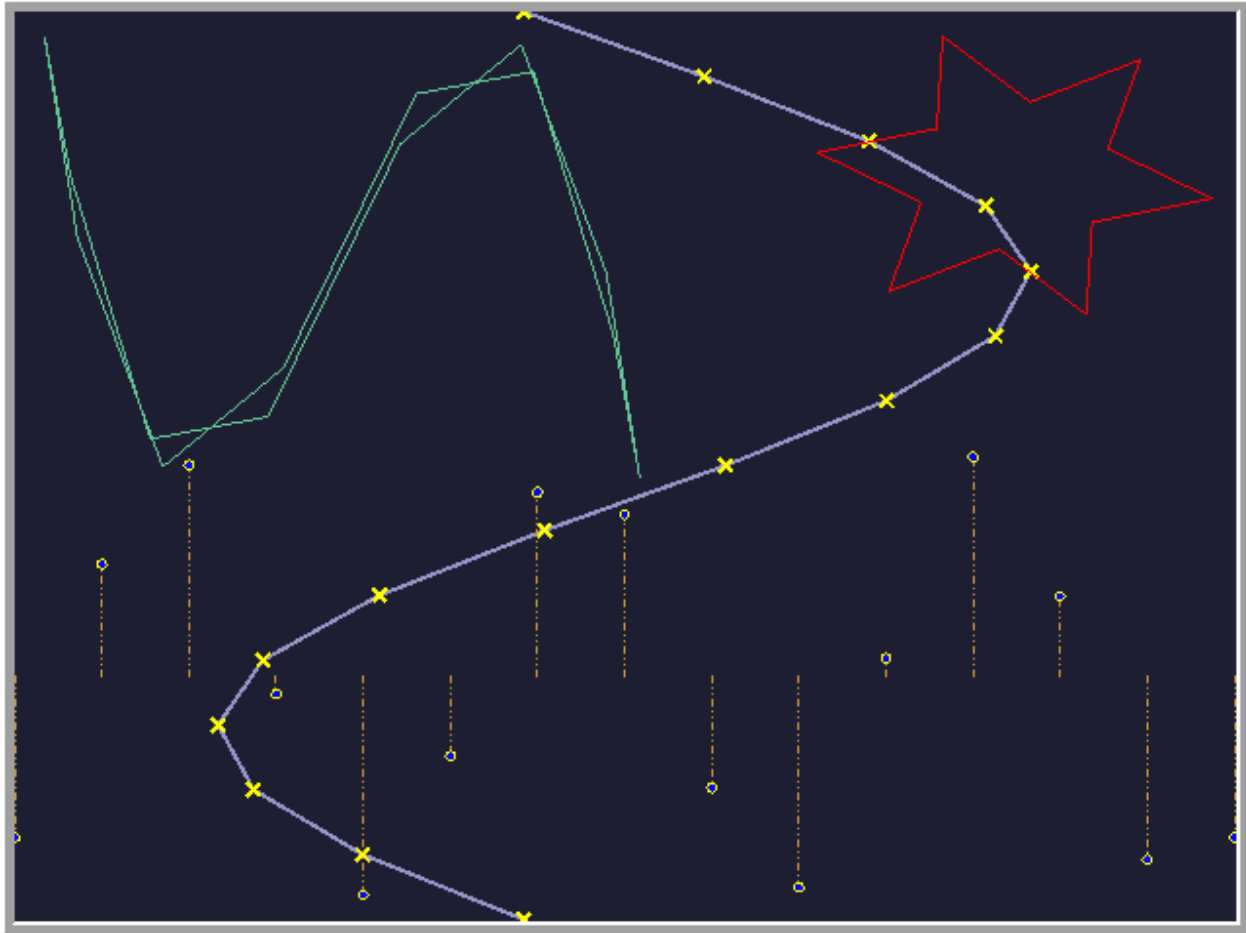
```

(continues on next page)

(continued from previous page)

```
if __name__ == "__main__":
    test_curvedemo1()
```

4.2.7 Curve demo 2



```
import numpy as np

from qtpy.QtWidgets import QFrame
from qtpy.QtGui import QPen, QBrush, QColor, QPainter, QPalette
from qtpy.QtCore import QSize
from qtpy.QtCore import Qt

from qwt import QwtScaleMap, QwtSymbol, QwtPlotCurve
from qwt.tests import utils

Size = 15
USize = 13
```

(continues on next page)

(continued from previous page)

```

class CurveDemo2(QFrame):
    def __init__(self, *args):
        QFrame.__init__(self, *args)

        self.setFrameStyle(QFrame.Box | QFrame.Raised)
        self.setLineWidth(2)
        self.setMidLineWidth(3)

        p = QPalette()
        p.setColor(self.backgroundRole(), QColor(30, 30, 50))
        self.setPalette(p)
        # make curves and maps
        self.tuples = []
        # curve 1
        curve = QwtPlotCurve()
        curve.setPen(QPen(QColor(150, 150, 200), 2))
        curve.setStyle(QwtPlotCurve.Lines)
        curve.setSymbol(
            QwtSymbol(QwtSymbol.XCross, QBrush(), QPen(Qt.yellow, 2), QSize(7, 7))
        )
        self.tuples.append(
            (curve, QwtScaleMap(0, 100, -1.5, 1.5), QwtScaleMap(0, 100, 0.0, 2 * np.pi))
        )
        # curve 2
        curve = QwtPlotCurve()
        curve.setPen(QPen(QColor(200, 150, 50), 1, Qt.DashDotDotLine))
        curve.setStyle(QwtPlotCurve.Sticks)
        curve.setSymbol(
            QwtSymbol(QwtSymbol.Ellipse, QBrush(Qt.blue), QPen(Qt.yellow), QSize(5, 5))
        )
        self.tuples.append(
            (curve, QwtScaleMap(0, 100, 0.0, 2 * np.pi), QwtScaleMap(0, 100, -3.0, 1.1))
        )
        # curve 3
        curve = QwtPlotCurve()
        curve.setPen(QPen(QColor(100, 200, 150)))
        curve.setStyle(QwtPlotCurve.Lines)
        self.tuples.append(
            (curve, QwtScaleMap(0, 100, -1.1, 3.0), QwtScaleMap(0, 100, -1.1, 3.0))
        )
        # curve 4
        curve = QwtPlotCurve()
        curve.setPen(QPen(Qt.red))
        curve.setStyle(QwtPlotCurve.Lines)
        self.tuples.append(
            (curve, QwtScaleMap(0, 100, -5.0, 1.1), QwtScaleMap(0, 100, -1.1, 5.0))
        )
        # data
        self.phase = 0.0
        self.base = np.arange(0.0, 2.01 * np.pi, 2 * np.pi / (USize - 1))

```

(continues on next page)

(continued from previous page)

```

        self.uval = np.cos(self.base)
        self.vval = np.sin(self.base)
        self.uval[1::2] *= 0.5
        self.vval[1::2] *= 0.5
        self.newValues()
        # start timer
        self.tid = self.startTimer(250)

    def paintEvent(self, event):
        QFrame.paintEvent(self, event)
        painter = QPainter(self)
        painter.setClipRect(self.contentsRect())
        self.drawContents(painter)

    def drawContents(self, painter):
        r = self.contentsRect()
        for curve, xMap, yMap in self.tuples:
            xMap.setPaintInterval(r.left(), r.right())
            yMap.setPaintInterval(r.top(), r.bottom())
            curve.draw(painter, xMap, yMap, r)

    def timerEvent(self, event):
        self.newValues()
        self.repaint()

    def newValues(self):
        phase = self.phase

        self.xval = np.arange(0, 2.01 * np.pi, 2 * np.pi / (Size - 1))
        self.yval = np.sin(self.xval - phase)
        self.zval = np.cos(3 * (self.xval + phase))

        s = 0.25 * np.sin(phase)
        c = np.sqrt(1.0 - s * s)
        u = self.uval
        self.uval = c * self.uval - s * self.vval
        self.vval = c * self.vval + s * u

        self.tuples[0][0].setData(self.yval, self.xval)
        self.tuples[1][0].setData(self.xval, self.zval)
        self.tuples[2][0].setData(self.yval, self.zval)
        self.tuples[3][0].setData(self.uval, self.vval)

        self.phase += 2 * np.pi / 100
        if self.phase > 2 * np.pi:
            self.phase = 0.0

def test_curvedemo2():
    """Curve demo 2"""
    utils.test_widget(CurveDemo2, options=False)

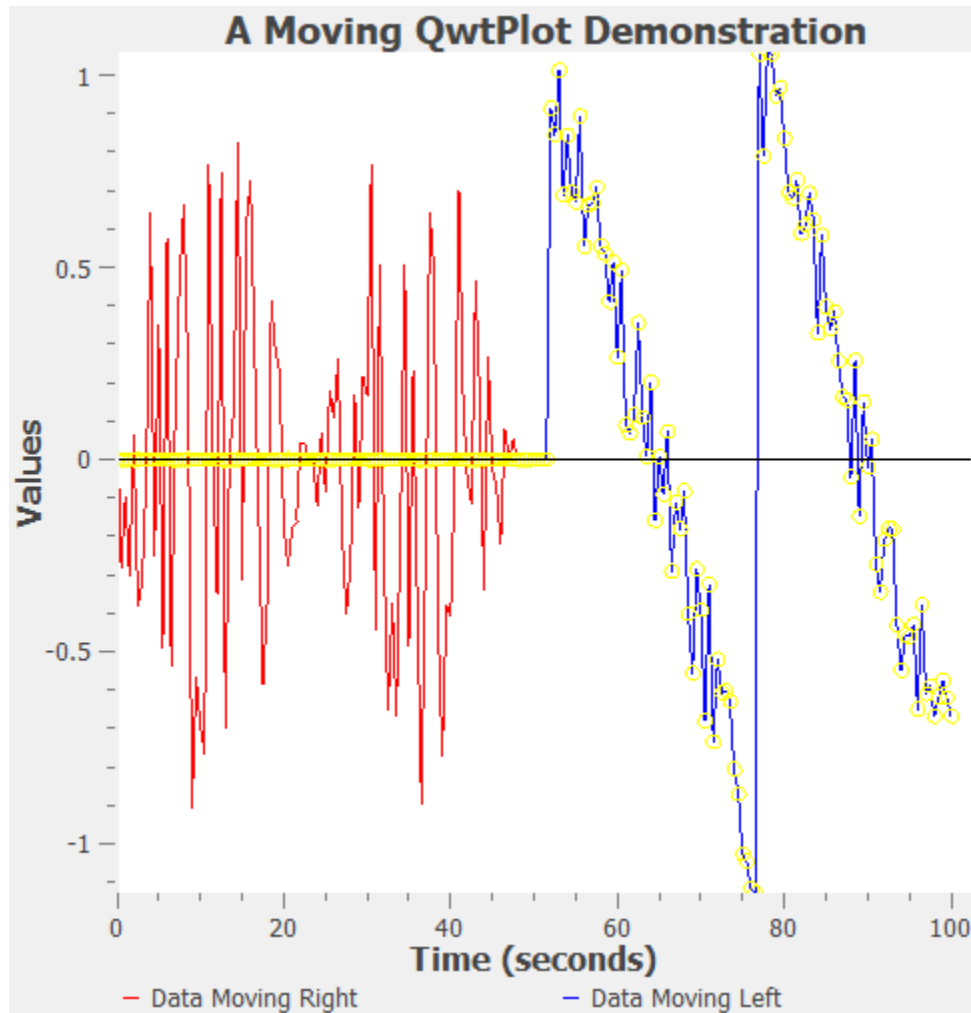
```

(continues on next page)

(continued from previous page)

```
if __name__ == "__main__":
    test_curvedemo2()
```

4.2.8 Data demo



```
import random
import numpy as np

from qtpy.QtWidgets import QFrame
from qtpy.QtGui import QPen, QBrush
from qtpy.QtCore import QSize, Qt

from qwt import (
    QwtPlot,
    QwtPlotMarker,
    QwtSymbol,
```

(continues on next page)

(continued from previous page)

```

    QwtLegend,
    QwtPlotCurve,
    QwtAbstractScaleDraw,
)
from qwt.tests import utils

class DataPlot(QwtPlot):
    def __init__(self, unattended=False):
        QwtPlot.__init__(self)

        self.setCanvasBackground(Qt.white)
        self.alignScales()

        # Initialize data
        self.x = np.arange(0.0, 100.1, 0.5)
        self.y = np.zeros(len(self.x), float)
        self.z = np.zeros(len(self.x), float)

        self.setTitle("A Moving QwtPlot Demonstration")
        self.insertLegend(QwtLegend(), QwtPlot.BottomLegend)

        self.curveR = QwtPlotCurve("Data Moving Right")
        self.curveR.attach(self)
        self.curveL = QwtPlotCurve("Data Moving Left")
        self.curveL.attach(self)

        self.curveL.setSymbol(
            QwtSymbol(QwtSymbol.Ellipse, QBrush(), QPen(Qt.yellow), QSize(7, 7))
        )

        self.curveR.setPen(QPen(Qt.red))
        self.curveL.setPen(QPen(Qt.blue))

        mY = QwtPlotMarker()
        mY.setLabelAlignment(Qt.AlignRight | Qt.AlignTop)
        mY.setLineStyle(QwtPlotMarker.HLine)
        mY.setYValue(0.0)
        mY.attach(self)

        self.setAxisTitle(QwtPlot.xBottom, "Time (seconds)")
        self.setAxisTitle(QwtPlot.yLeft, "Values")

        self.startTimer(10 if unattended else 50)
        self.phase = 0.0

    def alignScales(self):
        self.canvas().setFrameStyle(QFrame.Box | QFrame.Plain)
        self.canvas().setLineWidth(1)
        for axis_id in QwtPlot.AXES:
            scaleWidget = self.axisWidget(axis_id)
            if scaleWidget:

```

(continues on next page)

(continued from previous page)

```

        scaleWidget.setMargin(0)
        scaleDraw = self.axisScaleDraw(axis_id)
        if scaleDraw:
            scaleDraw.enableComponent(QwtAbstractScaleDraw.Backbone, False)

def timerEvent(self, e):
    if self.phase > np.pi - 0.0001:
        self.phase = 0.0

    # y moves from left to right:
    # shift y array right and assign new value y[0]
    self.y = np.concatenate((self.y[1:], self.y[:-1]))
    self.y[0] = np.sin(self.phase) * (-1.0 + 2.0 * random.random())

    # z moves from right to left:
    # Shift z array left and assign new value to z[n-1].
    self.z = np.concatenate((self.z[1:], self.z[:-1]))
    self.z[-1] = 0.8 - (2.0 * self.phase / np.pi) + 0.4 * random.random()

    self.curveR.setData(self.x, self.y)
    self.curveL.setData(self.x, self.z)

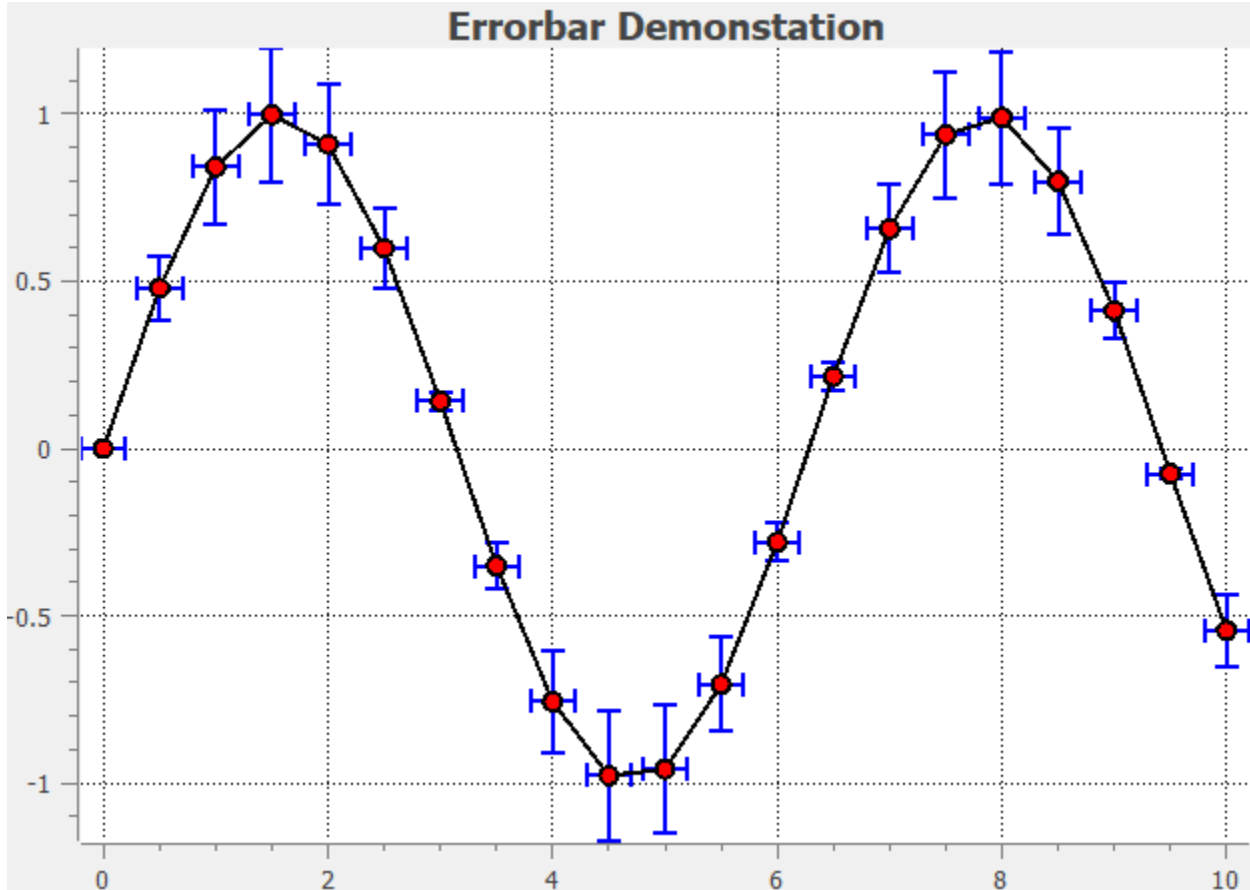
    self.replot()
    self.phase += np.pi * 0.02

def test_data():
    """Data Test"""
    utils.test_widget(DataPlot, size=(500, 300))

if __name__ == "__main__":
    test_data()

```

4.2.9 Error bar demo



```
import numpy as np

from qtpy.QtGui import QPen, QBrush
from qtpy.QtCore import QSize, QRectF, QLineF, Qt

from qwt import QwtPlot, QwtSymbol, QwtPlotGrid, QwtPlotCurve
from qwt.tests import utils

class ErrorBarPlotCurve(QwtPlotCurve):
    def __init__(
        self,
        x=[],
        y=[],
        dx=None,
        dy=None,
        curvePen=None,
        curveStyle=None,
        curveSymbol=None,
        errorPen=None,
        errorCap=0,
```

(continues on next page)

(continued from previous page)

```

    errorOnTop=False,
):
    """A curve of x versus y data with error bars in dx and dy.

    Horizontal error bars are plotted if dx is not None.
    Vertical error bars are plotted if dy is not None.

    x and y must be sequences with a shape (N,) and dx and dy must be
    sequences (if not None) with a shape (), (N,), or (2, N):
    - if dx or dy has a shape () or (N,), the error bars are given by
      (x-dx, x+dx) or (y-dy, y+dy),
    - if dx or dy has a shape (2, N), the error bars are given by
      (x-dx[0], x+dx[1]) or (y-dy[0], y+dy[1]).

    curvePen is the pen used to plot the curve

    curveStyle is the style used to plot the curve

    curveSymbol is the symbol used to plot the symbols

    errorPen is the pen used to plot the error bars

    errorCap is the size of the error bar caps

    errorOnTop is a boolean:
    - if True, plot the error bars on top of the curve,
    - if False, plot the curve on top of the error bars.
    """

    QwtPlotCurve.__init__(self)

    if curvePen is None:
        curvePen = QPen(Qt.NoPen)
    if curveStyle is None:
        curveStyle = QwtPlotCurve.Lines
    if curveSymbol is None:
        curveSymbol = QwtSymbol()
    if errorPen is None:
        errorPen = QPen(Qt.NoPen)

    self.setData(x, y, dx, dy)
    self.setPen(curvePen)
    self.setStyle(curveStyle)
    self.setSymbol(curveSymbol)
    self.errorPen = errorPen
    self.errorCap = errorCap
    self.errorOnTop = errorOnTop

    def setData(self, *args):
        """Set x versus y data with error bars in dx and dy.

        Horizontal error bars are plotted if dx is not None.

```

(continues on next page)

(continued from previous page)

Vertical error bars are plotted if dy is not None.

x and y must be sequences with a shape (N,) and dx and dy must be sequences (if not None) with a shape (), (N,), or (2, N):

- *if dx or dy has a shape () or (N,), the error bars are given by (x-dx, x+dx) or (y-dy, y+dy),*
- *if dx or dy has a shape (2, N), the error bars are given by (x-dx[0], x+dx[1]) or (y-dy[0], y+dy[1]).*

```

"""
if len(args) == 1:
    QwtPlotCurve.setData(self, *args)
    return

dx = None
dy = None
x, y = args[:2]
if len(args) > 2:
    dx = args[2]
    if len(args) > 3:
        dy = args[3]

self.__x = np.asarray(x, float)
if len(self.__x.shape) != 1:
    raise RuntimeError("len(asarray(x).shape) != 1")

self.__y = np.asarray(y, float)
if len(self.__y.shape) != 1:
    raise RuntimeError("len(asarray(y).shape) != 1")
if len(self.__x) != len(self.__y):
    raise RuntimeError("len(asarray(x)) != len(asarray(y))")

if dx is None:
    self.__dx = None
else:
    self.__dx = np.asarray(dx, float)
if len(self.__dx.shape) not in [0, 1, 2]:
    raise RuntimeError("len(asarray(dx).shape) not in [0, 1, 2]")

if dy is None:
    self.__dy = dy
else:
    self.__dy = np.asarray(dy, float)
if len(self.__dy.shape) not in [0, 1, 2]:
    raise RuntimeError("len(asarray(dy).shape) not in [0, 1, 2]")

QwtPlotCurve.setData(self, self.__x, self.__y)

def boundingRect(self):
    """Return the bounding rectangle of the data, error bars included."""
    if self.__dx is None:
        xmin = min(self.__x)
        xmax = max(self.__x)

```

(continues on next page)

(continued from previous page)

```

elif len(self.__dx.shape) in [0, 1]:
    xmin = min(self.__x - self.__dx)
    xmax = max(self.__x + self.__dx)
else:
    xmin = min(self.__x - self.__dx[0])
    xmax = max(self.__x + self.__dx[1])

if self.__dy is None:
    ymin = min(self.__y)
    ymax = max(self.__y)
elif len(self.__dy.shape) in [0, 1]:
    ymin = min(self.__y - self.__dy)
    ymax = max(self.__y + self.__dy)
else:
    ymin = min(self.__y - self.__dy[0])
    ymax = max(self.__y + self.__dy[1])

return QRectF(xmin, ymin, xmax - xmin, ymax - ymin)

def drawSeries(self, painter, xMap, yMap, canvasRect, first, last=-1):
    """Draw an interval of the curve, including the error bars

    painter is the QPainter used to draw the curve

    xMap is the QwtDiMap used to map x-values to pixels

    yMap is the QwtDiMap used to map y-values to pixels

    first is the index of the first data point to draw

    last is the index of the last data point to draw. If last < 0, last
    is transformed to index the last data point
    """

    if last < 0:
        last = self.dataSize() - 1

    if self.errorOnTop:
        QwtPlotCurve.drawSeries(self, painter, xMap, yMap, canvasRect, first, last)

    # draw the error bars
    painter.save()
    painter.setPen(self.errorPen)

    # draw the error bars with caps in the x direction
    if self.__dx is not None:
        # draw the bars
        if len(self.__dx.shape) in [0, 1]:
            xmin = self.__x - self.__dx
            xmax = self.__x + self.__dx
        else:
            xmin = self.__x - self.__dx[0]

```

(continues on next page)

(continued from previous page)

```

        xmax = self.__x + self.__dx[1]
    y = self.__y
    n, i = len(y), 0
    lines = []
    while i < n:
        yi = yMap.transform(y[i])
        lines.append(
            QLineF(xMap.transform(xmin[i]), yi, xMap.transform(xmax[i]), yi)
        )
        i += 1
    painter.drawLines(lines)
    if self.errorCap > 0:
        # draw the caps
        cap = self.errorCap / 2
        n, i, = (
            len(y),
            0,
        )
        lines = []
        while i < n:
            yi = yMap.transform(y[i])
            lines.append(
                QLineF(
                    xMap.transform(xmin[i]),
                    yi - cap,
                    xMap.transform(xmin[i]),
                    yi + cap,
                )
            )
            lines.append(
                QLineF(
                    xMap.transform(xmax[i]),
                    yi - cap,
                    xMap.transform(xmax[i]),
                    yi + cap,
                )
            )
            i += 1
        painter.drawLines(lines)

# draw the error bars with caps in the y direction
    if self.__dy is not None:
        # draw the bars
        if len(self.__dy.shape) in [0, 1]:
            ymin = self.__y - self.__dy
            ymax = self.__y + self.__dy
        else:
            ymin = self.__y - self.__dy[0]
            ymax = self.__y + self.__dy[1]
        x = self.__x
        n, i, = (
            len(x),

```

(continues on next page)

(continued from previous page)

```

        0,
    )
    lines = []
    while i < n:
        xi = xMap.transform(x[i])
        lines.append(
            QLineF(xi, yMap.transform(ymin[i]), xi, yMap.transform(ymax[i]))
        )
        i += 1
    painter.drawLines(lines)
    # draw the caps
    if self.errorCap > 0:
        cap = self.errorCap / 2
        n, i, j = len(x), 0, 0
        lines = []
        while i < n:
            xi = xMap.transform(x[i])
            lines.append(
                QLineF(
                    xi - cap,
                    yMap.transform(ymin[i]),
                    xi + cap,
                    yMap.transform(ymin[i]),
                )
            )
            lines.append(
                QLineF(
                    xi - cap,
                    yMap.transform(ymax[i]),
                    xi + cap,
                    yMap.transform(ymax[i]),
                )
            )
            i += 1
        painter.drawLines(lines)

    painter.restore()

    if not self.errorOnTop:
        QwtPlotCurve.drawSeries(self, painter, xMap, yMap, canvasRect, first, last)

class ErrorBarPlot(QwtPlot):
    def __init__(self, parent=None, title=None):
        super(ErrorBarPlot, self).__init__("Errorbar Demonstration")
        self.setCanvasBackground(Qt.white)
        self.plotLayout().setAlignCanvasToScales(True)
        grid = QwtPlotGrid()
        grid.attach(self)
        grid.setPen(QPen(Qt.black, 0, Qt.DotLine))

        # calculate data and errors for a curve with error bars

```

(continues on next page)

(continued from previous page)

```

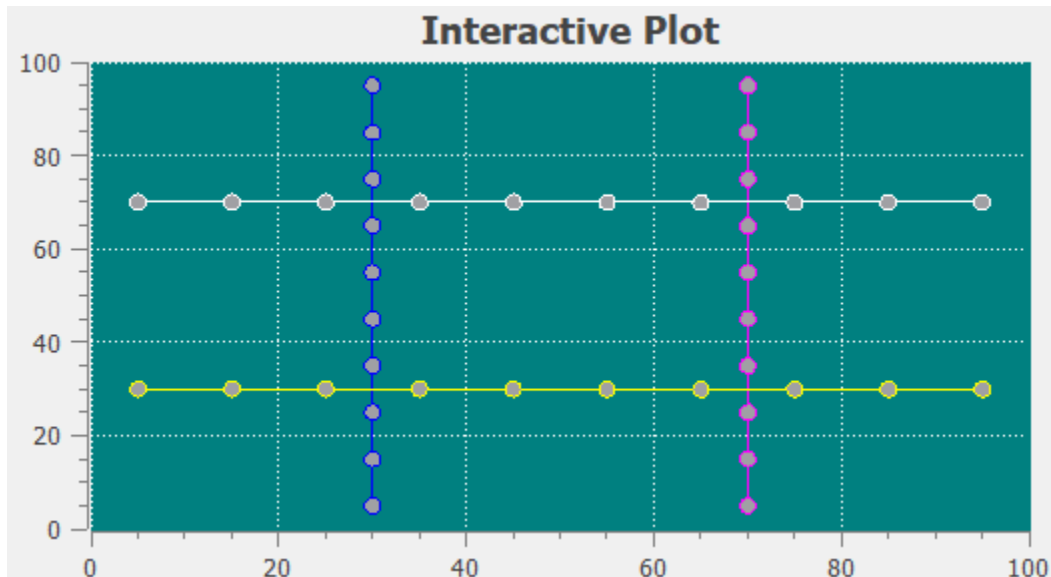
x = np.arange(0, 10.1, 0.5, float)
y = np.sin(x)
dy = 0.2 * abs(y)
# dy = (0.15 * abs(y), 0.25 * abs(y)) # uncomment for asymmetric error bars
dx = 0.2 # all error bars the same size
errorOnTop = False # uncomment to draw the curve on top of the error bars
# errorOnTop = True # uncomment to draw the error bars on top of the curve
symbol = QwtSymbol(
    QwtSymbol.Ellipse, QBrush(Qt.red), QPen(Qt.black, 2), QSize(9, 9)
)
curve = ErrorBarPlotCurve(
    x=x,
    y=y,
    dx=dx,
    dy=dy,
    curvePen=QPen(Qt.black, 2),
    curveSymbol=symbol,
    errorPen=QPen(Qt.blue, 2),
    errorCap=10,
    errorOnTop=errorOnTop,
)
curve.attach(self)

def test_errorbar():
    """Errorbar plot example"""
    utils.test_widget(ErrorBarPlot, size=(640, 480))

if __name__ == "__main__":
    test_errorbar()

```

4.2.10 Event filter demo



```
import os

import numpy as np
from qtpy.QtCore import QEvent, QObject, QPoint, QRect, QSize, Qt, Signal
from qtpy.QtGui import QBrush, QColor, QPainter, QPen, QPixmap
from qtpy.QtWidgets import QApplication, QMainWindow, QToolBar, QWhatsThis, QWidget

from qwt import (
    QwtPlot,
    QwtPlotCanvas,
    QwtPlotCurve,
    QwtPlotGrid,
    QwtScaleDiv,
    QwtScaleDraw,
    QwtSymbol,
)
from qwt.tests import utils

QT_API = os.environ["QT_API"]

class ColorBar(QWidget):
    colorSelected = Signal(QColor)

    def __init__(self, orientation, *args):
        QWidget.__init__(self, *args)
        self.__orientation = orientation
        self.__light = QColor(Qt.white)
        self.__dark = QColor(Qt.black)
        self.setCursor(Qt.PointingHandCursor)
```

(continues on next page)

(continued from previous page)

```

def setOrientation(self, orientation):
    self.__orientation = orientation
    self.update()

def orientation(self):
    return self.__orientation

def setRange(self, light, dark):
    self.__light = light
    self.__dark = dark
    self.update()

def setLight(self, color):
    self.__light = color
    self.update()

def setDark(self, color):
    self.__dark = color
    self.update()

def light(self):
    return self.__light

def dark(self):
    return self.__dark

def mousePressEvent(self, event):
    if event.button() == Qt.LeftButton:
        pm = self.grab()
        color = QColor()
        color.setRgb(pm.toImage().pixel(event.x(), event.y()))
        self.colorSelected.emit(color)
        event.accept()

def paintEvent(self, _):
    painter = QPainter(self)
    self.drawColorBar(painter, self.rect())

def drawColorBar(self, painter, rect):
    h1, s1, v1, _ = self.__light.getHsv()
    h2, s2, v2, _ = self.__dark.getHsv()
    painter.save()
    painter.setClipRect(rect)
    painter.setClipping(True)
    painter.fillRect(rect, QBrush(self.__dark))
    sectionSize = 2
    if self.__orientation == Qt.Horizontal:
        numIntervals = rect.width() / sectionSize
    else:
        numIntervals = rect.height() / sectionSize
    section = QRect()
    for i in range(int(numIntervals)):

```

(continues on next page)

(continued from previous page)

```

        if self.__orientation == Qt.Horizontal:
            section.setRect(
                rect.x() + i * sectionSize, rect.y(), sectionSize, rect.height()
            )
        else:
            section.setRect(
                rect.x(), rect.y() + i * sectionSize, rect.width(), sectionSize
            )
        ratio = float(i) / float(numIntervals)
        color = QColor()
        color.setHsv(
            h1 + int(ratio * (h2 - h1) + 0.5),
            s1 + int(ratio * (s2 - s1) + 0.5),
            v1 + int(ratio * (v2 - v1) + 0.5),
        )
        painter.fillRect(section, color)
    painter.restore()

class Plot(QwtPlot):
    def __init__(self, *args):
        QwtPlot.__init__(self, *args)

        self.setTitle("Interactive Plot")

        self.setCanvasColor(Qt.darkCyan)

        grid = QwtPlotGrid()
        grid.attach(self)
        grid.setMajorPen(QPen(Qt.white, 0, Qt.DotLine))

        self.setAxisScale(QwtPlot.xBottom, 0.0, 100.0)
        self.setAxisScale(QwtPlot.yLeft, 0.0, 100.0)

        # Avoid jumping when label with 3 digits
        # appear/disappear when scrolling vertically
        scaleDraw = self.axisScaleDraw(QwtPlot.yLeft)
        scaleDraw.setMinimumExtent(
            scaleDraw.extent(self.axisWidget(QwtPlot.yLeft).font())
        )

        self.plotLayout().setAlignCanvasToScales(True)

        self.__insertCurve(Qt.Vertical, Qt.blue, 30.0)
        self.__insertCurve(Qt.Vertical, Qt.magenta, 70.0)
        self.__insertCurve(Qt.Horizontal, Qt.yellow, 30.0)
        self.__insertCurve(Qt.Horizontal, Qt.white, 70.0)

        self.replot()

        scaleWidget = self.axisWidget(QwtPlot.yLeft)
        scaleWidget.setMargin(10)

```

(continues on next page)

(continued from previous page)

```

self.__colorBar = ColorBar(Qt.Vertical, scaleWidget)
self.__colorBar.setRange(QColor(Qt.red), QColor(Qt.darkBlue))
self.__colorBar.setFocusPolicy(Qt.TabFocus)
self.__colorBar.colorSelected.connect(self.setCanvasColor)

# we need the resize events, to lay out the color bar
scaleWidget.installEventFilter(self)

# we need the resize events, to lay out the wheel
self.canvas().installEventFilter(self)

scaleWidget.setWhatsThis(
    "Selecting a value at the scale will insert a new curve."
)
self.__colorBar.setWhatsThis(
    "Selecting a color will change the background of the plot."
)
self.axisWidget(QwtPlot.xBottom).setWhatsThis(
    "Selecting a value at the scale will insert a new curve."
)

def setCanvasColor(self, color):
    self.setCanvasBackground(color)
    self.replot()

def scrollLeftAxis(self, value):
    self.setAxisScale(QwtPlot.yLeft, value, value + 100)
    self.replot()

def eventFilter(self, obj, event):
    if event.type() == QEvent.Resize:
        size = event.size()
        if obj == self.axisWidget(QwtPlot.yLeft):
            margin = 2
            x = size.width() - obj.margin() + margin
            w = obj.margin() - 2 * margin
            y = int(obj.startBorderDist())
            h = int(size.height() - obj.startBorderDist() - obj.endBorderDist())
            self.__colorBar.setGeometry(x, y, w, h)
        return QwtPlot.eventFilter(self, obj, event)

def insertCurve(self, axis, base):
    if axis == QwtPlot.yLeft or axis == QwtPlot.yRight:
        o = Qt.Horizontal
    else:
        o = Qt.Vertical
    self.__insertCurve(o, QColor(Qt.red), base)
    self.replot()

def __insertCurve(self, orientation, color, base):
    curve = QwtPlotCurve()

```

(continues on next page)

(continued from previous page)

```

        curve.attach(self)
        curve.setPen(QPen(color))
        curve.setSymbol(
            QwtSymbol(QwtSymbol.Ellipse, QBrush(Qt.gray), QPen(color), QSize(8, 8))
        )
        fixed = base * np.ones(10, float)
        changing = np.arange(0, 95.0, 10.0, float) + 5.0
        if orientation == Qt.Horizontal:
            curve.setData(changing, fixed)
        else:
            curve.setData(fixed, changing)

class CanvasPicker(QObject):
    def __init__(self, plot):
        QObject.__init__(self, plot)
        self.__selectedCurve = None
        self.__selectedPoint = -1
        self.__plot = plot
        canvas = plot.canvas()
        canvas.installEventFilter(self)
        # We want the focus, but no focus rect.
        # The selected point will be highlighted instead.
        canvas.setFocusPolicy(Qt.StrongFocus)
        canvas.setCursor(Qt.PointingHandCursor)
        canvas.setFocusIndicator(QwtPlotCanvas.ItemFocusIndicator)
        canvas.setFocus()
        canvas.setWhatsThis(
            "All points can be moved using the left mouse button "
            "or with these keys:\n\n"
            "- Up: Select next curve\n"
            "- Down: Select previous curve\n"
            "- Left, \"-\": Select next point\n"
            "- Right, \"+\": Select previous point\n"
            "- 7, 8, 9, 4, 6, 1, 2, 3: Move selected point"
        )
        self.__shiftCurveCursor(True)

    def event(self, event):
        if event.type() == QEvent.User:
            self.__showCursor(True)
            return True
        return QObject.event(self, event)

    def eventFilter(self, object, event):
        if event.type() == QEvent.FocusIn:
            self.__showCursor(True)
        if event.type() == QEvent.FocusOut:
            self.__showCursor(False)
        if event.type() == QEvent.Paint:
            QApplication.postEvent(self, QEvent(QEvent.User))
        elif event.type() == QEvent.MouseButtonPress:

```

(continues on next page)

(continued from previous page)

```

        self.__select(event.position())
        return True
    elif event.type() == QEvent.MouseMove:
        self.__move(event.position())
        return True
    if event.type() == QEvent.KeyPress:
        delta = 5
        key = event.key()
        if key == Qt.Key_Up:
            self.__shiftCurveCursor(True)
            return True
        elif key == Qt.Key_Down:
            self.__shiftCurveCursor(False)
            return True
        elif key == Qt.Key_Right or key == Qt.Key_Plus:
            if self.__selectedCurve:
                self.__shiftPointCursor(True)
            else:
                self.__shiftCurveCursor(True)
            return True
        elif key == Qt.Key_Left or key == Qt.Key_Minus:
            if self.__selectedCurve:
                self.__shiftPointCursor(False)
            else:
                self.__shiftCurveCursor(True)
            return True
        if key == Qt.Key_1:
            self.__moveBy(-delta, delta)
        elif key == Qt.Key_2:
            self.__moveBy(0, delta)
        elif key == Qt.Key_3:
            self.__moveBy(delta, delta)
        elif key == Qt.Key_4:
            self.__moveBy(-delta, 0)
        elif key == Qt.Key_6:
            self.__moveBy(delta, 0)
        elif key == Qt.Key_7:
            self.__moveBy(-delta, -delta)
        elif key == Qt.Key_8:
            self.__moveBy(0, -delta)
        elif key == Qt.Key_9:
            self.__moveBy(delta, -delta)
    return False

def __select(self, pos):
    found, distance, point = None, 1e100, -1
    for curve in self.__plot.itemList():
        if isinstance(curve, QwtPlotCurve):
            i, d = curve.closestPoint(pos)
            if d < distance:
                found = curve
                point = i

```

(continues on next page)

(continued from previous page)

```

        distance = d
    self.__showCursor(False)
    self.__selectedCurve = None
    self.__selectedPoint = -1
    if found and distance < 10:
        self.__selectedCurve = found
        self.__selectedPoint = point
        self.__showCursor(True)

def __moveBy(self, dx, dy):
    if dx == 0 and dy == 0:
        return
    curve = self.__selectedCurve
    if not curve:
        return
    s = curve.sample(self.__selectedPoint)
    x = self.__plot.transform(curve.xAxis(), s.x()) + dx
    y = self.__plot.transform(curve.yAxis(), s.y()) + dy
    self.__move(QPoint(x, y))

def __move(self, pos):
    curve = self.__selectedCurve
    if not curve:
        return
    xData = np.zeros(curve.dataSize(), float)
    yData = np.zeros(curve.dataSize(), float)
    for i in range(curve.dataSize()):
        if i == self.__selectedPoint:
            xData[i] = self.__plot.invTransform(curve.xAxis(), pos.x())
            yData[i] = self.__plot.invTransform(curve.yAxis(), pos.y())
        else:
            s = curve.sample(i)
            xData[i] = s.x()
            yData[i] = s.y()
    curve.setData(xData, yData)
    self.__showCursor(True)
    self.__plot.replot()

def __showCursor(self, showIt):
    curve = self.__selectedCurve
    if not curve:
        return
    symbol = curve.symbol()
    brush = symbol.brush()
    if showIt:
        symbol.setBrush(symbol.brush().color().darker(180))
    curve.directPaint(self.__selectedPoint, self.__selectedPoint)
    if showIt:
        symbol.setBrush(brush)

def __shiftCurveCursor(self, up):
    curves = [

```

(continues on next page)

(continued from previous page)

```

        curve for curve in self.__plot.itemList() if isinstance(curve, QwtPlotCurve)
    ]
    if not curves:
        return
    if self.__selectedCurve in curves:
        index = curves.index(self.__selectedCurve)
        if up:
            index += 1
        else:
            index -= 1
        # keep index within [0, len(curves))
        index += len(curves)
        index %= len(curves)
    else:
        index = 0
    self.__showCursor(False)
    self.__selectedPoint = 0
    self.__selectedCurve = curves[index]
    self.__showCursor(True)

def __shiftPointCursor(self, up):
    curve = self.__selectedCurve
    if not curve:
        return
    if up:
        index = self.__selectedPoint + 1
    else:
        index = self.__selectedPoint - 1
    # keep index within [0, curve.dataSize())
    index += curve.dataSize()
    index %= curve.dataSize()
    if index != self.__selectedPoint:
        self.__showCursor(False)
        self.__selectedPoint = index
        self.__showCursor(True)

class ScalePicker(QObject):
    clicked = Signal(int, float)

    def __init__(self, plot):
        QObject.__init__(self, plot)
        for axis_id in QwtPlot.AXES:
            scaleWidget = plot.axisWidget(axis_id)
            if scaleWidget:
                scaleWidget.installEventFilter(self)

    def eventFilter(self, object, event):
        if event.type() == QEvent.MouseButtonPress:
            self.__mouseClicked(object, event.position())
            return True
        return QObject.eventFilter(self, object, event)

```

(continues on next page)

(continued from previous page)

```

def __mouseClicked(self, scale, pos):
    rect = self.__scaleRect(scale)
    margin = 10
    rect.setRect(
        rect.x() - margin,
        rect.y() - margin,
        rect.width() + 2 * margin,
        rect.height() + 2 * margin,
    )
    if rect.contains(pos):
        value = 0.0
        axis = -1
    sd = scale.scaleDraw()
    if scale.alignment() == QwtScaleDraw.LeftScale:
        value = sd.scaleMap().invTransform(pos.y())
        axis = QwtPlot.yLeft
    elif scale.alignment() == QwtScaleDraw.RightScale:
        value = sd.scaleMap().invTransform(pos.y())
        axis = QwtPlot.yRight
    elif scale.alignment() == QwtScaleDraw.BottomScale:
        value = sd.scaleMap().invTransform(pos.x())
        axis = QwtPlot.xBottom
    elif scale.alignment() == QwtScaleDraw.TopScale:
        value = sd.scaleMap().invTransform(pos.x())
        axis = QwtPlot.xBottom
    self.clicked.emit(axis, value)

def __scaleRect(self, scale):
    bld = scale.margin()
    mjt = scale.scaleDraw().tickLength(QwtScaleDiv.MajorTick)
    sbd = scale.startBorderDist()
    ebd = scale.endBorderDist()
    if scale.alignment() == QwtScaleDraw.LeftScale:
        return QRect(
            scale.width() - bld - mjt, sbd, mjt, scale.height() - sbd - ebd
        )
    elif scale.alignment() == QwtScaleDraw.RightScale:
        return QRect(bld, sbd, mjt, scale.height() - sbd - ebd)
    elif scale.alignment() == QwtScaleDraw.BottomScale:
        return QRect(sbd, bld, scale.width() - sbd - ebd, mjt)
    elif scale.alignment() == QwtScaleDraw.TopScale:
        return QRect(
            sbd, scale.height() - bld - mjt, scale.width() - sbd - ebd, mjt
        )
    else:
        return QRect()

class EventFilterWindow(QMainWindow):
    def __init__(self, parent=None):
        super(EventFilterWindow, self).__init__(parent=parent)

```

(continues on next page)

(continued from previous page)

```

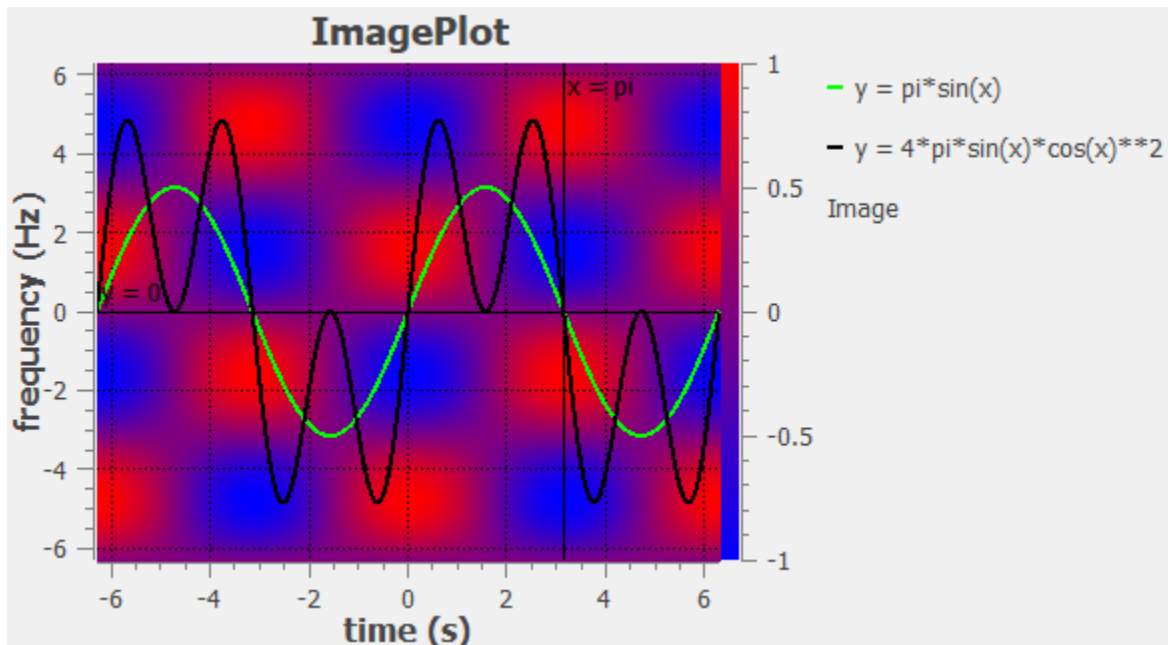
    toolBar = QToolBar(self)
    toolBar.addAction(QWhatsThis.createAction(toolBar))
    self.addToolBar(toolBar)
    plot = Plot()
    self.setCentralWidget(plot)
    plot.setWhatsThis(
        "An useless plot to demonstrate how to use event filtering.\n\n"
        "You can click on the color bar, the scales or move the slider.\n"
        "All points can be moved using the mouse or the keyboard."
    )
    CanvasPicker(plot)
    scalePicker = ScalePicker(plot)
    scalePicker.clicked.connect(plot.insertCurve)

def test_eventfilter():
    """Event filter example"""
    utils.test_widget(EventFilterWindow, size=(540, 400))

if __name__ == "__main__":
    test_eventfilter()

```

4.2.11 Image plot demo



```

import numpy as np

from qtpy.QtGui import QPen, QColor

```

(continues on next page)

(continued from previous page)

```

from qtpy.QtCore import Qt

from qwt import (
    QwtPlot,
    QwtPlotMarker,
    QwtLegend,
    QwtPlotGrid,
    QwtPlotCurve,
    QwtPlotItem,
    QwtLegendData,
    QwtLinearColorMap,
    QwtInterval,
    QwtScaleMap,
    toQImage,
)
from qwt.tests import utils

def bytescale(data, cmin=None, cmax=None, high=255, low=0):
    if (hasattr(data, "dtype") and data.dtype.char == np.uint8) or (
        hasattr(data, "typecode") and data.typecode == np.uint8
    ):
        return data
    high = high - low
    if cmin is None:
        cmin = min(np.ravel(data))
    if cmax is None:
        cmax = max(np.ravel(data))
    scale = high * 1.0 / (cmax - cmin or 1)
    bytedata = ((data * 1.0 - cmin) * scale + 0.4999).astype(np.uint8)
    return bytedata + np.asarray(low).astype(np.uint8)

def linearX(nx, ny):
    return np.repeat(np.arange(nx, typecode=np.float32)[: , np.newaxis], ny, -1)

def linearY(nx, ny):
    return np.repeat(np.arange(ny, typecode=np.float32)[np.newaxis, :], nx, 0)

def square(n, min, max):
    t = np.arange(min, max, float(max - min) / (n - 1))
    # return outer(cos(t), sin(t))
    return np.cos(t) * np.sin(t)[: , np.newaxis]

class PlotImage(QwtPlotItem):
    def __init__(self, title=""):
        QwtPlotItem.__init__(self)
        self.setTitle(title)
        self.setItemAttribute(QwtPlotItem.Legend)

```

(continues on next page)

(continued from previous page)

```

self.xyzs = None

def setData(self, xyzs, xRange=None, yRange=None):
    self.xyzs = xyzs
    shape = xyzs.shape
    if not xRange:
        xRange = (0, shape[0])
    if not yRange:
        yRange = (0, shape[1])

    self.xMap = QwtScaleMap(0, xyzs.shape[0], *xRange)
    self.plot().setAxisScale(QwtPlot.xBottom, *xRange)
    self.yMap = QwtScaleMap(0, xyzs.shape[1], *yRange)
    self.plot().setAxisScale(QwtPlot.yLeft, *yRange)

    self.image = QImage(bytescale(self.xyzs)).mirrored(False, True)
    for i in range(0, 256):
        self.image.setColor(i, qRgb(i, 0, 255 - i))

def updateLegend(self, legend, data):
    QwtPlotItem.updateLegend(self, legend, data)
    legend.find(self).setText(self.title())

def draw(self, painter, xMap, yMap, rect):
    """Paint image zoomed to xMap, yMap

    Calculate (x1, y1, x2, y2) so that it contains at least 1 pixel,
    and copy the visible region to scale it to the canvas.
    """

    assert isinstance(self.plot(), QwtPlot)

    # calculate y1, y2
    # the scanline order (index y) is inverted with respect to the y-axis
    y1 = y2 = self.image.height()
    y1 *= self.yMap.s2() - yMap.s2()
    y1 /= self.yMap.s2() - self.yMap.s1()
    y1 = max(0, int(y1 - 0.5))
    y2 *= self.yMap.s2() - yMap.s1()
    y2 /= self.yMap.s2() - self.yMap.s1()
    y2 = min(self.image.height(), int(y2 + 0.5))
    # calculate x1, x2 -- the pixel order (index x) is normal
    x1 = x2 = self.image.width()
    x1 *= xMap.s1() - self.xMap.s1()
    x1 /= self.xMap.s2() - self.xMap.s1()
    x1 = max(0, int(x1 - 0.5))
    x2 *= xMap.s2() - self.xMap.s1()
    x2 /= self.xMap.s2() - self.xMap.s1()
    x2 = min(self.image.width(), int(x2 + 0.5))
    # copy
    image = self.image.copy(x1, y1, x2 - x1, y2 - y1)
    # zoom
    image = image.scaled(

```

(continues on next page)

(continued from previous page)

```

        int(xMap.p2() - xMap.p1() + 1), int(yMap.p1() - yMap.p2() + 1)
    )
    # draw
    painter.drawImage(int(xMap.p1()), int(yMap.p2()), image)

class ImagePlot(QwtPlot):
    def __init__(self, *args):
        QwtPlot.__init__(self, *args)
        # set plot title
        self.setTitle("ImagePlot")
        # set plot layout
        self.plotLayout().setCanvasMargin(0)
        self.plotLayout().setAlignCanvasToScales(True)
        # set legend
        legend = QwtLegend()
        legend.setDefaultItemMode(QwtLegendData.Clickable)
        self.insertLegend(legend, QwtPlot.RightLegend)
        # set axis titles
        self.setAxisTitle(QwtPlot.xBottom, "time (s)")
        self.setAxisTitle(QwtPlot.yLeft, "frequency (Hz)")

        colorMap = QwtLinearColorMap(Qt.blue, Qt.red)
        interval = QwtInterval(-1, 1)
        self.enableAxis(QwtPlot.yRight)
        self.setAxisScale(QwtPlot.yRight, -1, 1)
        self.axisWidget(QwtPlot.yRight).setColorBarEnabled(True)
        self.axisWidget(QwtPlot.yRight).setColorMap(interval, colorMap)

        # calculate 3 NumPy arrays
        x = np.arange(-2 * np.pi, 2 * np.pi, 0.01)
        y = np.pi * np.sin(x)
        z = 4 * np.pi * np.cos(x) * np.cos(x) * np.sin(x)
        # attach a curve
        QwtPlotCurve.make(
            x, y, title="y = pi*sin(x)", linecolor=Qt.green, linewidth=2, plot=self
        )
        # attach another curve
        QwtPlotCurve.make(
            x, z, title="y = 4*pi*sin(x)*cos(x)**2", linewidth=2, plot=self
        )
        # attach a grid
        grid = QwtPlotGrid()
        grid.attach(self)
        grid.setPen(QPen(Qt.black, 0, Qt.DotLine))
        # attach a horizontal marker at y = 0
        QwtPlotMarker.make(
            label="y = 0",
            linestyle=QwtPlotMarker.HLine,
            align=Qt.AlignRight | Qt.AlignTop,
            plot=self,
        )

```

(continues on next page)

(continued from previous page)

```

# attach a vertical marker at x = pi
QwtPlotMarker.make(
    np.pi,
    0.0,
    label="x = pi",
    linestyle=QwtPlotMarker.VLine,
    align=Qt.AlignRight | Qt.AlignBottom,
    plot=self,
)
# attach a plot image
plotImage = PlotImage("Image")
plotImage.attach(self)
plotImage.setData(
    square(512, -2 * np.pi, 2 * np.pi),
    (-2 * np.pi, 2 * np.pi),
    (-2 * np.pi, 2 * np.pi),
)

legend.clicked.connect(self.toggleVisibility)

# replot
self.replot()

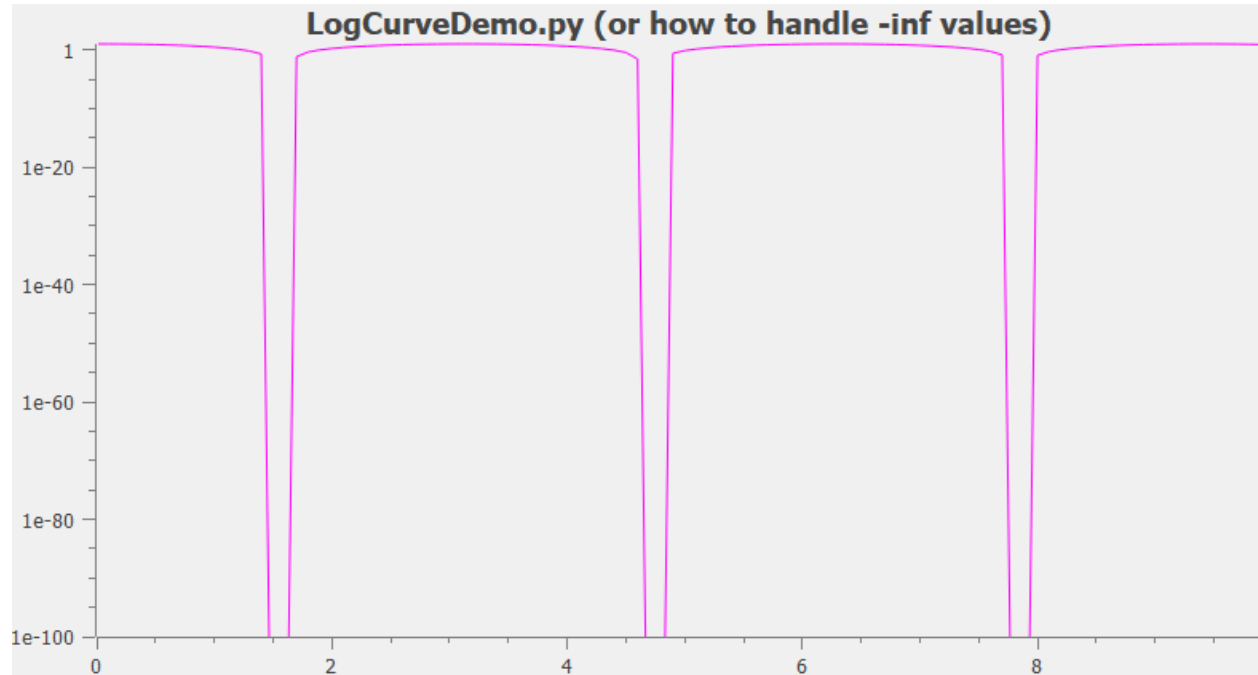
def toggleVisibility(self, plotItem, idx):
    """Toggle the visibility of a plot item"""
    plotItem.setVisible(not plotItem.isVisible())
    self.replot()

def test_image():
    """Image plot test"""
    utils.test_widget(ImagePlot, size=(600, 400))

if __name__ == "__main__":
    test_image()

```

4.2.12 Log curve plot demo



```
import numpy as np

np.seterr(all="raise")

from qtpy.QtCore import Qt

from qwt import QwtPlot, QwtPlotCurve, QwtLogScaleEngine
from qwt.tests import utils

class LogCurvePlot(QwtPlot):
    def __init__(self):
        super(LogCurvePlot, self).__init__(
            "LogCurveDemo.py (or how to handle -inf values)"
        )
        self.enableAxis(QwtPlot.xBottom)
        self.setAxisScaleEngine(QwtPlot.yLeft, QwtLogScaleEngine())
        x = np.arange(0.0, 10.0, 0.1)
        y = 10 * np.cos(x) ** 2 - 0.1
        QwtPlotCurve.make(x, y, linecolor=Qt.magenta, plot=self, antialiased=True)
        self.replot()

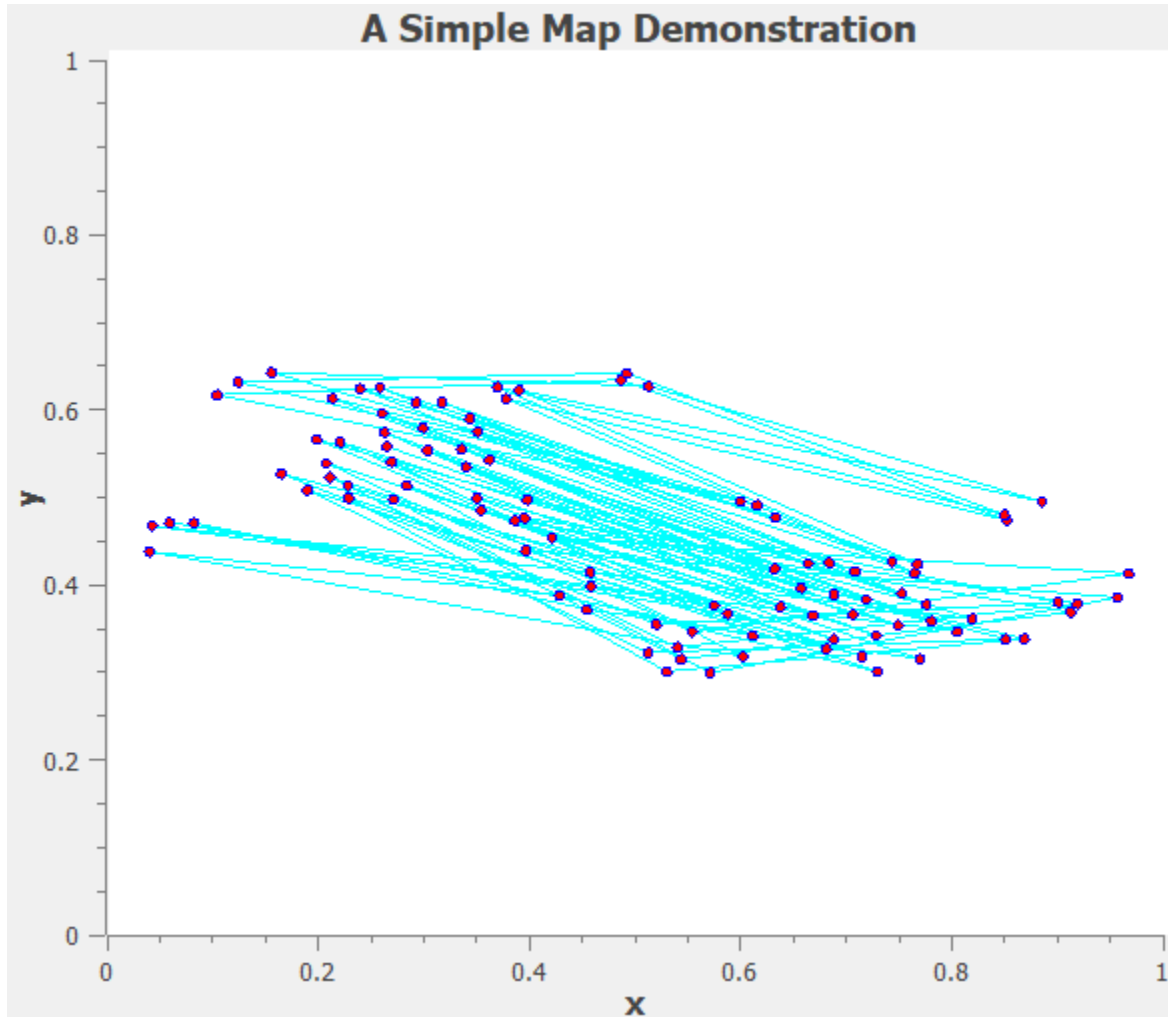
def test_logcurve():
    """Log curve demo"""
    utils.test_widget(LogCurvePlot, size=(800, 500))
```

(continues on next page)

(continued from previous page)

```
if __name__ == "__main__":
    test_logcurve()
```

4.2.13 Map demo



```
import random
import time
import numpy as np

from qtpy.QtWidgets import QMainWindow, QToolBar
from qtpy.QtGui import QPen, QBrush
from qtpy.QtCore import QSize, Qt

from qwt import QwtPlot, QwtSymbol, QwtPlotCurve
from qwt.tests import utils
```

(continues on next page)

(continued from previous page)

```

def standard_map(x, y, kappa):
    """provide one iterate of the initial conditions (x, y)
    for the standard map with parameter kappa."""
    y_new = y - kappa * np.sin(2.0 * np.pi * x)
    x_new = x + y_new
    # bring back to [0,1.0]^2
    if (x_new > 1.0) or (x_new < 0.0):
        x_new = x_new - np.floor(x_new)
    if (y_new > 1.0) or (y_new < 0.0):
        y_new = y_new - np.floor(y_new)
    return x_new, y_new

class MapDemo(QMainWindow):
    def __init__(self, *args):
        QMainWindow.__init__(self, *args)
        self.plot = QwtPlot(self)
        self.plot.setTitle("A Simple Map Demonstration")
        self.plot.setCanvasBackground(Qt.white)
        self.plot.setAxisTitle(QwtPlot.xBottom, "x")
        self.plot.setAxisTitle(QwtPlot.yLeft, "y")
        self.plot.setAxisScale(QwtPlot.xBottom, 0.0, 1.0)
        self.plot.setAxisScale(QwtPlot.yLeft, 0.0, 1.0)
        self.setCentralWidget(self.plot)
        # Initialize map data
        self.count = self.i = 1000
        self.xs = np.zeros(self.count, float)
        self.ys = np.zeros(self.count, float)
        self.kappa = 0.2
        self.curve = QwtPlotCurve("Map")
        self.curve.attach(self.plot)
        self.curve.setSymbol(
            QwtSymbol(QwtSymbol.Ellipse, QBrush(Qt.red), QPen(Qt.blue), QSize(5, 5))
        )
        self.curve.setPen(QPen(Qt.cyan))
        toolBar = QToolBar(self)
        self.addToolBar(toolBar)
        # 1 tick = 1 ms, 10 ticks = 10 ms (Linux clock is 100 Hz)
        self.ticks = 10
        self.tid = self.startTimer(self.ticks)
        self.timer_tic = None
        self.user_tic = None
        self.system_tic = None
        self.plot.replot()

    def setTicks(self, ticks):
        self.i = self.count
        self.ticks = int(ticks)
        self.killTimer(self.tid)
        self.tid = self.startTimer(ticks)

    def moreData(self):

```

(continues on next page)

(continued from previous page)

```

    if self.i == self.count:
        self.i = 0
        self.x = random.random()
        self.y = random.random()
        self.xs[self.i] = self.x
        self.ys[self.i] = self.y
        self.i += 1
        chunks = []
        self.timer_toc = time.time()
        if self.timer_tic:
            chunks.append("wall: %s s." % (self.timer_toc - self.timer_tic))
            print(" ".join(chunks))
            self.timer_tic = self.timer_toc
        else:
            self.x, self.y = standard_map(self.x, self.y, self.kappa)
            self.xs[self.i] = self.x
            self.ys[self.i] = self.y
            self.i += 1

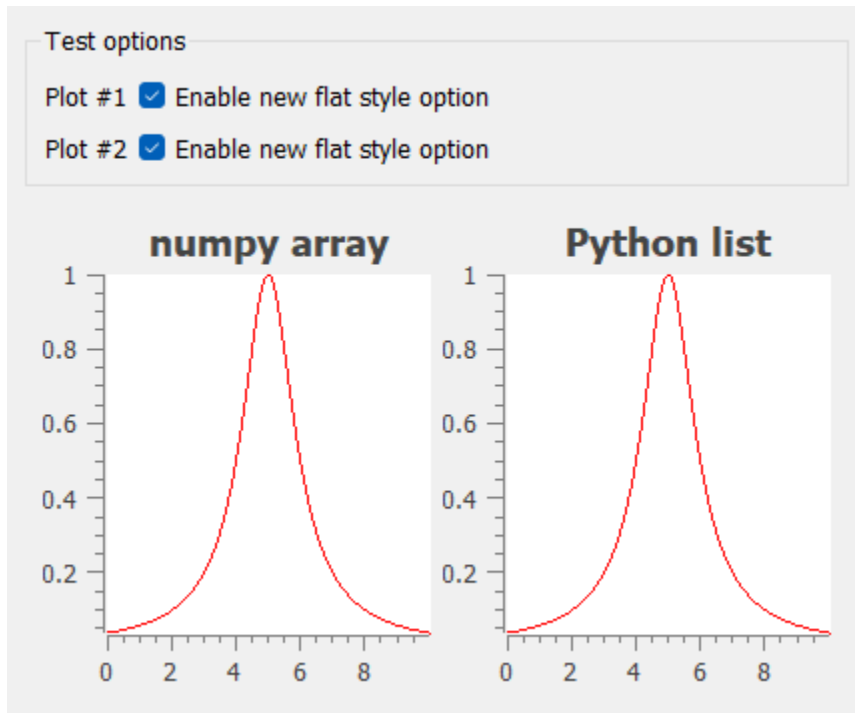
    def timerEvent(self, e):
        self.moreData()
        self.curve.setData(self.xs[: self.i], self.ys[: self.i])
        self.plot.replot()

def test_mapdemo():
    """Map demo"""
    utils.test_widget(MapDemo, size=(600, 600))

if __name__ == "__main__":
    test_mapdemo()

```

4.2.14 Multi demo



```
import numpy as np

from qtpy.QtWidgets import QGridLayout, QWidget
from qtpy.QtGui import QPen
from qtpy.QtCore import Qt

from qwt import QwtPlot, QwtPlotCurve
from qwt.tests import utils

def drange(start, stop, step):
    start, stop, step = float(start), float(stop), float(step)
    size = int(round((stop - start) / step))
    result = [start] * size
    for i in range(size):
        result[i] += i * step
    return result

def lorentzian(x):
    return 1.0 / (1.0 + (x - 5.0) ** 2)

class MultiDemo(QWidget):
    def __init__(self, *args):
        QWidget.__init__(self, *args)
        layout = QGridLayout(self)
```

(continues on next page)

(continued from previous page)

```

# try to create a plot for SciPy arrays

# make a curve and copy the data
numpy_curve = QwtPlotCurve("y = lorentzian(x)")
x = np.arange(0.0, 10.0, 0.01)
y = lorentzian(x)
numpy_curve.setData(x, y)
# here, we know we can plot NumPy arrays
numpy_plot = QwtPlot(self)
numpy_plot.setTitle("numpy array")
numpy_plot.setCanvasBackground(Qt.white)
numpy_plot.plotLayout().setCanvasMargin(0)
numpy_plot.plotLayout().setAlignCanvasToScales(True)
# insert a curve and make it red
numpy_curve.attach(numpy_plot)
numpy_curve.setPen(QPen(Qt.red))
layout.addWidget(numpy_plot, 0, 0)
numpy_plot.replot()

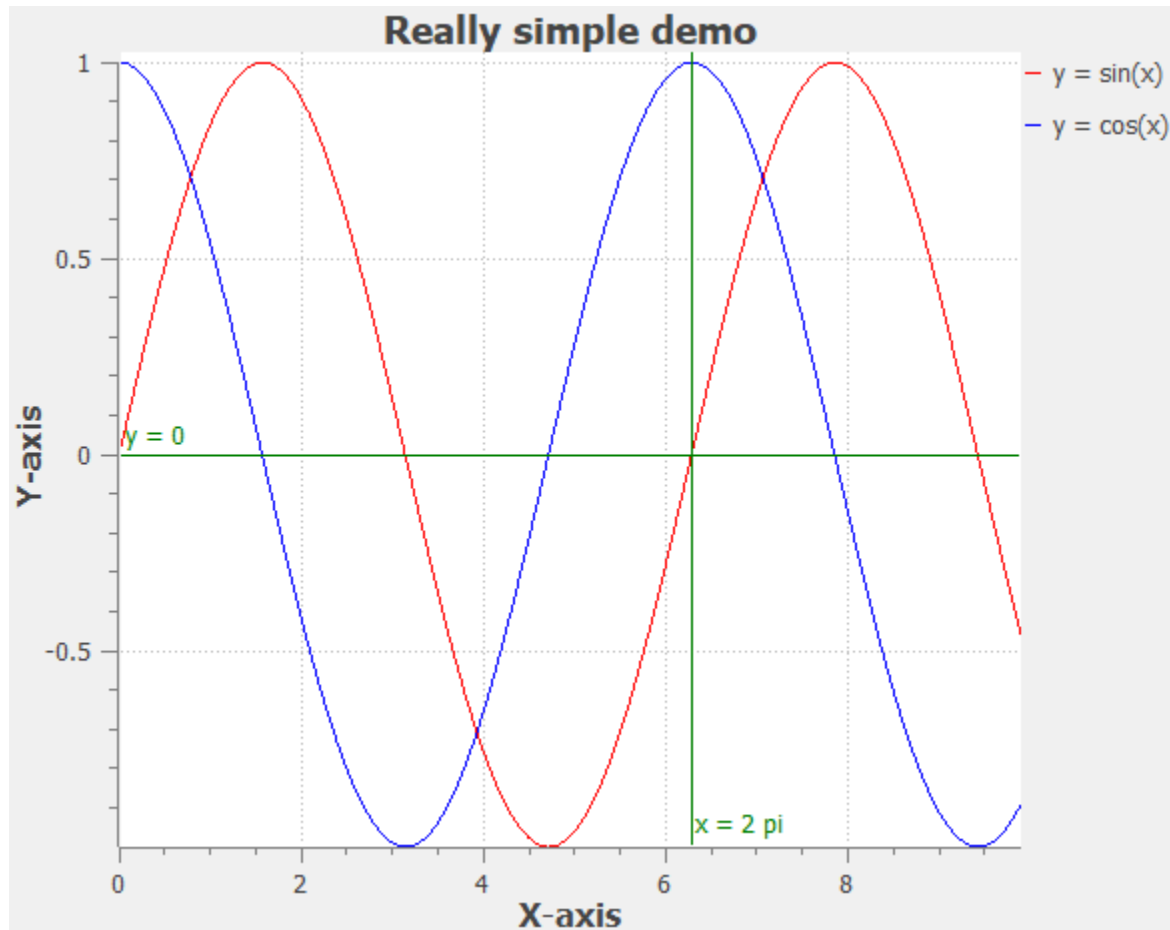
# create a plot widget for lists of Python floats
list_plot = QwtPlot(self)
list_plot.setTitle("Python list")
list_plot.setCanvasBackground(Qt.white)
list_plot.plotLayout().setCanvasMargin(0)
list_plot.plotLayout().setAlignCanvasToScales(True)
x = drange(0.0, 10.0, 0.01)
y = [lorentzian(item) for item in x]
# insert a curve, make it red and copy the lists
list_curve = QwtPlotCurve("y = lorentzian(x)")
list_curve.attach(list_plot)
list_curve.setPen(QPen(Qt.red))
list_curve.setData(x, y)
layout.addWidget(list_plot, 0, 1)
list_plot.replot()

def test_multidemo():
    """Multiple plot demo"""
    utils.test_widget(MultiDemo, size=(400, 300))

if __name__ == "__main__":
    test_multidemo()

```

4.2.15 Really simple demo



```
import os

import numpy as np
from qtpy.QtCore import Qt, QTimer

import qwt
from qwt.tests import utils

FNAMES = ("test_simple.svg", "test_simple.pdf", "test_simple.png")

class SimplePlot(qwt.QwtPlot):
    TEST_EXPORT = True

    def __init__(self):
        qwt.QwtPlot.__init__(self)
        self.setTitle("Really simple demo")
        self.insertLegend(qwt.QwtLegend(), qwt.QwtPlot.RightLegend)
        self.setAxisTitle(qwt.QwtPlot.xBottom, "X-axis")
        self.setAxisTitle(qwt.QwtPlot.yLeft, "Y-axis")
```

(continues on next page)

(continued from previous page)

```

self.enableAxis(self.xBottom)
self.setCanvasBackground(Qt.white)
canvas = self.canvas()
canvas.setBorderRadius(50)

qwt.QwtPlotGrid.make(self, color=Qt.lightGray, width=0, style=Qt.DotLine)

# insert a few curves
x = np.arange(0.0, 10.0, 0.1)
qwt.QwtPlotCurve.make(x, np.sin(x), "y = sin(x)", self, linecolor="red")
qwt.QwtPlotCurve.make(x, np.cos(x), "y = cos(x)", self, linecolor="blue")

# insert a horizontal marker at y = 0
qwt.QwtPlotMarker.make(
    label="y = 0",
    align=Qt.AlignRight | Qt.AlignTop,
    linestyle=qwt.QwtPlotMarker.HLine,
    color="darkGreen",
    plot=self,
)

# insert a vertical marker at x = 2 pi
qwt.QwtPlotMarker.make(
    xvalue=2 * np.pi,
    label="x = 2 pi",
    align=Qt.AlignRight | Qt.AlignTop,
    linestyle=qwt.QwtPlotMarker.VLine,
    color="darkGreen",
    plot=self,
)

if self.TEST_EXPORT and utils.TestEnvironment().unattended:
    QTimer.singleShot(0, self.export_to_different_formats)

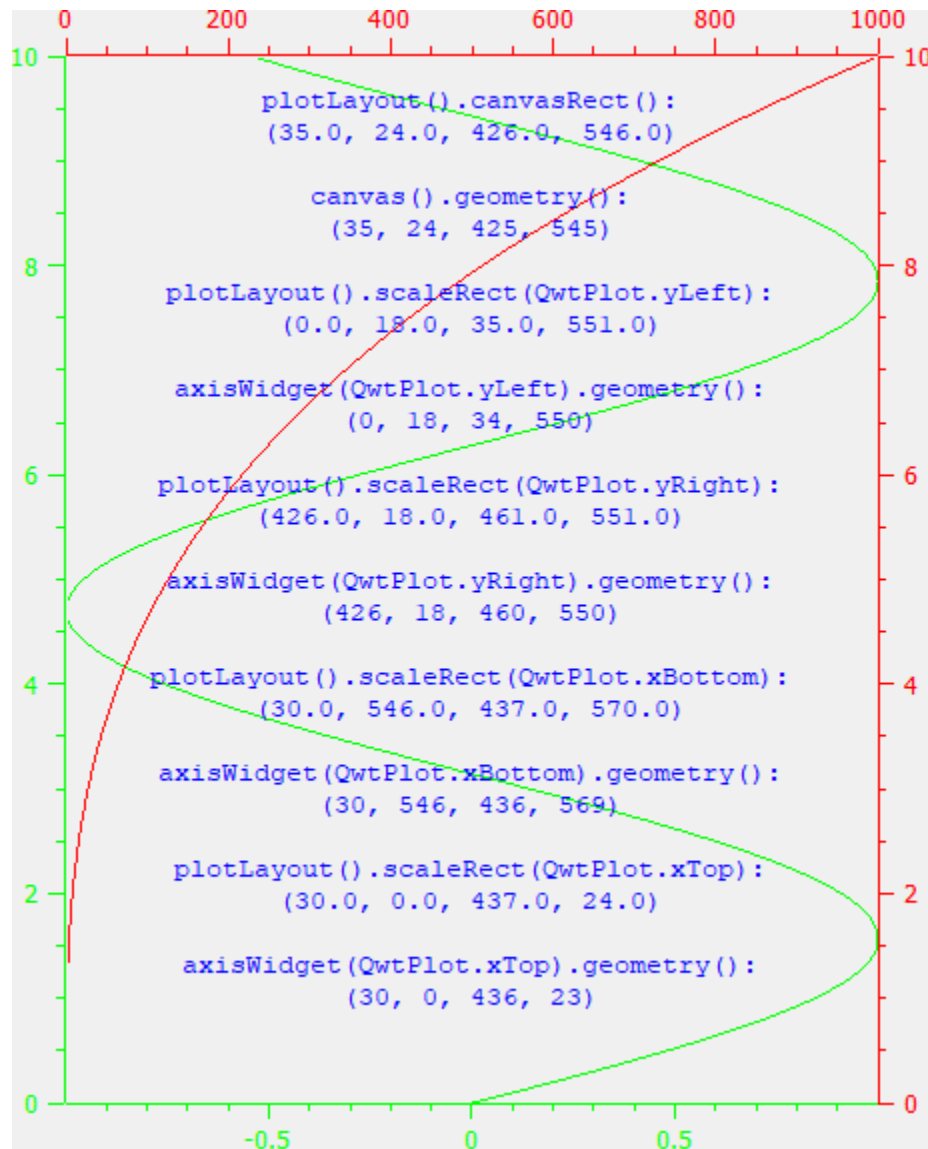
def export_to_different_formats(self):
    for fname in FNAMES:
        self.exportTo(fname)

def test_simple():
    """Simple plot example"""
    utils.test_widget(SimplePlot, size=(600, 400))
    for fname in FNAMES:
        if os.path.isfile(fname):
            os.remove(fname)

if __name__ == "__main__":
    test_simple()

```

4.2.16 Vertical plot demo



```
import numpy as np
from qtpy.QtCore import Qt
from qtpy.QtGui import QColor, QPalette, QPen

from qwt import QwtPlot, QwtPlotCurve, QwtPlotMarker, QwtText
from qwt.tests import utils

class VerticalPlot(QwtPlot):
    def __init__(self, parent=None):
        super(VerticalPlot, self).__init__(parent)
        self.setWindowTitle("PythonQwt")
        self.enableAxis(self.xTop, True)
```

(continues on next page)

(continued from previous page)

```

self.enableAxis(self.yRight, True)
y = np.linspace(0, 10, 500)
curve1 = QwtPlotCurve.make(np.sin(y), y, title="Test Curve 1")
curve2 = QwtPlotCurve.make(y**3, y, title="Test Curve 2")
curve2.setAxes(self.xTop, self.yRight)

for item, col, xa, ya in (
    (curve1, Qt.green, self.xBottom, self.yLeft),
    (curve2, Qt.red, self.xTop, self.yRight),
):
    item.attach(self)
    item.setPen(QPen(col))
    for axis_id in xa, ya:
        palette = self.axisWidget(axis_id).palette()
        palette.setColor(QPalette.WindowText, QColor(col))
        palette.setColor(QPalette.Text, QColor(col))
        self.axisWidget(axis_id).setPalette(palette)
        ticks_font = self.axisFont(axis_id)
        self.setAxisFont(axis_id, ticks_font)

self.marker = QwtPlotMarker.make(0, 5, plot=self)

def resizeEvent(self, event):
    super(VerticalPlot, self).resizeEvent(event)
    self.show_layout_details()

def show_layout_details(self):
    text = (
        "plotLayout().canvasRect():\n%r\n\n"
        "canvas().geometry():\n%r\n\n"
        "plotLayout().scaleRect(QwtPlot.yLeft):\n%r\n\n"
        "axisWidget(QwtPlot.yLeft).geometry():\n%r\n\n"
        "plotLayout().scaleRect(QwtPlot.yRight):\n%r\n\n"
        "axisWidget(QwtPlot.yRight).geometry():\n%r\n\n"
        "plotLayout().scaleRect(QwtPlot.xBottom):\n%r\n\n"
        "axisWidget(QwtPlot.xBottom).geometry():\n%r\n\n"
        "plotLayout().scaleRect(QwtPlot.xTop):\n%r\n\n"
        "axisWidget(QwtPlot.xTop).geometry():\n%r\n\n"
        % (
            self.plotLayout().canvasRect().getCoords(),
            self.canvas().geometry().getCoords(),
            self.plotLayout().scaleRect(QwtPlot.yLeft).getCoords(),
            self.axisWidget(QwtPlot.yLeft).geometry().getCoords(),
            self.plotLayout().scaleRect(QwtPlot.yRight).getCoords(),
            self.axisWidget(QwtPlot.yRight).geometry().getCoords(),
            self.plotLayout().scaleRect(QwtPlot.xBottom).getCoords(),
            self.axisWidget(QwtPlot.xBottom).geometry().getCoords(),
            self.plotLayout().scaleRect(QwtPlot.xTop).getCoords(),
            self.axisWidget(QwtPlot.xTop).geometry().getCoords(),
        )
    )
    self.marker.setLabel(QwtText.make(text, family="Courier New", color=Qt.blue))

```

(continues on next page)

(continued from previous page)

```
def test_vertical():  
    """Vertical plot example"""  
    utils.test_widget(VerticalPlot, size=(300, 650))  
  
if __name__ == "__main__":  
    test_vertical()
```

REFERENCE

Public API:

5.1 Plot widget fundamentals

5.1.1 QwtPlot

class qwt.plot.QwtPlot(*args)

A 2-D plotting widget

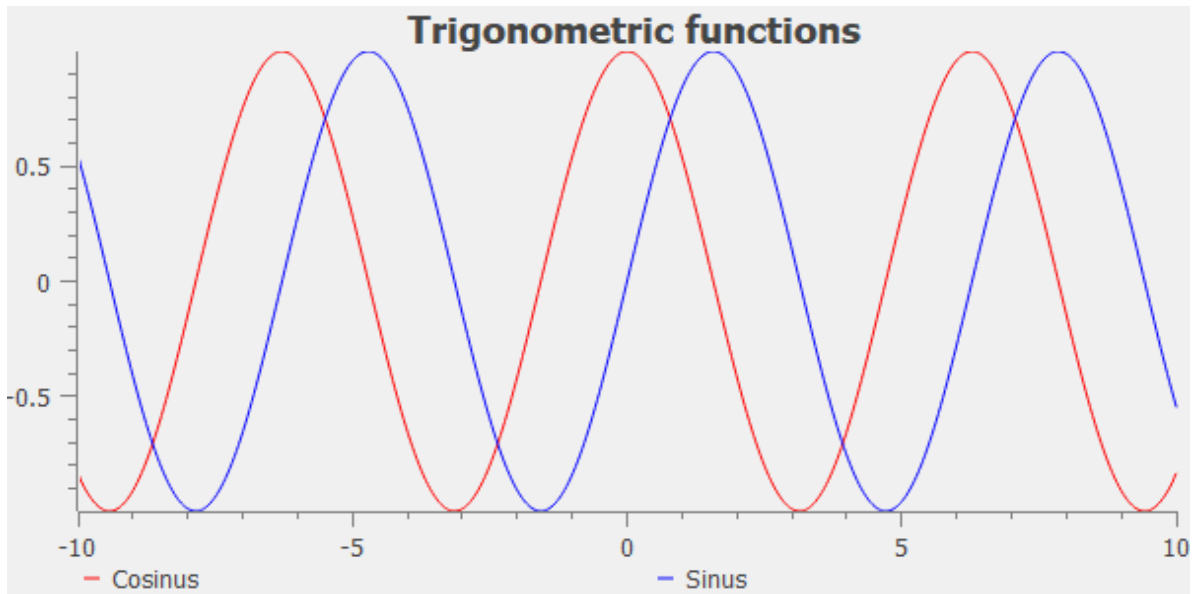
QwtPlot is a widget for plotting two-dimensional graphs. An unlimited number of plot items can be displayed on its canvas. Plot items might be curves (*qwt.plot_curve.QwtPlotCurve*), markers (*qwt.plot_marker.QwtPlotMarker*), the grid (*qwt.plot_grid.QwtPlotGrid*), or anything else derived from *QwtPlotItem*.

A plot can have up to four axes, with each plot item attached to an x- and a y axis. The scales at the axes can be explicitly set (*QwtScaleDiv*), or are calculated from the plot items, using algorithms (*QwtScaleEngine*) which can be configured separately for each axis.

The following example is a good starting point to see how to set up a plot widget:

```
from qtpy import QtWidgets as QW
import qwt
import numpy as np

app = QW.QApplication([])
x = np.linspace(-10, 10, 500)
plot = qwt.QwtPlot("Trigonometric functions")
plot.insertLegend(qwt.QwtLegend(), qwt.QwtPlot.BottomLegend)
qwt.QwtPlotCurve.make(x, np.cos(x), "Cosinus", plot, linecolor="red",
    ↪ antialiased=True)
qwt.QwtPlotCurve.make(x, np.sin(x), "Sinus", plot, linecolor="blue",
    ↪ antialiased=True)
plot.resize(600, 300)
plot.show()
```



```
class QwtPlot([title=""],[parent=None])
```

Parameters

- **title** (*str*) – Title text
- **parent** (*QWidget*) – Parent widget

itemAttached

A signal indicating, that an item has been attached/detached

Parameters

- **plotItem** – Plot item
- **on** – Attached/Detached

legendDataChanged

A signal with the attributes how to update the legend entries for a plot item.

Parameters

- **itemInfo** – Info about a plot item, build from `itemToInfo()`
- **data** – Attributes of the entries (usually ≤ 1) for the plot item.

insertItem(*item*)

Insert a plot item

Parameters

item (`qwt.plot.QwtPlotItem`) – PlotItem

See also:

`removeItem()`

Note: This was a member of `QwtPlotDict` in older versions.

removeItem(*item*)

Remove a plot item

Parameters

item (`qwt.plot.QwtPlotItem`) – PlotItem

See also:

[`insertItem\(\)`](#)

Note: This was a member of QwtPlotDict in older versions.

detachItems(*rtti=None*)

Detach items from the dictionary

Parameters

rtti (*int or None*) – In case of `QwtPlotItem.Rtti_PlotItem` or None (default) detach all items otherwise only those items of the type rtti.

Note: This was a member of QwtPlotDict in older versions.

itemList(*rtti=None*)

A list of attached plot items.

Use caution when iterating these lists, as removing/detaching an item will invalidate the iterator. Instead you can place pointers to objects to be removed in a removal list, and traverse that list later.

Parameters

rtti (*int*) – In case of `QwtPlotItem.Rtti_PlotItem` detach all items otherwise only those items of the type rtti.

Returns

List of all attached plot items of a specific type. If rtti is None, return a list of all attached plot items.

Note: This was a member of QwtPlotDict in older versions.

setFlatStyle(*state*)

Set or reset the flatStyle option

If the flatStyle option is set, the plot will be rendered without any margin (scales, canvas, layout).

Enabling this option makes the plot look flat and compact.

The flatStyle option is set to True by default.

Parameters

state (*bool*) – True or False.

See also:

[`flatStyle\(\)`](#)

flatStyle()**Returns**

True if the flatStyle option is set.

See also:

[`setFlatStyle\(\)`](#)

initAxesData()

Initialize axes

axisWidget(*axisId*)

Parameters

axisId (*int*) – Axis index

Returns

Scale widget of the specified axis, or None if axisId is invalid.

setAxisScaleEngine(*axisId*, *scaleEngine*)

Change the scale engine for an axis

Parameters

- **axisId** (*int*) – Axis index
- **scaleEngine** (`qwt.scale_engine.QwtScaleEngine`) – Scale engine

See also:

[`axisScaleEngine\(\)`](#)

axisScaleEngine(*axisId*)

Parameters

axisId (*int*) – Axis index

Returns

Scale engine for a specific axis

See also:

[`setAxisScaleEngine\(\)`](#)

axisAutoScale(*axisId*)

Parameters

axisId (*int*) – Axis index

Returns

True, if autoscaling is enabled

axisEnabled(*axisId*)

Parameters

axisId (*int*) – Axis index

Returns

True, if a specified axis is enabled

axisFont(*axisId*)

Parameters

axisId (*int*) – Axis index

Returns

The font of the scale labels for a specified axis

axisMaxMajor(*axisId*)**Parameters****axisId** (*int*) – Axis index**Returns**

The maximum number of major ticks for a specified axis

See also:`setAxisMaxMajor(), qwt.scale_engine.QwtScaleEngine.divideScale()`**axisMaxMinor**(*axisId*)**Parameters****axisId** (*int*) – Axis index**Returns**

The maximum number of minor ticks for a specified axis

See also:`setAxisMaxMinor(), qwt.scale_engine.QwtScaleEngine.divideScale()`**axisScaleDiv**(*axisId*)**Parameters****axisId** (*int*) – Axis index**Returns**

The scale division of a specified axis

`axisScaleDiv(axisId).lowerBound(), axisScaleDiv(axisId).upperBound()` are the current limits of the axis scale.**See also:**`qwt.scale_div.QwtScaleDiv, setAxisScaleDiv(), qwt.scale_engine.QwtScaleEngine.divideScale()`**axisScaleDraw**(*axisId*)**Parameters****axisId** (*int*) – Axis index**Returns**

Specified scaleDraw for axis, or NULL if axis is invalid.

axisStepSize(*axisId*)**Parameters****axisId** (*int*) – Axis index**Returns**

step size parameter value

This doesn't need to be the step size of the current scale.

See also:`setAxisScale(), qwt.scale_engine.QwtScaleEngine.divideScale()`

axisMargin(*axisId*)

Parameters

axisId (*int*) – Axis index

Returns

Margin in % of the canvas size

See also:

[`setAxisMargin\(\)`](#)

axisInterval(*axisId*)

Parameters

axisId (*int*) – Axis index

Returns

The current interval of the specified axis

This is only a convenience function for `axisScaleDiv(axisId).interval()`

See also:

[`qwt.scale_div.QwtScaleDiv`](#), [`axisScaleDiv\(\)`](#)

axisTitle(*axisId*)

Parameters

axisId (*int*) – Axis index

Returns

Title of a specified axis

enableAxis(*axisId*, *tf=True*)

Enable or disable a specified axis

When an axis is disabled, this only means that it is not visible on the screen. Curves, markers and can be attached to disabled axes, and transformation of screen coordinates into values works as normal.

Only xBottom and yLeft are enabled by default.

Parameters

- **axisId** (*int*) – Axis index
- **tf** (*bool*) – True (enabled) or False (disabled)

invTransform(*axisId*, *pos*)

Transform the x or y coordinate of a position in the drawing region into a value.

Parameters

- **axisId** (*int*) – Axis index
- **pos** (*int*) – position

Warning: The position can be an x or a y coordinate, depending on the specified axis.

transform(*axisId*, *value*)

Transform a value into a coordinate in the plotting region

Parameters

- **axisId** (*int*) – Axis index
- **value** (*float*) – Value

Returns

X or Y coordinate in the plotting region corresponding to the value.

setAxisFont(*axisId*, *font*)

Change the font of an axis

Parameters

- **axisId** (*int*) – Axis index
- **font** (*QFont*) – Font

Warning: This function changes the font of the tick labels, not of the axis title.

setAxisAutoScale(*axisId*, *on=True*)

Enable autoscaling for a specified axis

This member function is used to switch back to autoscaling mode after a fixed scale has been set. Autoscaling is enabled by default.

Parameters

- **axisId** (*int*) – Axis index
- **on** (*bool*) – On/Off

See also:

[`setAxisScale\(\)`](#), [`setAxisScaleDiv\(\)`](#), [`updateAxes\(\)`](#)

Note: The autoscaling flag has no effect until `updateAxes()` is executed (called by `replot()`).

setAxisScale(*axisId*, *min_*, *max_*, *stepSize=0*)

Disable autoscaling and specify a fixed scale for a selected axis.

In `updateAxes()` the scale engine calculates a scale division from the specified parameters, that will be assigned to the scale widget. So updates of the scale widget usually happen delayed with the next `replot`.

Parameters

- **axisId** (*int*) – Axis index
- **min** (*float*) – Minimum of the scale
- **max** (*float*) – Maximum of the scale
- **stepSize** (*float*) – Major step size. If `step == 0`, the step size is calculated automatically using the `maxMajor` setting.

See also:

[`setAxisMaxMajor\(\)`](#), [`setAxisAutoScale\(\)`](#), [`axisStepSize\(\)`](#), [`qwt.scale_engine.QwtScaleEngine.divideScale\(\)`](#)

setAxisScaleDiv(*axisId*, *scaleDiv*)

Disable autoscaling and specify a fixed scale for a selected axis.

The scale division will be stored locally only until the next call of `updateAxes()`. So updates of the scale widget usually happen delayed with the next replot.

Parameters

- **axisId** (*int*) – Axis index
- **scaleDiv** (`qwt.scale_div.QwtScaleDiv`) – Scale division

See also:

`setAxisScale()`, `setAxisAutoScale()`

setAxisScaleDraw(*axisId*, *scaleDraw*)

Set a scale draw

Parameters

- **axisId** (*int*) – Axis index
- **scaleDraw** (`qwt.scale_draw.QwtScaleDraw`) – Object responsible for drawing scales.

By passing `scaleDraw` it is possible to extend `QwtScaleDraw` functionality and let it take place in `QwtPlot`. Please note that `scaleDraw` has to be created with `new` and will be deleted by the corresponding `QwtScale` member (like a child object).

See also:

`qwt.scale_draw.QwtScaleDraw`, `qwt.scale_widget.QwtScaleWidget`

Warning: The attributes of `scaleDraw` will be overwritten by those of the previous `QwtScaleDraw`.

setAxisLabelAlignment(*axisId*, *alignment*)

Change the alignment of the tick labels

Parameters

- **axisId** (*int*) – Axis index
- **alignment** (`Qt.Alignment`) – Or'd `Qt.AlignmentFlags`

See also:

`qwt.scale_draw.QwtScaleDraw.setLabelAlignment()`

setAxisLabelRotation(*axisId*, *rotation*)

Rotate all tick labels

Parameters

- **axisId** (*int*) – Axis index
- **rotation** (*float*) – Angle in degrees. When changing the label rotation, the label alignment might be adjusted too.

See also:

`setLabelRotation()`, `setAxisLabelAlignment()`

setAxisLabelAutoSize(*axisId*, *state*)

Set tick labels automatic size option (default: on)

Parameters

- **axisId** (*int*) – Axis index
- **state** (*bool*) – On/off

See also:

[`qwt.scale_draw.QwtScaleDraw.setLabelAutoSize\(\)`](#)

setAxisMaxMinor(*axisId*, *maxMinor*)

Set the maximum number of minor scale intervals for a specified axis

Parameters

- **axisId** (*int*) – Axis index
- **maxMinor** (*int*) – Maximum number of minor steps

See also:

[`axisMaxMinor\(\)`](#)

setAxisMaxMajor(*axisId*, *maxMajor*)

Set the maximum number of major scale intervals for a specified axis

Parameters

- **axisId** (*int*) – Axis index
- **maxMajor** (*int*) – Maximum number of major steps

See also:

[`axisMaxMajor\(\)`](#)

setAxisMargin(*axisId*, *margin*)

Set the margin of the scale widget

Parameters

- **axisId** (*int*) – Axis index
- **margin** (*float*) – Margin in % of the canvas size

See also:

[`axisMargin\(\)`](#)

setAxisTitle(*axisId*, *title*)

Change the title of a specified axis

Parameters

- **axisId** (*int*) – Axis index
- **title** ([`qwt.text.QwtText`](#) or *str*) – axis title

updateAxes()

Rebuild the axes scales

In case of autoscaling the boundaries of a scale are calculated from the bounding rectangles of all plot items, having the *QwtPlotItem.AutoScale* flag enabled (*QwtScaleEngine.autoScale()*). Then a scale division is calculated (*QwtScaleEngine.didvideScale()*) and assigned to scale widget.

When the scale boundaries have been assigned with `setAxisScale()` a scale division is calculated (`QwtScaleEngine.divideScale()`) for this interval and assigned to the scale widget.

When the scale has been set explicitly by `setAxisScaleDiv()` the locally stored scale division gets assigned to the scale widget.

The scale widget indicates modifications by emitting a `QwtScaleWidget.scaleDivChanged()` signal.

`updateAxes()` is usually called by `replot()`.

See also:

`setAxisAutoScale()`, `setAxisScale()`, `setAxisScaleDiv()`, `replot()`, `QwtPlotItem.boundingRect()`

setCanvas(*canvas*)

Set the drawing canvas of the plot widget.

The default canvas is a `QwtPlotCanvas`.

Parameters

canvas (`QWidget`) – Canvas Widget

See also:

`canvas()`

event(*self*, *e*: `QEvent` | `None`) → bool

eventFilter(*self*, *a0*: `QObject` | `None`, *a1*: `QEvent` | `None`) → bool

autoRefresh()

Replots the plot if `autoReplot()` is True.

setAutoReplot(*tf*=`True`)

Set or reset the autoReplot option

If the autoReplot option is set, the plot will be updated implicitly by manipulating member functions. Since this may be time-consuming, it is recommended to leave this option switched off and call `replot()` explicitly if necessary.

The autoReplot option is set to false by default, which means that the user has to call `replot()` in order to make changes visible.

Parameters

tf (`bool`) – True or False. Defaults to True.

See also:

`autoReplot()`

autoReplot()

Returns

True if the autoReplot option is set.

See also:

`setAutoReplot()`

setTitle(*title*)

Change the plot's title

Parameters

title (`str` or `qwt.text.QwtText`) – New title

See also:

title()

title()

Returns

Title of the plot

See also:

setTitle()

titleLabel()

Returns

Title label widget.

setFooter(*text*)

Change the text the footer

Parameters

text (*str* or `qwt.text.QwtText`) – New text of the footer

See also:

footer()

footer()

Returns

Text of the footer

See also:

setFooter()

footerLabel()

Returns

Footer label widget.

setPlotLayout(*layout*)

Assign a new plot layout

Parameters

layout (`qwt.plot_layout.QwtPlotLayout`) – Layout

See also:

plotLayout()

plotLayout()

Returns

the plot's layout

See also:

setPlotLayout()

legend()

Returns

the plot's legend

See also:

[*insertLegend\(\)*](#)

canvas()

Returns

the plot's canvas

sizeHint()

Returns

Size hint for the plot widget

See also:

[*minimumSizeHint\(\)*](#)

minimumSizeHint()

Returns

Return a minimum size hint

resizeEvent(*self*, *a0*: *QResizeEvent* | *None*)

replot()

Redraw the plot

If the *autoReplot* option is not set (which is the default) or if any curves are attached to raw data, the plot has to be refreshed explicitly in order to make changes visible.

See also:

[*updateAxes\(\)*](#), [*setAutoReplot\(\)*](#)

updateLayout()

Adjust plot content to its current size.

See also:

[*resizeEvent\(\)*](#)

getCanvasMarginsHint(*maps*, *canvasRect*)

Calculate the canvas margins

Parameters

- **maps** (*list*) – *QwtPlot.axisCnt* maps, mapping between plot and paint device coordinates
- **canvasRect** (*QRectF*) – Bounding rectangle where to paint

Plot items might indicate, that they need some extra space at the borders of the canvas by the *Qwt-PlotItem.Margins* flag.

See also:

[*updateCanvasMargins\(\)*](#), [*getCanvasMarginHint\(\)*](#)

updateCanvasMargins()

Update the canvas margins

Plot items might indicate, that they need some extra space at the borders of the canvas by the *Qwt-PlotItem.Margins* flag.

See also:

[`getCanvasMarginsHint\(\)`](#), [`QwtPlotItem.getCanvasMarginHint\(\)`](#)

drawCanvas(*painter*)

Redraw the canvas.

Parameters

painter (*QPainter*) – Painter used for drawing

Warning: `drawCanvas` calls `drawItems` what is also used for printing. Applications that like to add individual plot items better overload `drawItems()`

See also:

[`getCanvasMarginsHint\(\)`](#), [`QwtPlotItem.getCanvasMarginHint\(\)`](#)

drawItems(*painter*, *canvasRect*, *maps*)

Redraw the canvas.

Parameters

- **painter** (*QPainter*) – Painter used for drawing
- **canvasRect** (*QRectF*) – Bounding rectangle where to paint
- **maps** (*list*) – *QwtPlot.axisCnt* maps, mapping between plot and paint device coordinates

Note: Usually `canvasRect` is `contentsRect()` of the plot canvas. Due to a bug in Qt this rectangle might be wrong for certain frame styles (f.e *QFrame.Box*) and it might be necessary to fix the margins manually using *QWidget.setContentsMargins()*

canvasMap(*axisId*)

Parameters

axisId (*int*) – Axis

Returns

Map for the axis on the canvas. With this map pixel coordinates can translated to plot coordinates and vice versa.

See also:

[`qwt.scale_map.QwtScaleMap`](#), [`transform\(\)`](#), [`invTransform\(\)`](#)

setCanvasBackground(*brush*)

Change the background of the plotting area

Sets brush to *QPalette.Window* of all color groups of the palette of the canvas. Using *canvas().setPalette()* is a more powerful way to set these colors.

Parameters

brush (*QBrush*) – New background brush

See also:

[`canvasBackground\(\)`](#)

canvasBackground()

Returns

Background brush of the plotting area.

See also:

[*setCanvasBackground\(\)*](#)

axisValid(*axis_id*)

Parameters

axis_id (*int*) – Axis

Returns

True if the specified axis exists, otherwise False

insertLegend(*legend*, *pos*=None, *ratio*=-1)

Insert a legend

If the position legend is *QwtPlot.LeftLegend* or *QwtPlot.RightLegend* the legend will be organized in one column from top to down. Otherwise the legend items will be placed in a table with a best fit number of columns from left to right.

insertLegend() will set the plot widget as parent for the legend. The legend will be deleted in the destructor of the plot or when another legend is inserted.

Legends, that are not inserted into the layout of the plot widget need to connect to the legendDataChanged() signal. Calling updateLegend() initiates this signal for an initial update. When the application code wants to implement its own layout this also needs to be done for rendering plots to a document (see *QwtPlotRenderer*).

Parameters

- **legend** (*qwt.legend.QwtAbstractLegend*) – Legend
- **pos** (*QwtPlot.LegendPosition*) – The legend's position.
- **ratio** (*float*) – Ratio between legend and the bounding rectangle of title, canvas and axes

Note: For top/left position the number of columns will be limited to 1, otherwise it will be set to unlimited.

Note: The legend will be shrunk if it would need more space than the given ratio. The ratio is limited to [0.0 .. 1.0]. In case of <= 0.0 it will be reset to the default ratio. The default vertical/horizontal ratio is 0.33/0.5.

See also:

[*legend\(\)*](#), [*qwt.plot_layout.QwtPlotLayout.legendPosition\(\)*](#), [*qwt.plot_layout.QwtPlotLayout.setLegendPosition\(\)*](#)

updateLegend(*plotItem*=None)

If plotItem is None, emit *QwtPlot.legendDataChanged* for all plot item. Otherwise, emit the signal for passed plot item.

Parameters

plotItem (*qwt.plot.QwtPlotItem*) – Plot item

See also:

`QwtPlotItem.legendData()`, `QwtPlot.legendDataChanged`

updateLegendItems(*plotItem*, *legendData*)

Update all plot items interested in legend attributes

Call `QwtPlotItem.updateLegend()`, when the `QwtPlotItem.LegendInterest` flag is set.

Parameters

- **plotItem** (`qwt.plot.QwtPlotItem`) – Plot item
- **legendData** (*list*) – Entries to be displayed for the plot item (usually 1)

See also:

`QwtPlotItem.LegendInterest()`, `QwtPlotItem.updateLegend()`

attachItem(*plotItem*, *on*)

Attach/Detach a plot item

Parameters

- **plotItem** (`qwt.plot.QwtPlotItem`) – Plot item
- **on** (*bool*) – When true attach the item, otherwise detach it

print_(*printer*)

Print plot to printer

Parameters

printer (*QPaintDevice or QPrinter or QSvgGenerator*) – Printer

exportTo(*filename*, *size*=(800, 600), *size_mm*=None, *resolution*=85, *format_*=None)

Export plot to PDF or image file (SVG, PNG, ...)

Parameters

- **filename** (*str*) – Filename
- **size** (*tuple*) – (width, height) size in pixels
- **size_mm** (*tuple*) – (width, height) size in millimeters
- **resolution** (*int*) – Resolution in dots per Inch (dpi)
- **format** (*str*) – File format (PDF, SVG, PNG, ...)

5.1.2 QwtPlotItem

class `qwt.plot.QwtPlotItem`(*title*=None, *icon*=None)

Base class for items on the plot canvas

A plot item is “something”, that can be painted on the plot canvas, or only affects the scales of the plot widget. They can be categorized as:

- Representator

A “Representator” is an item that represents some sort of data on the plot canvas. The different representator classes are organized according to the characteristics of the data:

- `qwt.plot_marker.QwtPlotMarker`: Represents a point or a horizontal/vertical coordinate
- `qwt.plot_curve.QwtPlotCurve`: Represents a series of points

- Decorators

A “Decorator” is an item, that displays additional information, that is not related to any data:

– `qwt.plot_grid.QwtPlotGrid`

Depending on the *QwtPlotItem.ItemAttribute* flags, an item is included into autoscaling or has an entry on the legend.

Before misusing the existing item classes it might be better to implement a new type of plot item (don’t implement a watermark as spectrogram). Deriving a new type of *QwtPlotItem* primarily means to implement the *YourPlotItem.draw()* method.

See also:

The cpuplot example shows the implementation of additional plot items.

class `QwtPlotItem`(`[title=None]`)

Constructor

Parameters

title (`qwt.text.QwtText` or `str`) – Title of the item

attach(`plot`)

Attach the item to a plot.

This method will attach a *QwtPlotItem* to the *QwtPlot* argument. It will first detach the *QwtPlotItem* from any plot from a previous call to attach (if necessary). If a `None` argument is passed, it will detach from any *QwtPlot* it was attached to.

Parameters

plot (`qwt.plot.QwtPlot`) – Plot widget

See also:

`detach()`

detach()

Detach the item from a plot.

This method detaches a *QwtPlotItem* from any *QwtPlot* it has been associated with.

See also:

`attach()`

rtti()

Return rtti for the specific class represented. *QwtPlotItem* is simply a virtual interface class, and base classes will implement this method with specific rtti values so a user can differentiate them.

Returns

rtti value

plot()

Returns

attached plot

z()

Plot items are painted in increasing z-order.

Returns

item z order

See also:

`setZ()`, `QwtPlotDict.itemList()`

setZ(*z*)

Set the z value

Plot items are painted in increasing z-order.

Parameters

z (*float*) – Z-value

See also:

`z()`, `QwtPlotDict.itemList()`

setTitle(*title*)

Set a new title

Parameters

title (`qwt.text.QwtText` or *str*) – Title

See also:

`title()`

title()

Returns

Title of the item

See also:

`setTitle()`

setIcon(*icon*)

Set item icon

Parameters

icon (*QIcon*) – Icon

See also:

`icon()`

icon()

Returns

Icon of the item

See also:

`setIcon()`

setItemAttribute(*attribute*, *on=True*)

Toggle an item attribute

Parameters

- **attribute** (*int*) – Attribute type
- **on** (*bool*) – True/False

See also:

`testItemAttribute()`

testItemAttribute(*attribute*)

Test an item attribute

Parameters

attribute (*int*) – Attribute type

Returns

True/False

See also:

[*setItemAttribute\(\)*](#)

setItemInterest(*interest, on=True*)

Toggle an item interest

Parameters

- **attribute** (*int*) – Interest type
- **on** (*bool*) – True/False

See also:

[*testItemInterest\(\)*](#)

testItemInterest(*interest*)

Test an item interest

Parameters

attribute (*int*) – Interest type

Returns

True/False

See also:

[*setItemInterest\(\)*](#)

setRenderHint(*hint, on=True*)

Toggle a render hint

Parameters

- **hint** (*int*) – Render hint
- **on** (*bool*) – True/False

See also:

[*testRenderHint\(\)*](#)

testRenderHint(*hint*)

Test a render hint

Parameters

attribute (*int*) – Render hint

Returns

True/False

See also:

[*setRenderHint\(\)*](#)

setLegendIconSize(*size*)

Set the size of the legend icon

The default setting is 8x8 pixels

Parameters

size (*QSize*) – Size

See also:

[*legendIconSize\(\)*](#), [*legendIcon\(\)*](#)

legendIconSize()**Returns**

Legend icon size

See also:

[*setLegendIconSize\(\)*](#), [*legendIcon\(\)*](#)

legendIcon(*index*, *size*)**Parameters**

- **index** (*int*) – Index of the legend entry (usually there is only one)
- **size** (*QSizeF*) – Icon size

Returns

Icon representing the item on the legend

The default implementation returns an invalid icon

See also:

[*setLegendIconSize\(\)*](#), [*legendData\(\)*](#)

show()

Show the item

hide()

Hide the item

setVisible(*on*)

Show/Hide the item

Parameters

on (*bool*) – Show if True, otherwise hide

See also:

[*isVisible\(\)*](#), [*show\(\)*](#), [*hide\(\)*](#)

isVisible()**Returns**

True if visible

See also:

[*setVisible\(\)*](#), [*show\(\)*](#), [*hide\(\)*](#)

itemChanged()

Update the legend and call *QwtPlot.autoRefresh()* for the parent plot.

See also:

QwtPlot.legendChanged(), *QwtPlot.autoRefresh()*

legendChanged()

Update the legend of the parent plot.

See also:

QwtPlot.updateLegend(), *itemChanged()*

setAxes(xAxis, yAxis)

Set X and Y axis

The item will painted according to the coordinates of its Axes.

Parameters

- **xAxis** (*int*) – X Axis (*QwtPlot.xBottom* or *QwtPlot.xTop*)
- **yAxis** (*int*) – Y Axis (*QwtPlot.yLeft* or *QwtPlot.yRight*)

See also:

setXAxis(), *setYAxis()*, *xAxis()*, *yAxis()*

setAxis(xAxis, yAxis)

Set X and Y axis

Warning: *setAxis* has been removed in Qwt6: please use *setAxes()* instead

setXAxis(axis)

Set the X axis

The item will painted according to the coordinates its Axes.

Parameters

axis (*int*) – X Axis (*QwtPlot.xBottom* or *QwtPlot.xTop*)

See also:

setAxes(), *setYAxis()*, *xAxis()*, *yAxis()*

setYAxis(axis)

Set the Y axis

The item will painted according to the coordinates its Axes.

Parameters

axis (*int*) – Y Axis (*QwtPlot.yLeft* or *QwtPlot.yRight*)

See also:

setAxes(), *setXAxis()*, *xAxis()*, *yAxis()*

xAxis()**Returns**

xAxis

yAxis()**Returns**

yAxis

boundingRect()**Returns**

An invalid bounding rect: QRectF(1.0, 1.0, -2.0, -2.0)

Note: A width or height < 0.0 is ignored by the autoscaler

getCanvasMarginHint(xMap, yMap, canvasRect)

Calculate a hint for the canvas margin

When the QwtPlotItem::Margins flag is enabled the plot item indicates, that it needs some margins at the borders of the canvas. This is f.e. used by bar charts to reserve space for displaying the bars.

The margins are in target device coordinates (pixels on screen)

Parameters

- **xMap** (`qwt.scale_map.QwtScaleMap`) – Maps x-values into pixel coordinates.
- **yMap** (`qwt.scale_map.QwtScaleMap`) – Maps y-values into pixel coordinates.
- **canvasRect** (`QRectF`) – Contents rectangle of the canvas in painter coordinates

See also:

`QwtPlot.getCanvasMarginsHint()`, `QwtPlot.updateCanvasMargins()`,

legendData()

Return all information, that is needed to represent the item on the legend

`QwtLegendData` is basically a list of `QVariants` that makes it possible to overload and reimplement `legendData()` to return almost any type of information, that is understood by the receiver that acts as the legend.

The default implementation returns one entry with the `title()` of the item and the `legendIcon()`.

Returns

Data, that is needed to represent the item on the legend

See also:

`title()`, `legendIcon()`, `qwt.legend.QwtLegend`

updateLegend(item, data)

Update the item to changes of the legend info

Plot items that want to display a legend (not those, that want to be displayed on a legend !) will have to implement `updateLegend()`.

`updateLegend()` is only called when the `LegendInterest` interest is enabled. The default implementation does nothing.

Parameters

- **item** (`qwt.plot.QwtPlotItem`) – Plot item to be displayed on a legend
- **data** (`list`) – Attributes how to display item on the legend

Note: Plot items, that want to be displayed on a legend need to enable the *QwtPlotItem.Legend* flag and to implement `legendData()` and `legendIcon()`

scaleRect(*xMap*, *yMap*)

Calculate the bounding scale rectangle of 2 maps

Parameters

- **xMap** (`qwt.scale_map.QwtScaleMap`) – Maps x-values into pixel coordinates.
- **yMap** (`qwt.scale_map.QwtScaleMap`) – Maps y-values into pixel coordinates.

Returns

Bounding scale rect of the scale maps, not normalized

paintRect(*xMap*, *yMap*)

Calculate the bounding paint rectangle of 2 maps

Parameters

- **xMap** (`qwt.scale_map.QwtScaleMap`) – Maps x-values into pixel coordinates.
- **yMap** (`qwt.scale_map.QwtScaleMap`) – Maps y-values into pixel coordinates.

Returns

Bounding paint rectangle of the scale maps, not normalized

5.1.3 QwtPlotCanvas

class `qwt.plot_canvas.QwtPlotCanvas`(*plot=None*)

Canvas of a QwtPlot.

Canvas is the widget where all plot items are displayed

See also:

`qwt.plot.QwtPlot.setCanvas()`

Paint attributes:

- *QwtPlotCanvas.BackingStore*:

Paint double buffered reusing the content of the pixmap buffer when possible.

Using a backing store might improve the performance significantly, when working with widget overlays (like rubber bands). Disabling the cache might improve the performance for incremental paints (using `qwt.plot_directpainter.QwtPlotDirectPainter`).

- *QwtPlotCanvas.Opaque*:

Try to fill the complete contents rectangle of the plot canvas

When using styled backgrounds Qt assumes, that the canvas doesn't fill its area completely (f.e because of rounded borders) and fills the area below the canvas. When this is done with gradients it might result in a serious performance bottleneck - depending on the size.

When the Opaque attribute is enabled the canvas tries to identify the gaps with some heuristics and to fill those only.

Warning: Will not work for semitransparent backgrounds

- *QwtPlotCanvas.HackStyledBackground:*

Try to improve painting of styled backgrounds

QwtPlotCanvas supports the box model attributes for customizing the layout with style sheets. Unfortunately the design of Qt style sheets has no concept how to handle backgrounds with rounded corners - beside of padding.

When *HackStyledBackground* is enabled the plot canvas tries to separate the background from the background border by reverse engineering to paint the background before and the border after the plot items. In this order the border gets perfectly antialiased and you can avoid some pixel artifacts in the corners.

- *QwtPlotCanvas.ImmediatePaint:*

When *ImmediatePaint* is set *replot()* calls *repaint()* instead of *update()*.

See also:

[*replot\(\)*](#), [*QWidget.repaint\(\)*](#), [*QWidget.update\(\)*](#)

Focus indicators:

- *QwtPlotCanvas.NoFocusIndicator:*

Don't paint a focus indicator

- *QwtPlotCanvas.CanvasFocusIndicator:*

The focus is related to the complete canvas. Paint the focus indicator using *paintFocus()*

- *QwtPlotCanvas.ItemFocusIndicator:*

The focus is related to an item (curve, point, ...) on the canvas. It is up to the application to display a focus indication using f.e. highlighting.

class *QwtPlotCanvas*(*[plot=None]*)

Constructor

Parameters

plot ([*qwt.plot.QwtPlot*](#)) – Parent plot widget

See also:

[*qwt.plot.QwtPlot.setCanvas\(\)*](#)

plot()

Returns

Parent plot widget

setPaintAttribute(*attribute, on=True*)

Changing the paint attributes

Paint attributes:

- *QwtPlotCanvas.BackingStore*
- *QwtPlotCanvas.Opaque*
- *QwtPlotCanvas.HackStyledBackground*
- *QwtPlotCanvas.ImmediatePaint*

Parameters

- **attribute** (*int*) – Paint attribute
- **on** (*bool*) – On/Off

See also:

[*testPaintAttribute\(\)*](#), [*backingStore\(\)*](#)

testPaintAttribute(*attribute*)

Test whether a paint attribute is enabled

Parameters

- **attribute** (*int*) – Paint attribute

Returns

True, when attribute is enabled

See also:

[*setPaintAttribute\(\)*](#)

backingStore()**Returns**

Backing store, might be None

invalidateBackingStore()

Invalidate the internal backing store

setFocusIndicator(*focusIndicator*)

Set the focus indicator

Focus indicators:

- *QwtPlotCanvas.NoFocusIndicator*
- *QwtPlotCanvas.CanvasFocusIndicator*
- *QwtPlotCanvas.ItemFocusIndicator*

Parameters

- **focusIndicator** (*int*) – Focus indicator

See also:

[*focusIndicator\(\)*](#)

focusIndicator()**Returns**

Focus indicator

See also:

[*setFocusIndicator\(\)*](#)

setBorderRadius(*radius*)

Set the radius for the corners of the border frame

Parameters

- **radius** (*float*) – Radius of a rounded corner

See also:

[*borderRadius\(\)*](#)

borderRadius()

Returns

Radius for the corners of the border frame

See also:

[*setBorderRadius\(\)*](#)

event(*self*, *e*: *QEvent* | *None*) → bool

paintEvent(*self*, *a0*: *QPaintEvent* | *None*)

drawBorder(*painter*)

Draw the border of the plot canvas

Parameters

painter (*QPainter*) – Painter

See also:

[*setBorderRadius\(\)*](#)

resizeEvent(*self*, *a0*: *QResizeEvent* | *None*)

drawFocusIndicator(*painter*)

Draw the focus indication

Parameters

painter (*QPainter*) – Painter

replot()

Invalidate the paint cache and repaint the canvas

updateStyleSheetInfo()

Update the cached information about the current style sheet

borderPath(*rect*)

Calculate the painter path for a styled or rounded border

When the canvas has no styled background or rounded borders the painter path is empty.

Parameters

rect (*QRect*) – Bounding rectangle of the canvas

Returns

Painter path, that can be used for clipping

5.2 Plot items

5.2.1 QwtPlotGrid

class `qwt.plot_grid.QwtPlotGrid`(*title='Grid'*)

A class which draws a coordinate grid

The *QwtPlotGrid* class can be used to draw a coordinate grid. A coordinate grid consists of major and minor vertical and horizontal grid lines. The locations of the grid lines are determined by the X and Y scale divisions which can be assigned with *setXDiv()* and *setYDiv()*. The *draw()* member draws the grid within a bounding rectangle.

classmethod **make**(*plot=None, z=None, enablemajor=None, enableminor=None, color=None, width=None, style=None, mincolor=None, minwidth=None, minstyle=None*)

Create and setup a new *QwtPlotGrid* object (convenience function).

Parameters

- **plot** (`qwt.plot.QwtPlot` or *None*) – Plot to attach the curve to
- **z** (*float* or *None*) – Z-value
- **enablemajor** (*bool* or *None*) – Tuple of two boolean values (x, y) for enabling major grid lines
- **enableminor** (*bool* or *None*) – Tuple of two boolean values (x, y) for enabling minor grid lines
- **color** (*QColor* or *str* or *None*) – Pen color for both major and minor grid lines (default: *Qt.gray*)
- **width** (*float* or *None*) – Pen width for both major and minor grid lines (default: 1.0)
- **style** (*Qt.PenStyle* or *None*) – Pen style for both major and minor grid lines (default: *Qt.DotLine*)
- **mincolor** (*QColor* or *str* or *None*) – Pen color for minor grid lines only (default: *Qt.gray*)
- **minwidth** (*float* or *None*) – Pen width for minor grid lines only (default: 1.0)
- **minstyle** (*Qt.PenStyle* or *None*) – Pen style for minor grid lines only (default: *Qt.DotLine*)

See also:

[*setMinorPen\(\)*](#), [*setMajorPen\(\)*](#)

rtti()

Returns

Return *QwtPlotItem.Rtti_PlotGrid*

enableX(*on*)

Enable or disable vertical grid lines

Parameters

on (*bool*) – Enable (true) or disable

See also:

[*enableXMin\(\)*](#)

enableY(*on*)

Enable or disable horizontal grid lines

Parameters

on (*bool*) – Enable (true) or disable

See also:

[`enableYMin\(\)`](#)

enableXMin(*on*)

Enable or disable minor vertical grid lines.

Parameters

on (*bool*) – Enable (true) or disable

See also:

[`enableX\(\)`](#)

enableYMin(*on*)

Enable or disable minor horizontal grid lines.

Parameters

on (*bool*) – Enable (true) or disable

See also:

[`enableY\(\)`](#)

setXDiv(*scaleDiv*)

Assign an x axis scale division

Parameters

scaleDiv ([`qwt.scale_div.QwtScaleDiv`](#)) – Scale division

setYDiv(*scaleDiv*)

Assign an y axis scale division

Parameters

scaleDiv ([`qwt.scale_div.QwtScaleDiv`](#)) – Scale division

setPen(args*)**

Build and/or assign a pen for both major and minor grid lines

setPen(*color*, *width*, *style*)

Build and assign a pen for both major and minor grid lines

In Qt5 the default pen width is 1.0 (0.0 in Qt4) what makes it non cosmetic (see [`QPen.isCosmetic\(\)`](#)). This method signature has been introduced to hide this incompatibility.

Parameters

- **color** ([`QColor`](#)) – Pen color
- **width** ([`float`](#)) – Pen width
- **style** ([`Qt.PenStyle`](#)) – Pen style

setPen(*pen*)

Assign a pen for both major and minor grid lines

Parameters

pen ([`QPen`](#)) – New pen

See also:

[`pen\(\)`](#), [`brush\(\)`](#)

setMajorPen(*args)

Build and/or assign a pen for both major grid lines

setMajorPen(color, width, style)

Build and assign a pen for both major grid lines

In Qt5 the default pen width is 1.0 (0.0 in Qt4) what makes it non cosmetic (see *QPen.isCosmetic()*). This method signature has been introduced to hide this incompatibility.

Parameters

- **color** (*QColor*) – Pen color
- **width** (*float*) – Pen width
- **style** (*Qt.PenStyle*) – Pen style

setMajorPen(pen)

Assign a pen for the major grid lines

Parameters

pen (*QPen*) – New pen

See also:

majorPen(), *setMinorPen()*, *setPen()*, *pen()*, *brush()*

setMinorPen(*args)

Build and/or assign a pen for both minor grid lines

setMinorPen(color, width, style)

Build and assign a pen for both minor grid lines

In Qt5 the default pen width is 1.0 (0.0 in Qt4) what makes it non cosmetic (see *QPen.isCosmetic()*). This method signature has been introduced to hide this incompatibility.

Parameters

- **color** (*QColor*) – Pen color
- **width** (*float*) – Pen width
- **style** (*Qt.PenStyle*) – Pen style

setMinorPen(pen)

Assign a pen for the minor grid lines

Parameters

pen (*QPen*) – New pen

See also:

minorPen(), *setMajorPen()*, *setPen()*, *pen()*, *brush()*

draw(painter, xMap, yMap, canvasRect)

Draw the grid

The grid is drawn into the bounding rectangle such that grid lines begin and end at the rectangle's borders. The X and Y maps are used to map the scale divisions into the drawing region screen.

Parameters

- **painter** (*QPainter*) – Painter
- **xMap** (*qwt.scale_map.QwtScaleMap*) – X axis map
- **yMap** (*qwt.scale_map.QwtScaleMap*) – Y axis
- **canvasRect** (*QRectF*) – Contents rectangle of the plot canvas

majorPen()**Returns**

the pen for the major grid lines

See also:

[*setMajorPen\(\)*](#), [*setMinorPen\(\)*](#), [*setPen\(\)*](#)

minorPen()**Returns**

the pen for the minor grid lines

See also:

[*setMinorPen\(\)*](#), [*setMajorPen\(\)*](#), [*setPen\(\)*](#)

xEnabled()**Returns**

True if vertical grid lines are enabled

See also:

[*enableX\(\)*](#)

yEnabled()**Returns**

True if horizontal grid lines are enabled

See also:

[*enableY\(\)*](#)

xMinEnabled()**Returns**

True if minor vertical grid lines are enabled

See also:

[*enableXMin\(\)*](#)

yMinEnabled()**Returns**

True if minor horizontal grid lines are enabled

See also:

[*enableYMin\(\)*](#)

xScaleDiv()**Returns**

the scale division of the x axis

yScaleDiv()**Returns**

the scale division of the y axis

updateScaleDiv(*xScaleDiv*, *yScaleDiv*)

Update the grid to changes of the axes scale division

Parameters

- **xMap** (`qwt.scale_map.QwtScaleMap`) – Scale division of the x-axis
- **yMap** (`qwt.scale_map.QwtScaleMap`) – Scale division of the y-axis

See also:

`updateAxes()`

5.2.2 QwtPlotCurve

class `qwt.plot_curve.QwtPlotCurve`(*title=None*)

A plot item, that represents a series of points

A curve is the representation of a series of points in the x-y plane. It supports different display styles and symbols.

See also:

`qwt.symbol.QwtSymbol()`, `qwt.scale_map.QwtScaleMap()`

Curve styles:

- *QwtPlotCurve.NoCurve*:
Don't draw a curve. Note: This doesn't affect the symbols.
- *QwtPlotCurve.Lines*:
Connect the points with straight lines.
- *QwtPlotCurve.Sticks*:
Draw vertical or horizontal sticks (depending on the `orientation()`) from a baseline which is defined by `setBaseline()`.
- *QwtPlotCurve.Steps*:
Connect the points with a step function. The step function is drawn from the left to the right or vice versa, depending on the `QwtPlotCurve::Inverted` attribute.
- *QwtPlotCurve.Dots*:
Draw dots at the locations of the data points. Note: This is different from a dotted line (see `setPen()`), and faster as a curve in `QwtPlotCurve::NoStyle` style and a symbol painting a point.
- *QwtPlotCurve.UserCurve*:
Styles `>= QwtPlotCurve.UserCurve` are reserved for derived classes of `QwtPlotCurve` that overload `drawCurve()` with additional application specific curve types.

Curve attributes:

- *QwtPlotCurve.Inverted*:
For *QwtPlotCurve.Steps* only. Draws a step function from the right to the left.

Legend attributes:

- *QwtPlotCurve.LegendNoAttribute*:
QwtPlotCurve tries to find a color representing the curve and paints a rectangle with it.

- *QwtPlotCurve.LegendShowLine*:

If the style() is not *QwtPlotCurve.NoCurve* a line is painted with the curve pen().

- *QwtPlotCurve.LegendShowSymbol*:

If the curve has a valid symbol it is painted.

- *QwtPlotCurve.LegendShowBrush*:

If the curve has a brush a rectangle filled with the curve brush() is painted.

class **QwtPlotCurve**([*title=None*])

Constructor

Parameters

title (*qwt.text.QwtText* or *str* or *None*) – Curve title

classmethod **make**(*xdata=None, ydata=None, title=None, plot=None, z=None, x_axis=None, y_axis=None, style=None, symbol=None, linecolor=None, linewidth=None, linestyle=None, antialiased=False, size=None, finite=None*)

Create and setup a new *QwtPlotCurve* object (convenience function).

Parameters

- **xdata** – List/array of x values
- **ydata** – List/array of y values
- **title** (*qwt.text.QwtText* or *str* or *None*) – Curve title
- **plot** (*qwt.plot.QwtPlot* or *None*) – Plot to attach the curve to
- **z** (*float* or *None*) – Z-value
- **x_axis** (*int* or *None*) – curve X-axis (default: *QwtPlot.yLeft*)
- **y_axis** (*int* or *None*) – curve Y-axis (default: *QwtPlot.xBottom*)
- **style** (*int* or *None*) – curve style (*QwtPlotCurve.NoCurve*, *QwtPlotCurve.Lines*, *QwtPlotCurve.Sticks*, *QwtPlotCurve.Steps*, *QwtPlotCurve.Dots*, *QwtPlotCurve.UserCurve*)
- **symbol** (*qwt.symbol.QwtSymbol* or *None*) – curve symbol
- **linecolor** (*QColor* or *str* or *None*) – curve line color
- **linewidth** (*float* or *None*) – curve line width
- **linestyle** (*Qt.PenStyle* or *None*) – curve pen style
- **antialiased** (*bool*) – if True, enable antialiasing rendering
- **size** (*int* or *None*) – size of xData and yData
- **finite** (*bool*) – if True, keep only finite array elements (remove all infinity and not a number values), otherwise do not filter array elements

See also:

setData(), *setPen()*, *attach()*

init()

Initialize internal members

rtti()

Returns

QwtPlotItem.Rtti_PlotCurve

setLegendAttribute(*attribute*, *on=True*)

Specify an attribute how to draw the legend icon

Legend attributes:

- *QwtPlotCurve.LegendNoAttribute*
- *QwtPlotCurve.LegendShowLine*
- *QwtPlotCurve.LegendShowSymbol*
- *QwtPlotCurve.LegendShowBrush*

Parameters

- **attribute** (*int*) – Legend attribute
- **on** (*bool*) – On/Off

See also:

[*testLegendAttribute\(\)*](#), [*legendIcon\(\)*](#)

testLegendAttribute(*attribute*)

Parameters

attribute (*int*) – Legend attribute

Returns

True, when attribute is enabled

See also:

[*setLegendAttribute\(\)*](#)

setStyle(*style*)

Set the curve's drawing style

Valid curve styles:

- *QwtPlotCurve.NoCurve*
- *QwtPlotCurve.Lines*
- *QwtPlotCurve.Sticks*
- *QwtPlotCurve.Steps*
- *QwtPlotCurve.Dots*
- *QwtPlotCurve.UserCurve*

Parameters

style (*int*) – Curve style

See also:

[*style\(\)*](#)

style()**Returns**

Style of the curve

See also:

[`setStyle\(\)`](#)

setSymbol(*symbol*)

Assign a symbol

The curve will take the ownership of the symbol, hence the previously set symbol will be delete by setting a new one. If symbol is None no symbol will be drawn.

Parameters

symbol ([`qwt.symbol.QwtSymbol`](#)) – Symbol

See also:

[`symbol\(\)`](#)

symbol()**Returns**

Current symbol or None, when no symbol has been assigned

See also:

[`setSymbol\(\)`](#)

setPen(*args)

Build and/or assign a pen, depending on the arguments.

setPen(*color*, *width*, *style*)

Build and assign a pen

In Qt5 the default pen width is 1.0 (0.0 in Qt4) what makes it non cosmetic (see [`QPen.isCosmetic\(\)`](#)). This method signature has been introduced to hide this incompatibility.

Parameters

- **color** ([`QColor`](#)) – Pen color
- **width** ([`float`](#)) – Pen width
- **style** ([`Qt.PenStyle`](#)) – Pen style

setPen(*pen*)

Assign a pen

Parameters

pen ([`QPen`](#)) – New pen

See also:

[`pen\(\)`](#), [`brush\(\)`](#)

pen()**Returns**

Pen used to draw the lines

See also:

[`setPen\(\)`](#), [`brush\(\)`](#)

setBrush(*brush*)

Assign a brush.

In case of *brush.style() != QBrush.NoBrush* and *style() != QwtPlotCurve.Sticks* the area between the curve and the baseline will be filled.

In case *not brush.color().isValid()* the area will be filled by *pen.color()*. The fill algorithm simply connects the first and the last curve point to the baseline. So the curve data has to be sorted (ascending or descending).

Parameters

brush (*QBrush* or *QColor*) – New brush

See also:

[*brush\(\)*](#), [*setBaseline\(\)*](#), [*baseline\(\)*](#)

brush()**Returns**

Brush used to fill the area between lines and the baseline

See also:

[*setBrush\(\)*](#), [*setBaseline\(\)*](#), [*baseline\(\)*](#)

directPaint(*from_*, *to*)

When observing a measurement while it is running, new points have to be added to an existing seriesItem. This method can be used to display them avoiding a complete redraw of the canvas.

Setting *plot().canvas().setAttribute(Qt.WA_PaintOutsidePaintEvent, True)* will result in faster painting, if the paint engine of the canvas widget supports this feature.

Parameters

- **from** (*int*) – Index of the first point to be painted
- **to** (*int*) – Index of the last point to be painted

See also:

[*drawSeries\(\)*](#)

drawSeries(*painter*, *xMap*, *yMap*, *canvasRect*, *from_*, *to*)

Draw an interval of the curve

Parameters

- **painter** (*QPainter*) – Painter
- **xMap** ([*qwt.scale_map.QwtScaleMap*](#)) – Maps x-values into pixel coordinates.
- **yMap** ([*qwt.scale_map.QwtScaleMap*](#)) – Maps y-values into pixel coordinates.
- **canvasRect** (*QRectF*) – Contents rectangle of the canvas
- **from** (*int*) – Index of the first point to be painted
- **to** (*int*) – Index of the last point to be painted. If *to* < 0 the curve will be painted to its last point.

See also:

[*drawCurve\(\)*](#), [*drawSymbols\(\)*](#)

drawCurve(*painter*, *style*, *xMap*, *yMap*, *canvasRect*, *from_*, *to*)

Draw the line part (without symbols) of a curve interval.

Parameters

- **painter** (*QPainter*) – Painter
- **style** (*int*) – curve style, see *QwtPlotCurve.CurveStyle*
- **xMap** (*qwt.scale_map.QwtScaleMap*) – Maps x-values into pixel coordinates.
- **yMap** (*qwt.scale_map.QwtScaleMap*) – Maps y-values into pixel coordinates.
- **canvasRect** (*QRectF*) – Contents rectangle of the canvas
- **from** (*int*) – Index of the first point to be painted
- **to** (*int*) – Index of the last point to be painted. If *to* < 0 the curve will be painted to its last point.

See also:

[draw\(\)](#), [drawDots\(\)](#), [drawLines\(\)](#), [drawSteps\(\)](#), [drawSticks\(\)](#)

drawLines(*painter*, *xMap*, *yMap*, *canvasRect*, *from_*, *to*)

Draw lines

Parameters

- **painter** (*QPainter*) – Painter
- **xMap** (*qwt.scale_map.QwtScaleMap*) – Maps x-values into pixel coordinates.
- **yMap** (*qwt.scale_map.QwtScaleMap*) – Maps y-values into pixel coordinates.
- **canvasRect** (*QRectF*) – Contents rectangle of the canvas
- **from** (*int*) – Index of the first point to be painted
- **to** (*int*) – Index of the last point to be painted. If *to* < 0 the curve will be painted to its last point.

See also:

[draw\(\)](#), [drawDots\(\)](#), [drawSteps\(\)](#), [drawSticks\(\)](#)

drawSticks(*painter*, *xMap*, *yMap*, *canvasRect*, *from_*, *to*)

Draw sticks

Parameters

- **painter** (*QPainter*) – Painter
- **xMap** (*qwt.scale_map.QwtScaleMap*) – Maps x-values into pixel coordinates.
- **yMap** (*qwt.scale_map.QwtScaleMap*) – Maps y-values into pixel coordinates.
- **canvasRect** (*QRectF*) – Contents rectangle of the canvas
- **from** (*int*) – Index of the first point to be painted
- **to** (*int*) – Index of the last point to be painted. If *to* < 0 the curve will be painted to its last point.

See also:

[draw\(\)](#), [drawDots\(\)](#), [drawSteps\(\)](#), [drawLines\(\)](#)

drawDots(*painter*, *xMap*, *yMap*, *canvasRect*, *from_*, *to*)

Draw dots

Parameters

- **painter** (*QPainter*) – Painter
- **xMap** (*qwt.scale_map.QwtScaleMap*) – Maps x-values into pixel coordinates.
- **yMap** (*qwt.scale_map.QwtScaleMap*) – Maps y-values into pixel coordinates.
- **canvasRect** (*QRectF*) – Contents rectangle of the canvas
- **from** (*int*) – Index of the first point to be painted
- **to** (*int*) – Index of the last point to be painted. If to < 0 the curve will be painted to its last point.

See also:`draw()`, `drawSticks()`, `drawSteps()`, `drawLines()`**drawSteps**(*painter*, *xMap*, *yMap*, *canvasRect*, *from_*, *to*)

Draw steps

Parameters

- **painter** (*QPainter*) – Painter
- **xMap** (*qwt.scale_map.QwtScaleMap*) – Maps x-values into pixel coordinates.
- **yMap** (*qwt.scale_map.QwtScaleMap*) – Maps y-values into pixel coordinates.
- **canvasRect** (*QRectF*) – Contents rectangle of the canvas
- **from** (*int*) – Index of the first point to be painted
- **to** (*int*) – Index of the last point to be painted. If to < 0 the curve will be painted to its last point.

See also:`draw()`, `drawSticks()`, `drawDots()`, `drawLines()`**setCurveAttribute**(*attribute*, *on=True*)

Specify an attribute for drawing the curve

Supported curve attributes:

- *QwtPlotCurve.Inverted*

Parameters

- **attribute** (*int*) – Curve attribute
- **on** (*bool*) – On/Off

See also:`testCurveAttribute()`**testCurveAttribute**(*attribute*)**Returns**

True, if attribute is enabled

See also:

[`setCurveAttribute\(\)`](#)

fillCurve(*painter*, *xMap*, *yMap*, *canvasRect*, *polygon*)

Fill the area between the curve and the baseline with the curve brush

Parameters

- **painter** (*QPainter*) – Painter
- **xMap** ([`qwt.scale_map.QwtScaleMap`](#)) – Maps x-values into pixel coordinates.
- **yMap** ([`qwt.scale_map.QwtScaleMap`](#)) – Maps y-values into pixel coordinates.
- **canvasRect** (*QRectF*) – Contents rectangle of the canvas
- **polygon** (*QPolygonF*) – Polygon - will be modified !

See also:

[`setBrush\(\)`](#), [`setBaseline\(\)`](#), [`setStyle\(\)`](#)

closePolyline(*painter*, *xMap*, *yMap*, *polygon*)

Complete a polygon to be a closed polygon including the area between the original polygon and the baseline.

Parameters

- **painter** (*QPainter*) – Painter
- **xMap** ([`qwt.scale_map.QwtScaleMap`](#)) – Maps x-values into pixel coordinates.
- **yMap** ([`qwt.scale_map.QwtScaleMap`](#)) – Maps y-values into pixel coordinates.
- **polygon** (*QPolygonF*) – Polygon to be completed

drawSymbols(*painter*, *symbol*, *xMap*, *yMap*, *canvasRect*, *from_*, *to*)

Draw symbols

Parameters

- **painter** (*QPainter*) – Painter
- **symbol** ([`qwt.symbol.QwtSymbol`](#)) – Curve symbol
- **xMap** ([`qwt.scale_map.QwtScaleMap`](#)) – Maps x-values into pixel coordinates.
- **yMap** ([`qwt.scale_map.QwtScaleMap`](#)) – Maps y-values into pixel coordinates.
- **canvasRect** (*QRectF*) – Contents rectangle of the canvas
- **from** (*int*) – Index of the first point to be painted
- **to** (*int*) – Index of the last point to be painted. If *to* < 0 the curve will be painted to its last point.

See also:

[`setSymbol\(\)`](#), [`drawSeries\(\)`](#), [`drawCurve\(\)`](#)

setBaseline(*value*)

Set the value of the baseline

The baseline is needed for filling the curve with a brush or the Sticks drawing style.

The interpretation of the baseline depends on the *orientation()*. With *Qt.Horizontal*, the baseline is interpreted as a horizontal line at *y* = *baseline()*, with *Qt.Vertical*, it is interpreted as a vertical line at *x* = *baseline()*.

The default value is 0.0.

Parameters

value (*float*) – Value of the baseline

See also:

[*baseline\(\)*](#), [*setBrush\(\)*](#), [*setStyle\(\)*](#)

baseline()

Returns

Value of the baseline

See also:

[*setBaseline\(\)*](#)

closestPoint(*pos*)

Find the closest curve point for a specific position

Parameters

pos (*QPoint*) – Position, where to look for the closest curve point

Returns

tuple (*index*, *dist*)

dist is the distance between the position and the closest curve point. *index* is the index of the closest curve point, or -1 if none can be found (f.e when the curve has no points).

Note: *closestPoint()* implements a dumb algorithm, that iterates over all points

legendIcon(*index*, *size*)

Parameters

- **index** (*int*) – Index of the legend entry (ignored as there is only one)
- **size** (*QSizeF*) – Icon size

Returns

Icon representing the curve on the legend

See also:

[*qwt.plot.QwtPlotItem.setLegendIconSize\(\)*](#), [*qwt.plot.QwtPlotItem.legendData\(\)*](#)

setData(args*, ***kwargs*)**

Initialize data with a series data object or an array of points.

setData(*data*):

Parameters

data (*.plot_series.QwtSeriesData*) – Series data (e.g. *QwtPointArrayData* instance)

setData(*xData*, *yData*, [*size=None*], [*finite=True*]):

Initialize data with *x* and *y* arrays.

This signature was removed in Qwt6 and is temporarily maintained here to ensure compatibility with Qwt5.

Same as *setSamples(x, y, [size=None], [finite=True])*

Parameters

- **x** – List/array of x values
- **y** – List/array of y values
- **size** (*int* or *None*) – size of xData and yData
- **finite** (*bool*) – if True, keep only finite array elements (remove all infinity and not a number values), otherwise do not filter array elements

See also:

`setSamples()`

setSamples(*args, **kwargs)

Initialize data with an array of points.

setSamples(data):

Parameters

data (*.plot_series.QwtSeriesData*) – Series data (e.g. *QwtPointArrayData* instance)

setSamples(samples):

Same as *setData(QwtPointArrayData(samples))*

Parameters

samples – List/array of points

setSamples(xData, yData, [size=None], [finite=True]):

Same as *setData(QwtPointArrayData(xData, yData, [size=None]))*

Parameters

- **xData** – List/array of x values
- **yData** – List/array of y values
- **size** (*int* or *None*) – size of xData and yData
- **finite** (*bool*) – if True, keep only finite array elements (remove all infinity and not a number values), otherwise do not filter array elements

See also:

plot_series.QwtPointArrayData

5.2.3 QwtPlotMarker

class qwt.plot_marker.QwtPlotMarker(title=None)

A class for drawing markers

A marker can be a horizontal line, a vertical line, a symbol, a label or any combination of them, which can be drawn around a center point inside a bounding rectangle.

The *setSymbol()* member assigns a symbol to the marker. The symbol is drawn at the specified point.

With *setLabel()*, a label can be assigned to the marker. The *setLabelAlignment()* member specifies where the label is drawn. All the Align*-constants in *Qt.AlignmentFlags* (see Qt documentation) are valid. The interpretation of the alignment depends on the marker's line style. The alignment refers to the center point of the marker, which means, for example, that the label would be printed left above the center point if the alignment was set to *Qt.AlignLeft* | *Qt.AlignTop*.

Line styles:

- *QwtPlotMarker.NoLine*: No line
- *QwtPlotMarker.HLine*: A horizontal line
- *QwtPlotMarker.VLine*: A vertical line
- *QwtPlotMarker.Cross*: A crosshair

classmethod **make**(*xvalue=None, yvalue=None, title=None, label=None, symbol=None, plot=None, z=None, x_axis=None, y_axis=None, align=None, orientation=None, spacing=None, linestyle=None, color=None, width=None, style=None, antialiased=False*)

Create and setup a new *QwtPlotMarker* object (convenience function).

Parameters

- **xvalue** (*float* or *None*) – x position (optional, default: *None*)
- **yvalue** (*float* or *None*) – y position (optional, default: *None*)
- **title** (*qwt.text.QwtText* or *str* or *None*) – Marker title
- **label** (*qwt.text.QwtText* or *str* or *None*) – Label text
- **symbol** (*qwt.symbol.QwtSymbol* or *None*) – New symbol
- **plot** (*qwt.plot.QwtPlot* or *None*) – Plot to attach the curve to
- **z** (*float* or *None*) – Z-value
- **x_axis** (*int*) – curve X-axis (default: *QwtPlot.yLeft*)
- **y_axis** (*int*) – curve Y-axis (default: *QwtPlot.xBottom*)
- **align** (*Qt.Alignment* or *None*) – Alignment of the label
- **orientation** (*Qt.Orientation* or *None*) – Orientation of the label
- **spacing** (*int* or *None*) – Spacing (distance between the position and the label)
- **linestyle** (*int*) – Line style
- **color** (*QColor* or *str* or *None*) – Pen color
- **width** (*float*) – Pen width
- **style** (*Qt.PenStyle*) – Pen style
- **antialiased** (*bool*) – if True, enable antialiasing rendering

See also:

setData(), *setPen()*, *attach()*

rtti()

Returns

QwtPlotItem.Rtti_PlotMarker

value()

Returns

Value

xValue()

Returns

x Value

yValue()

Returns

y Value

setValue(*args)

Set Value

setValue(pos):

Parameters

pos (*QPointF*) – Position

setValue(x, y):

Parameters

- **x** (*float*) – x position
- **y** (*float*) – y position

setXValue(x)

Set X Value

Parameters

x (*float*) – x position

setYValue(y)

Set Y Value

Parameters

y (*float*) – y position

draw(painter, xMap, yMap, canvasRect)

Draw the marker

Parameters

- **painter** (*QPainter*) – Painter
- **xMap** (*qwt.scale_map.QwtScaleMap*) – x Scale Map
- **yMap** (*qwt.scale_map.QwtScaleMap*) – y Scale Map
- **canvasRect** (*QRectF*) – Contents rectangle of the canvas in painter coordinates

drawLines(painter, canvasRect, pos)

Draw the lines marker

Parameters

- **painter** (*QPainter*) – Painter
- **canvasRect** (*QRectF*) – Contents rectangle of the canvas in painter coordinates
- **pos** (*QPointF*) – Position of the marker, translated into widget coordinates

See also:

[*drawLabel\(\)*](#), [*qwt.symbol.QwtSymbol.drawSymbol\(\)*](#)

drawLabel(painter, canvasRect, pos)

Align and draw the text label of the marker

Parameters

- **painter** (*QPainter*) – Painter
- **canvasRect** (*QRectF*) – Contents rectangle of the canvas in painter coordinates
- **pos** (*QPointF*) – Position of the marker, translated into widget coordinates

See also:

[*drawLabel\(\)*](#), [*qwt.symbol.QwtSymbol.drawSymbol\(\)*](#)

setLineStyle(*style*)

Set the line style

Parameters

style (*int*) – Line style

Line styles:

- *QwtPlotMarker.NoLine*: No line
- *QwtPlotMarker.HLine*: A horizontal line
- *QwtPlotMarker.VLine*: A vertical line
- *QwtPlotMarker.Cross*: A crosshair

See also:

[*lineStyle\(\)*](#)

lineStyle()**Returns**

the line style

See also:

[*setLineStyle\(\)*](#)

setSymbol(*symbol*)

Assign a symbol

Parameters

symbol (*qwt.symbol.QwtSymbol*) – New symbol

See also:

[*symbol\(\)*](#)

symbol()**Returns**

the symbol

See also:

[*setSymbol\(\)*](#)

setLabel(*label*)

Set the label

Parameters

label (*qwt.text.QwtText* or *str*) – Label text

See also:

[*label\(\)*](#)

label()**Returns**

the label

See also:

[*setLabel\(\)*](#)

setLabelAlignment(*align*)

Set the alignment of the label

In case of *QwtPlotMarker.HLine* the alignment is relative to the y position of the marker, but the horizontal flags correspond to the canvas rectangle. In case of *QwtPlotMarker.VLine* the alignment is relative to the x position of the marker, but the vertical flags correspond to the canvas rectangle.

In all other styles the alignment is relative to the marker's position.

Parameters

align (*Qt.Alignment*) – Alignment

See also:

[*labelAlignment\(\)*](#), [*labelOrientation\(\)*](#)

labelAlignment()**Returns**

the label alignment

See also:

[*setLabelAlignment\(\)*](#), [*setLabelOrientation\(\)*](#)

setLabelOrientation(*orientation*)

Set the orientation of the label

When orientation is *Qt.Vertical* the label is rotated by 90.0 degrees (from bottom to top).

Parameters

orientation (*Qt.Orientation*) – Orientation of the label

See also:

[*labelOrientation\(\)*](#), [*setLabelAlignment\(\)*](#)

labelOrientation()**Returns**

the label orientation

See also:

[*setLabelOrientation\(\)*](#), [*labelAlignment\(\)*](#)

setSpacing(*spacing*)

Set the spacing

When the label is not centered on the marker position, the spacing is the distance between the position and the label.

Parameters

spacing (*int*) – Spacing

See also:

[*spacing\(\)*](#), [*setLabelAlignment\(\)*](#)

spacing()**Returns**

the spacing

See also:

[`setSpacing\(\)`](#)

setLinePen(*args)

Build and/or assign a line pen, depending on the arguments.

setLinePen(color, width, style)

Build and assign a line pen

In Qt5 the default pen width is 1.0 (0.0 in Qt4) what makes it non cosmetic (see `QPen.isCosmetic()`). This method signature has been introduced to hide this incompatibility.

Parameters

- **color** (`QColor`) – Pen color
- **width** (`float`) – Pen width
- **style** (`Qt.PenStyle`) – Pen style

setLinePen(pen)

Specify a pen for the line.

Parameters

pen (`QPen`) – New pen

See also:

`pen()`, `brush()`

linePen()

Returns

the line pen

See also:

[`setLinePen\(\)`](#)

boundingRect()

Returns

An invalid bounding rect: `QRectF(1.0, 1.0, -2.0, -2.0)`

Note: A width or height < 0.0 is ignored by the autoscaler

legendIcon(index, size)

Parameters

- **index** (`int`) – Index of the legend entry (ignored as there is only one)
- **size** (`QSizeF`) – Icon size

Returns

Icon representing the marker on the legend

See also:

[`qwt.plot.QwtPlotItem.setLegendIconSize\(\)`](#), [`qwt.plot.QwtPlotItem.legendData\(\)`](#)

5.3 Additional plot features

5.3.1 QwtLegend

class `qwt.legend.QwtLegendData`

Attributes of an entry on a legend

QwtLegendData is an abstract container (like *QAbstractModel*) to exchange attributes, that are only known between to the plot item and the legend.

By overloading *QwtPlotItem.legendData()* any other set of attributes could be used, that can be handled by a modified (or completely different) implementation of a legend.

See also:

[*qwt.legend.QwtLegend*](#)

Note: The stockchart example implements a legend as a tree with checkable items

setValues(*map_*)

Set the legend attributes

Parameters

map (*dict*) – Values

See also:

[*values\(\)*](#)

values()

Returns

Legend attributes

See also:

[*setValues\(\)*](#)

hasRole(*role*)

Parameters

role (*int*) – Attribute role

Returns

True, when the internal map has an entry for role

setValue(*role*, *data*)

Set an attribute value

Parameters

- **role** (*int*) – Attribute role
- **data** (*QVariant*) – Attribute value

See also:

[*value\(\)*](#)

value(*role*)

Parameters

role (*int*) – Attribute role

Returns

Attribute value for a specific role

See also:

[`setValue\(\)`](#)

isValid()

Returns

True, when the internal map is empty

title()

Returns

Value of the TitleRole attribute

icon()

Returns

Value of the IconRole attribute

mode()

Returns

Value of the ModeRole attribute

class `qwt.legend.QwtLegendLabel` (*parent=None*)

A widget representing something on a QwtLegend.

setData(*legendData*)

Set the attributes of the legend label

Parameters

legendData (`QwtLegendData`) – Attributes of the label

See also:

[`data\(\)`](#)

data()

Returns

Attributes of the label

See also:

[`setData\(\)`](#), [`qwt.plot.QwtPlotItem.legendData\(\)`](#)

setText(*text*)

Set the text to the legend item

Parameters

text (`qwt.text.QwtText`) – Text label

See also:

[`text\(\)`](#)

setItemMode(*mode*)

Set the item mode. The default is *QwtLegendData.ReadOnly*.

Parameters

mode (*int*) – Item mode

See also:

[*itemMode\(\)*](#)

itemMode()**Returns**

Item mode

See also:

[*setItemMode\(\)*](#)

setIcon(*icon*)

Assign the icon

Parameters

icon (*QPixmap*) – QPixmap representing a plot item

See also:

[*icon\(\)*](#), [*qwt.plot.QwtPlotItem.legendIcon\(\)*](#)

icon()**Returns**

QPixmap representing a plot item

See also:

[*setIcon\(\)*](#)

setSpacing(*spacing*)

Change the spacing between icon and text

Parameters

spacing (*int*) – Spacing

See also:

[*spacing\(\)*](#), [*qwt.text.QwtTextLabel.margin\(\)*](#)

spacing()**Returns**

Spacing between icon and text

See also:

[*setSpacing\(\)*](#)

setChecked(*on*)

Check/Uncheck a the item

Parameters

on (*bool*) – check/uncheck

See also:

[*isChecked\(\)*](#), [*setItemMode\(\)*](#)

isChecked()

Returns

true, if the item is checked

See also:

[*setChecked\(\)*](#)

setDown(*down*)

Set the item being down

Parameters

on (*bool*) – true, if the item is down

See also:

[*isDown\(\)*](#)

isDown()

Returns

true, if the item is down

See also:

[*setDown\(\)*](#)

sizeHint()

Returns

a size hint

paintEvent (*self*, *a0*: *QPaintEvent* | *None*)

mousePressEvent (*self*, *a0*: *QMouseEvent* | *None*)

mouseReleaseEvent (*self*, *a0*: *QMouseEvent* | *None*)

keyPressEvent (*self*, *a0*: *QKeyEvent* | *None*)

keyReleaseEvent (*self*, *a0*: *QKeyEvent* | *None*)

class `qwt.legend.QwtLegend`(*parent=None*)

The legend widget

The QwtLegend widget is a tabular arrangement of legend items. Legend items might be any type of widget, but in general they will be a QwtLegendLabel.

See also:

`:py:class`qwt.legend.QwtLegendLabel``, `:py:class`qwt.plot.QwtPlotItem``, `:py:class`qwt.plot.QwtPlot``

class `QwtLegend`(*[parent=None]*)

Constructor

Parameters

parent (*QWidget*) – Parent widget

clicked

A signal which is emitted when the user has clicked on a legend label, which is in *QwtLegendData.Clickable* mode.

Parameters

- **itemInfo** – Info for the item of the selected legend item
- **index** – Index of the legend label in the list of widgets that are associated with the plot item

Note: Clicks are disabled as default

checked

A signal which is emitted when the user has clicked on a legend label, which is in *QwtLegendData.Checkable* mode

Parameters

- **itemInfo** – Info for the item of the selected legend label
- **index** – Index of the legend label in the list of widgets that are associated with the plot item
- **on** – True when the legend label is checked

Note: Clicks are disabled as default

setMaxColumns(*numColumns*)

Set the maximum number of entries in a row

F.e when the maximum is set to 1 all items are aligned vertically. 0 means unlimited

Parameters

numColumns (*int*) – Maximum number of entries in a row

See also:

`maxColumns()`, `QwtDynGridLayout.setMaxColumns()`

maxColumns()

Returns

Maximum number of entries in a row

See also:

`setMaxColumns()`, `QwtDynGridLayout.maxColumns()`

setDefaultItemMode(*mode*)

Set the default mode for legend labels

Legend labels will be constructed according to the attributes in a *QwtLegendData* object. When it doesn't contain a value for the *QwtLegendData.ModeRole* the label will be initialized with the default mode of the legend.

Parameters

mode (*int*) – Default item mode

See also:

`itemMode()`, `QwtLegendData.value()`, `QwtPlotItem::legendData()`

... note:

Changing the mode doesn't have any effect on existing labels.

defaultItemMode()

Returns

Default item mode

See also:

[*setDefaultItemMode\(\)*](#)

contentsWidget()

The contents widget is the only child of the viewport of the internal *QScrollArea* and the parent widget of all legend items.

Returns

Container widget of the legend items

horizontalScrollBar()

Returns

Horizontal scrollbar

See also:

[*verticalScrollBar\(\)*](#)

verticalScrollBar()

Returns

Vertical scrollbar

See also:

[*horizontalScrollBar\(\)*](#)

updateLegend(*itemInfo*, *data*)

Update the entries for an item

Parameters

- **itemInfo** (*QVariant*) – Info for an item
- **data** (*list*) – Default item mode

createWidget(*data*)

Create a widget to be inserted into the legend

The default implementation returns a *QwtLegendLabel*.

Parameters

data (*QwtLegendData*) – Attributes of the legend entry

Returns

Widget representing data on the legend

... note:

updateWidget() will called soon after createWidget()
with the same attributes.

updateWidget(*widget*, *data*)

Update the widget

Parameters

- **widget** (*QWidget*) – Usually a *QwtLegendLabel*
- **data** (*QwtLegendData*) – Attributes to be displayed

See also:

createWidget()

... note:

When widget **is** no *QwtLegendLabel* *updateWidget()* does nothing.

sizeHint()

Return a size hint

heightForWidth(*width*)

Parameters

width (*int*) – Width

Returns

The preferred height, for a width.

eventFilter(*object_*, *event*)

Handle *QEvent.ChildRemoved* and *QEvent.LayoutRequest* events for the *contentsWidget()*.

Parameters

- **object** (*QObject*) – Object to be filtered
- **event** (*QEvent*) – Event

Returns

Forwarded to *QwtAbstractLegend.eventFilter()*

renderLegend(*painter*, *rect*, *fillBackground*)

Render the legend into a given rectangle.

Parameters

- **painter** (*QPainter*) – Painter
- **rect** (*QRectF*) – Bounding rectangle
- **fillBackground** (*bool*) – When true, fill rect with the widget background

renderItem(*painter*, *widget*, *rect*, *fillBackground*)

Render a legend entry into a given rectangle.

Parameters

- **painter** (*QPainter*) – Painter
- **widget** (*QWidget*) – Widget representing a legend entry
- **rect** (*QRectF*) – Bounding rectangle
- **fillBackground** (*bool*) – When true, fill rect with the widget background

legendWidgets(*itemInfo*)

List of widgets associated to a item

Parameters

itemInfo (*QVariant*) – Info about an item

legendWidget(*itemInfo*)

First widget in the list of widgets associated to an item

Parameters

itemInfo (*QVariant*) – Info about an item

itemInfo(*widget*)

Find the item that is associated to a widget

Parameters

widget (*QWidget*) – Widget on the legend

Returns

Associated item info

5.3.2 Color maps

QwtColorMap

class qwt.color_map.**QwtColorMap**(*format_=None*)

QwtColorMap is used to map values into colors.

For displaying 3D data on a 2D plane the 3rd dimension is often displayed using colors, like f.e in a spectrogram.

Each color map is optimized to return colors for only one of the following image formats:

- *QImage.Format_Indexed8*
- *QImage.Format_ARGB32*

class **QwtColorMap**(*format_*)

Parameters

format (*int*) – Preferred format of the color map (QwtColorMap.RGB or QwtColorMap.Indexed)

See also:

qwt.QwtScaleWidget

color(*interval*, *value*)

Map a value into a color

Parameters

- **interval** (*qwt.interval.QwtInterval*) – valid interval for value
- **value** (*float*) – value

Returns

the color corresponding to value

Warning: This method is slow for Indexed color maps. If it is necessary to map many values, its better to get the color table once and find the color using *colorIndex()*.

colorTable(*interval*)

Build and return a color map of 256 colors

Parameters

interval (`qwt.interval.QwtInterval`) – range for the values

Returns

a color table, that can be used for a *QImage*

The color table is needed for rendering indexed images in combination with using *colorIndex()*.

QwtLinearColorMap

class `qwt.color_map.QwtLinearColorMap(*args)`

Build a linear color map with two stops.

class `QwtLinearColorMap(format_)`

Build a color map with two stops at 0.0 and 1.0. The color at 0.0 is *Qt.blue*, at 1.0 it is *Qt.yellow*.

Parameters

format (*int*) – Preferred format of the color map (`QwtColorMap.RGB` or `QwtColorMap.Indexed`)

`QwtLinearColorMap(color1, color2, [format_=QwtColorMap.RGB]):`

Build a color map with two stops at 0.0 and 1.0.

Parameters

- **color1** (*QColor*) – color at 0.
- **color2** (*QColor*) – color at 1.
- **format** (*int*) – Preferred format of the color map (`QwtColorMap.RGB` or `QwtColorMap.Indexed`)

`QwtLinearColorMap.setMode(mode)`

Set the mode of the color map

Parameters

mode (*int*) – `QwtLinearColorMap.FixedColors` or `QwtLinearColorMap.ScaledColors`

FixedColors means the color is calculated from the next lower color stop. *ScaledColors* means the color is calculated by interpolating the colors of the adjacent stops.

`QwtLinearColorMap.mode()`

Returns

the mode of the color map

See also:

`QwtLinearColorMap.setMode()`

QwtAlphaColorMap

class qwt.color_map.QwtAlphaColorMap(*color*)

QwtAlphaColorMap varies the alpha value of a color

class QwtAlphaColorMap(*color*)

Build a color map varying the alpha value of a color.

Parameters

color (*QColor*) – color of the map

setColor(*color*)

Set the color of the map

Parameters

color (*QColor*) – color of the map

color()

Returns

the color of the map

See also:

[*QwtAlphaColorMap.setColor\(\)*](#)

5.3.3 QwtPlotRenderer

class qwt.plot_renderer.QwtPlotRenderer(*parent=None*)

Renderer for exporting a plot to a document, a printer or anything else, that is supported by QPainter/QPaintDevice

Discard flags:

- *QwtPlotRenderer.DiscardNone*: Render all components of the plot
- *QwtPlotRenderer.DiscardBackground*: Don't render the background of the plot
- *QwtPlotRenderer.DiscardTitle*: Don't render the title of the plot
- *QwtPlotRenderer.DiscardLegend*: Don't render the legend of the plot
- *QwtPlotRenderer.DiscardCanvasBackground*: Don't render the background of the canvas
- *QwtPlotRenderer.DiscardFooter*: Don't render the footer of the plot
- *QwtPlotRenderer.DiscardCanvasFrame*: Don't render the frame of the canvas

Note: The *QwtPlotRenderer.DiscardCanvasFrame* flag has no effect when using style sheets, where the frame is part of the background

Layout flags:

- *QwtPlotRenderer.DefaultLayout*: Use the default layout as on screen
- *QwtPlotRenderer.FrameWithScales*: Instead of the scales a box is painted around the plot canvas, where the scale ticks are aligned to.

setDiscardFlag(*flag*, *on=True*)

Change a flag, indicating what to discard from rendering

Parameters

- **flag** (*int*) – Flag to change
- **on** (*bool*) – On/Off

See also:

[*testDiscardFlag\(\)*](#), [*setDiscardFlags\(\)*](#), [*discardFlags\(\)*](#)

testDiscardFlag(*flag*)

Parameters

flag (*int*) – Flag to be tested

Returns

True, if flag is enabled.

See also:

[*setDiscardFlag\(\)*](#), [*setDiscardFlags\(\)*](#), [*discardFlags\(\)*](#)

setDiscardFlags(*flags*)

Set the flags, indicating what to discard from rendering

Parameters

flags (*int*) – Flags

See also:

[*testDiscardFlag\(\)*](#), [*setDiscardFlag\(\)*](#), [*discardFlags\(\)*](#)

discardFlags()

Returns

Flags, indicating what to discard from rendering

See also:

[*setDiscardFlag\(\)*](#), [*setDiscardFlags\(\)*](#), [*testDiscardFlag\(\)*](#)

setLayoutFlag(*flag*, *on=True*)

Change a layout flag

Parameters

flag (*int*) – Flag to change

See also:

[*testLayoutFlag\(\)*](#), [*setLayoutFlags\(\)*](#), [*layoutFlags\(\)*](#)

testLayoutFlag(*flag*)

Parameters

flag (*int*) – Flag to be tested

Returns

True, if flag is enabled.

See also:

[*setLayoutFlag\(\)*](#), [*setLayoutFlags\(\)*](#), [*layoutFlags\(\)*](#)

setLayoutFlags(*flags*)

Set the layout flags

Parameters

flags (*int*) – Flags

See also:

[`setLayoutFlag\(\)`](#), [`testLayoutFlag\(\)`](#), [`layoutFlags\(\)`](#)

layoutFlags()**Returns**

Layout flags

See also:

[`setLayoutFlags\(\)`](#), [`setLayoutFlag\(\)`](#), [`testLayoutFlag\(\)`](#)

renderDocument(*plot*, *filename*, *sizeMM*=(300, 200), *resolution*=85, *format_*=None)

Render a plot to a file

The format of the document will be auto-detected from the suffix of the file name.

Parameters

- **plot** ([`qwt.plot.QwtPlot`](#)) – Plot widget
- **fileName** (*str*) – Path of the file, where the document will be stored
- **sizeMM** (*QSizeF*) – Size for the document in millimeters
- **resolution** (*int*) – Resolution in dots per Inch (dpi)

renderTo(*plot*, *dest*)

Render a plot to a file

Supported formats are:

- pdf: Portable Document Format PDF
- ps: Postscript
- svg: Scalable Vector Graphics SVG
- all image formats supported by Qt, see [`QImageWriter.supportedImageFormats\(\)`](#)

Scalable vector graphic formats like PDF or SVG are superior to raster graphics formats.

Parameters

- **plot** ([`qwt.plot.QwtPlot`](#)) – Plot widget
- **dest** – [`QPaintDevice`](#), [`QPrinter`](#) or [`QSvgGenerator`](#) instance

See also:

[`render\(\)`](#), [`qwt.painter.QwtPainter.setRoundingAlignment\(\)`](#)

render(*plot*, *painter*, *plotRect*)

Paint the contents of a [`QwtPlot`](#) instance into a given rectangle.

Parameters

- **plot** ([`qwt.plot.QwtPlot`](#)) – Plot to be rendered
- **painter** ([`QPainter`](#)) – Painter

- **format** (*str*) – Format for the document
- **plotRect** (*QRectF*) – Bounding rectangle

See also:

[*renderDocument\(\)*](#), [*renderTo\(\)*](#), `qwt.painter.QwtPainter.setRoundingAlignment()`

renderTitle(*plot, painter, rect*)

Render the title into a given rectangle.

Parameters

- **plot** (`qwt.plot.QwtPlot`) – Plot widget
- **painter** (*QPainter*) – Painter
- **rect** (*QRectF*) – Bounding rectangle

renderFooter(*plot, painter, rect*)

Render the footer into a given rectangle.

Parameters

- **plot** (`qwt.plot.QwtPlot`) – Plot widget
- **painter** (*QPainter*) – Painter
- **rect** (*QRectF*) – Bounding rectangle

renderLegend(*plot, painter, rect*)

Render the legend into a given rectangle.

Parameters

- **plot** (`qwt.plot.QwtPlot`) – Plot widget
- **painter** (*QPainter*) – Painter
- **rect** (*QRectF*) – Bounding rectangle

renderScale(*plot, painter, axisId, startDist, endDist, baseDist, rect*)

Paint a scale into a given rectangle. Paint the scale into a given rectangle.

Parameters

- **plot** (`qwt.plot.QwtPlot`) – Plot widget
- **painter** (*QPainter*) – Painter
- **axisId** (*int*) – Axis
- **startDist** (*int*) – Start border distance
- **endDist** (*int*) – End border distance
- **baseDist** (*int*) – Base distance
- **rect** (*QRectF*) – Bounding rectangle

renderCanvas(*plot, painter, canvasRect, maps*)

Render the canvas into a given rectangle.

Parameters

- **plot** (`qwt.plot.QwtPlot`) – Plot widget
- **painter** (*QPainter*) – Painter

- **rect** (*QRectF*) – Bounding rectangle
- **maps** (*qwt.scale_map.QwtScaleMap*) – mapping between plot and paint device coordinates

buildCanvasMaps(*plot*, *canvasRect*)

Calculated the scale maps for rendering the canvas

Parameters

- **plot** (*qwt.plot.QwtPlot*) – Plot widget
- **canvasRect** (*QRectF*) – Target rectangle

Returns

Calculated scale maps

exportTo(*plot*, *documentname*, *sizeMM=None*, *resolution=85*)

Execute a file dialog and render the plot to the selected file

Parameters

- **plot** (*qwt.plot.QwtPlot*) – Plot widget
- **documentName** (*str*) – Default document name
- **sizeMM** (*QSizeF*) – Size for the document in millimeters
- **resolution** (*int*) – Resolution in dots per Inch (dpi)

Returns

True, when exporting was successful

See also:

[*renderDocument\(\)*](#)

5.4 Scales

5.4.1 QwtScaleMap

class *qwt.scale_map.QwtScaleMap*(*args)

A scale map

QwtScaleMap offers transformations from the coordinate system of a scale into the linear coordinate system of a paint device and vice versa.

The scale and paint device intervals are both set to [0,1].

class *QwtScaleMap*([*other=None*])

Constructor (eventually, copy constructor)

Parameters

other (*qwt.scale_map.QwtScaleMap*) – Other scale map

class *QwtScaleMap*(*p1*, *p2*, *s1*, *s2*)

Constructor (was provided by *PyQwt* but not by *Qwt*)

Parameters

- **p1** (*int*) – First border of the paint interval

- **p2** (*int*) – Second border of the paint interval
- **s1** (*float*) – First border of the scale interval
- **s2** (*float*) – Second border of the scale interval

s1()

Returns

First border of the scale interval

s2()

Returns

Second border of the scale interval

p1()

Returns

First border of the paint interval

p2()

Returns

Second border of the paint interval

pDist()

Returns

$abs(p2() - p1())$

sDist()

Returns

$abs(s2() - s1())$

transform_scalar(s)

Transform a point related to the scale interval into an point related to the interval of the paint device

Parameters

s (*float*) – Value relative to the coordinates of the scale

Returns

Transformed value

See also:

[*invTransform_scalar\(\)*](#)

invTransform_scalar(p)

Transform an paint device value into a value in the interval of the scale.

Parameters

p (*float*) – Value relative to the coordinates of the paint device

Returns

Transformed value

See also:

[*transform_scalar\(\)*](#)

isInverting()

Returns

True, when ($p1() < p2()$) \neq ($s1() < s2()$)

setTransformation(*transform*)

Initialize the map with a transformation

Parameters

transform (`qwt.transform.QwtTransform`) – Transformation

transformation()

Returns

the transformation

setScaleInterval(*s1*, *s2*)

Specify the borders of the scale interval

Parameters

- **s1** (*float*) – first border
- **s2** (*float*) – second border

Warning: Scales might be aligned to transformation depending boundaries

setPaintInterval(*p1*, *p2*)

Specify the borders of the paint device interval

Parameters

- **p1** (*float*) – first border
- **p2** (*float*) – second border

transform(args*)**

Transform a rectangle from scale to paint coordinates.

Transform a scalar:

Parameters

scalar (*float*) – Scalar

Transform a rectangle:

Parameters

- **xMap** (`qwt.scale_map.QwtScaleMap`) – X map
- **yMap** (`qwt.scale_map.QwtScaleMap`) – Y map
- **rect** (*QRectF*) – Rectangle in paint coordinates

Transform a point:

Parameters

- **xMap** (`qwt.scale_map.QwtScaleMap`) – X map
- **yMap** (`qwt.scale_map.QwtScaleMap`) – Y map
- **pos** (*QPointF*) – Position in scale coordinates

See also:

[*invTransform\(\)*](#)

invTransform(*args)

Transform from paint to scale coordinates

Scalar: `scalemap.invTransform(scalar)` Point (QPointF): `scalemap.invTransform(xMap, yMap, pos)` Rect-angle (QRectF): `scalemap.invTransform(xMap, yMap, rect)`

5.4.2 QwtScaleWidget

class `qwt.scale_widget.QwtScaleWidget(*args)`

A Widget which contains a scale

This Widget can be used to decorate composite widgets with a scale.

Layout flags:

- *QwtScaleWidget.TitleInverted*: The title of vertical scales is painted from top to bottom. Otherwise it is painted from bottom to top.

class `QwtScaleWidget([parent=None])`

Alignment default is *QwtScaleDraw.LeftScale*.

Parameters

parent (*QWidget* or *None*) – Parent widget

class `QwtScaleWidget(align, parent)`

Parameters

- **align** (*int*) – Alignment
- **parent** (*QWidget*) – Parent widget

initScale(align)

Initialize the scale

Parameters

align (*int*) – Alignment

setLayoutFlag(flag, on=True)

Toggle an layout flag

Parameters

- **flag** (*int*) – Layout flag
- **on** (*bool*) – True/False

See also:

[*testLayoutFlag\(\)*](#)

testLayoutFlag(flag)

Test a layout flag

Parameters

flag (*int*) – Layout flag

Returns

True/False

See also:

[`setLayoutFlag\(\)`](#)

setTitle(*title*)

Give title new text contents

Parameters

title (`qwt.text.QwtText` or `str`) – New title

See also:

[`title\(\)`](#)

setAlignment(*alignment*)

Change the alignment

Parameters

alignment (`int`) – New alignment

Valid alignment values: see [`qwt.scale_draw.QwtScaleDraw`](#)

See also:

[`alignment\(\)`](#)

alignment()

Returns

position

See also:

[`setAlignment\(\)`](#)

setBorderDist(*dist1*, *dist2*)

Specify distances of the scale's endpoints from the widget's borders. The actual borders will never be less than minimum border distance.

Parameters

- **dist1** (`int`) – Left or top Distance
- **dist2** (`int`) – Right or bottom distance

See also:

[`borderDist\(\)`](#)

setMargin(*margin*)

Specify the margin to the colorBar/base line.

Parameters

margin (`int`) – Margin

See also:

[`margin\(\)`](#)

setSpacing(*spacing*)

Specify the distance between color bar, scale and title

Parameters

spacing (`int`) – Spacing

See also:

[*spacing\(\)*](#)

setLabelAlignment(*alignment*)

Change the alignment for the labels.

Parameters

spacing (*int*) – Spacing

See also:

[*qwt.scale_draw.QwtScaleDraw.setLabelAlignment\(\)*](#), [*setLabelRotation\(\)*](#)

setLabelRotation(*rotation*)

Change the rotation for the labels.

Parameters

rotation (*float*) – Rotation

See also:

[*qwt.scale_draw.QwtScaleDraw.setLabelRotation\(\)*](#), [*setLabelFlags\(\)*](#)

setLabelAutoSize(*state*)

Set the automatic size option for labels (default: on).

Parameters

state (*bool*) – On/off

See also:

[*qwt.scale_draw.QwtScaleDraw.setLabelAutoSize\(\)*](#)

setScaleDraw(*scaleDraw*)

Set a scale draw

scaleDraw has to be created with new and will be deleted in class destructor or the next call of *setScaleDraw()*. scaleDraw will be initialized with the attributes of the previous scaleDraw object.

Parameters

scaleDraw ([*qwt.scale_draw.QwtScaleDraw*](#)) – ScaleDraw object

See also:

[*scaleDraw\(\)*](#)

scaleDraw()

Returns

scaleDraw of this scale

See also:

[*qwt.scale_draw.QwtScaleDraw.setScaleDraw\(\)*](#)

title()

Returns

title

See also:

[*setTitle\(\)*](#)

startBorderDist()

Returns

start border distance

See also:

[*setBorderDist\(\)*](#)

endBorderDist()

Returns

end border distance

See also:

[*setBorderDist\(\)*](#)

margin()

Returns

margin

See also:

[*setMargin\(\)*](#)

spacing()

Returns

distance between scale and title

See also:

[*setSpacing\(\)*](#)

paintEvent(*self*, *a0*: *QPaintEvent* | *None*)

draw(*painter*)

Draw the scale

Parameters

painter (*QPainter*) – Painter

colorBarRect(*rect*)

Calculate the the rectangle for the color bar

Parameters

rect (*QRectF*) – Bounding rectangle for all components of the scale

Returns

Rectangle for the color bar

resizeEvent(*self*, *a0*: *QResizeEvent* | *None*)

layoutScale(*update_geometry=True*)

Recalculate the scale's geometry and layout based on the current geometry and fonts.

Parameters

update_geometry (*bool*) – Notify the layout system and call update to redraw the scale

drawColorBar(*painter*, *rect*)

Draw the color bar of the scale widget

Parameters

- **painter** (*QPainter*) – Painter
- **rect** (*QRectF*) – Bounding rectangle for the color bar

See also:

[*setColorBarEnabled\(\)*](#)

drawTitle(*painter*, *align*, *rect*)

Rotate and paint a title according to its position into a given rectangle.

Parameters

- **painter** (*QPainter*) – Painter
- **align** (*int*) – Alignment
- **rect** (*QRectF*) – Bounding rectangle

scaleChange()

Notify a change of the scale

This method can be overloaded by derived classes. The default implementation updates the geometry and repaints the widget.

sizeHint(*self*) → *QSize***minimumSizeHint**(*self*) → *QSize***titleHeightForWidth**(*width*)

Find the height of the title for a given width.

Parameters

- **width** (*int*) – Width

Returns

Height

dimForLength(*length*, *scaleFont*)

Find the minimum dimension for a given length. *dim* is the height, *length* the width seen in direction of the title.

Parameters

- **length** (*int*) – width for horizontal, height for vertical scales
- **scaleFont** (*QFont*) – Font of the scale

Returns

height for horizontal, width for vertical scales

getBorderDistHint()

Calculate a hint for the border distances.

This member function calculates the distance of the scale's endpoints from the widget borders which is required for the mark labels to fit into the widget. The maximum of this distance and the minimum border distance is returned.

Parameters

- **start** (*int*) – Return parameter for the border width at the beginning of the scale
- **end** (*int*) – Return parameter for the border width at the end of the scale

Warning: The minimum border distance depends on the font.

See also:

[`setMinBorderDist\(\)`](#), [`getMinBorderDist\(\)`](#), [`setBorderDist\(\)`](#)

setMinBorderDist(*start*, *end*)

Set a minimum value for the distances of the scale’s endpoints from the widget borders. This is useful to avoid that the scales are “jumping”, when the tick labels or their positions change often.

Parameters

- **start** (*int*) – Minimum for the start border
- **end** (*int*) – Minimum for the end border

See also:

[`getMinBorderDist\(\)`](#), [`getBorderDistHint\(\)`](#)

getMinBorderDist()

Get the minimum value for the distances of the scale’s endpoints from the widget borders.

Parameters

- **start** (*int*) – Return parameter for the border width at the beginning of the scale
- **end** (*int*) – Return parameter for the border width at the end of the scale

See also:

[`setMinBorderDist\(\)`](#), [`getBorderDistHint\(\)`](#)

setScaleDiv(*scaleDiv*)

Assign a scale division

The scale division determines where to set the tick marks.

Parameters

scaleDiv ([`qwt.scale_div.QwtScaleDiv`](#)) – Scale Division

See also:

For more information about scale divisions, see [`qwt.scale_div.QwtScaleDiv`](#).

setTransformation(*transformation*)

Set the transformation

Parameters

transformation ([`qwt.transform.Transform`](#)) – Transformation

See also:

[`qwt.scale_draw.QwtAbstractScaleDraw.scaleDraw\(\)`](#), [`qwt.scale_map.QwtScaleMap`](#)

setColorBarEnabled(*on*)

En/disable a color bar associated to the scale

Parameters

on (*bool*) – On/Off

See also:

isColorBarEnabled(), *setColorBarWidth()*

isColorBarEnabled()

Returns

True, when the color bar is enabled

See also:

setColorBarEnabled(), *setColorBarWidth()*

setColorBarWidth(*width*)

Set the width of the color bar

Parameters

width (*int*) – Width

See also:

colorBarWidth(), *setColorBarEnabled()*

colorBarWidth()

Returns

Width of the color bar

See also:

setColorBarWidth(), *setColorBarEnabled()*

colorBarInterval()

Returns

Value interval for the color bar

See also:

setColorMap(), *colorMap()*

setColorMap(*interval*, *colorMap*)

Set the color map and value interval, that are used for displaying the color bar.

Parameters

- **interval** (*qwt.interval.QwtInterval*) – Value interval
- **colorMap** (*qwt.color_map.QwtColorMap*) – Color map

See also:

colorMap(), *colorBarInterval()*

colorMap()

Returns

Color map

See also:

setColorMap(), *colorBarInterval()*

5.4.3 QwtScaleDiv

class `qwt.scale_div.QwtScaleDiv(*args)`

A class representing a scale division

A Qwt scale is defined by its boundaries and 3 list for the positions of the major, medium and minor ticks.

The *upperLimit()* might be smaller than the *lowerLimit()* to indicate inverted scales.

Scale divisions can be calculated from a *QwtScaleEngine*.

See also:

`qwt.scale_engine.QwtScaleEngine.divideScale()`, `qwt.plot.QwtPlot.setAxisScaleDiv()`

Scale tick types:

- *QwtScaleDiv.NoTick*: No ticks
- *QwtScaleDiv.MinorTick*: Minor ticks
- *QwtScaleDiv.MediumTick*: Medium ticks
- *QwtScaleDiv.MajorTick*: Major ticks
- *QwtScaleDiv.NTickTypes*: Number of valid tick types

class `QwtScaleDiv`

Basic constructor. Lower bound = Upper bound = 0.

class `QwtScaleDiv(interval, ticks)`

Parameters

- **interval** (`qwt.interval.QwtInterval`) – Interval
- **ticks** (*list*) – list of major, medium and minor ticks

class `QwtScaleDiv(lowerBound, upperBound)`

Parameters

- **lowerBound** (*float*) – First boundary
- **upperBound** (*float*) – Second boundary

class `QwtScaleDiv(lowerBound, upperBound, ticks)`

Parameters

- **lowerBound** (*float*) – First boundary
- **upperBound** (*float*) – Second boundary
- **ticks** (*list*) – list of major, medium and minor ticks

class `QwtScaleDiv(lowerBound, upperBound, minorTicks, mediumTicks, majorTicks)`

Parameters

- **lowerBound** (*float*) – First boundary
- **upperBound** (*float*) – Second boundary
- **minorTicks** (*list*) – list of minor ticks
- **mediumTicks** (*list*) – list of medium ticks

- **majorTicks** (*list*) – list of major ticks

Note: lowerBound might be greater than upperBound for inverted scales

setInterval(*args)

Change the interval

setInterval(*lowerBound*, *upperBound*)

Parameters

- **lowerBound** (*float*) – First boundary
- **upperBound** (*float*) – Second boundary

setInterval(*interval*)

Parameters

interval (`qwt.interval.QwtInterval`) – Interval

Note: lowerBound might be greater than upperBound for inverted scales

interval()

Returns

Interval

setLowerBound(*lowerBound*)

Set the first boundary

Parameters

lowerBound (*float*) – First boundary

See also:

[*lowerBound\(\)*](#), [*setUpperBound\(\)*](#)

lowerBound()

Returns

the first boundary

See also:

[*upperBound\(\)*](#)

setUpperBound(*upperBound*)

Set the second boundary

Parameters

lowerBound (*float*) – Second boundary

See also:

[*upperBound\(\)*](#), [*setLowerBound\(\)*](#)

upperBound()

Returns

the second boundary

See also:

[*lowerBound\(\)*](#)

range()

Returns

upperBound() - lowerBound()

isEmpty()

Check if the scale division is empty(lowerBound() == upperBound())

isIncreasing()

Check if the scale division is increasing(lowerBound() <= upperBound())

contains(value)

Return if a value is between lowerBound() and upperBound()

Parameters

value (*float*) – Value

Returns

True/False

invert()

Invert the scale division

See also:

[*inverted\(\)*](#)

inverted()

Returns

A scale division with inverted boundaries and ticks

See also:

[*invert\(\)*](#)

bounded(lowerBound, upperBound)

Return a scale division with an interval [lowerBound, upperBound] where all ticks outside this interval are removed

Parameters

- **lowerBound** (*float*) – First boundary
- **upperBound** – Second boundary

Returns

Scale division with all ticks inside of the given interval

Note: lowerBound might be greater than upperBound for inverted scales

setTicks(tickType, ticks)

Assign ticks

Parameters

- **type** (*int*) – MinorTick, MediumTick or MajorTick
- **ticks** (*list*) – Values of the tick positions

ticks(*tickType*)

Return a list of ticks

Parameters

type (*int*) – MinorTick, MediumTick or MajorTick

Returns

Tick list

5.4.4 QwtScaleEngine

class `qwt.scale_engine.QwtScaleEngine`(*base=10*)

Base class for scale engines.

A scale engine tries to find “reasonable” ranges and step sizes for scales.

The layout of the scale can be varied with *setAttribute()*.

PythonQwt offers implementations for logarithmic and linear scales.

Layout attributes:

- *QwtScaleEngine.NoAttribute*: No attributes
- *QwtScaleEngine.IncludeReference*: Build a scale which includes the *reference()* value
- *QwtScaleEngine.Symmetric*: Build a scale which is symmetric to the *reference()* value
- *QwtScaleEngine.Floating*: The endpoints of the scale are supposed to be equal the outmost included values plus the specified margins (see *setMargins()*). If this attribute is *not* set, the endpoints of the scale will be integer multiples of the step size.
- *QwtScaleEngine.Inverted*: Turn the scale upside down

autoScale(*maxNumSteps*, *x1*, *x2*, *stepSize*)

Align and divide an interval

Parameters

- **maxNumSteps** (*int*) – Max. number of steps
- **x1** (*float*) – First limit of the interval (In/Out)
- **x2** (*float*) – Second limit of the interval (In/Out)
- **stepSize** (*float*) – Step size

Returns

tuple (*x1*, *x2*, *stepSize*)

divideScale(*x1*, *x2*, *maxMajorSteps*, *maxMinorSteps*, *stepSize=0.0*)

Calculate a scale division

Parameters

- **x1** (*float*) – First interval limit
- **x2** (*float*) – Second interval limit
- **maxMajorSteps** (*int*) – Maximum for the number of major steps
- **maxMinorSteps** (*int*) – Maximum number of minor steps
- **stepSize** (*float*) – Step size. If *stepSize* == 0.0, the *scaleEngine* calculates one

Returns

Calculated scale division

setTransformation(*transform*)

Assign a transformation

Parameters

transform (`qwt.transform.QwtTransform`) – Transformation

The transformation object is used as factory for clones that are returned by *transformation()*

The scale engine takes ownership of the transformation.

See also:

`QwtTransform.copy()`, *transformation()*

transformation()

Create and return a clone of the transformation of the engine. When the engine has no special transformation `None` is returned, indicating no transformation.

Returns

A clone of the transformation

See also:

setTransformation()

lowerMargin()**Returns**

the margin at the lower end of the scale

The default margin is 0.

See also:

setMargins()

upperMargin()**Returns**

the margin at the upper end of the scale

The default margin is 0.

See also:

setMargins()

setMargins(*lower*, *upper*)

Specify margins at the scale's endpoints

Parameters

- **lower** (*float*) – minimum distance between the scale's lower boundary and the smallest enclosed value
- **upper** (*float*) – minimum distance between the scale's upper boundary and the greatest enclosed value

Returns

A clone of the transformation

Margins can be used to leave a minimum amount of space between the enclosed intervals and the boundaries of the scale.

Warning: *QwtLogScaleEngine* measures the margins in decades.

See also:

upperMargin(), *lowerMargin()*

divideInterval(*intervalSize*, *numSteps*)

Calculate a step size for a given interval

Parameters

- **intervalSize** (*float*) – Interval size
- **numSteps** (*float*) – Number of steps

Returns

Step size

contains(*interval*, *value*)

Check if an interval “contains” a value

Parameters

- **intervalSize** (*float*) – Interval size
- **value** (*float*) – Value

Returns

True, when the value is inside the interval

strip(*ticks*, *interval*)

Remove ticks from a list, that are not inside an interval

Parameters

- **ticks** (*list*) – Tick list
- **interval** (*qwt.interval.QwtInterval*) – Interval

Returns

Stripped tick list

buildInterval(*value*)

Build an interval around a value

In case of $v == 0.0$ the interval is $[-0.5, 0.5]$, otherwise it is $[0.5 * v, 1.5 * v]$

Parameters

value (*float*) – Initial value

Returns

Calculated interval

setAttribute(*attribute*, *on=True*)

Change a scale attribute

Parameters

- **attribute** (*int*) – Attribute to change

- **on** (*bool*) – On/Off

Returns

Calculated interval

See also:

[`testAttribute\(\)`](#)

testAttribute(*attribute*)

Parameters

attribute (*int*) – Attribute to be tested

Returns

True, if attribute is enabled

See also:

[`setAttribute\(\)`](#)

setAttributes(*attributes*)

Change the scale attribute

Parameters

attributes – Set scale attributes

See also:

[`attributes\(\)`](#)

attributes()

Returns

Scale attributes

See also:

[`setAttributes\(\)`](#), [`testAttribute\(\)`](#)

setReference(*r*)

Specify a reference point

Parameters

r (*float*) – new reference value

The reference point is needed if options *IncludeReference* or *Symmetric* are active. Its default value is 0.0.

reference()

Returns

the reference value

See also:

[`setReference\(\)`](#), [`setAttribute\(\)`](#)

setBase(*base*)

Set the base of the scale engine

While a base of 10 is what 99.9% of all applications need certain scales might need a different base: f.e 2

The default setting is 10

Parameters

base (*int*) – Base of the engine

See also:

[`base\(\)`](#)

base()

Returns

Base of the scale engine

See also:

[`setBase\(\)`](#)

5.4.5 QwtLinearScaleEngine

class `qwt.scale_engine.QwtLinearScaleEngine`(*base=10*)

A scale engine for linear scales

The step size will fit into the pattern $f\$left\{ 1,2,5right\} \cdot 10^{\{n\}}f\$, where n is an integer.$

autoScale(*maxNumSteps*, *x1*, *x2*, *stepSize*)

Align and divide an interval

Parameters

- **maxNumSteps** (*int*) – Max. number of steps
- **x1** (*float*) – First limit of the interval (In/Out)
- **x2** (*float*) – Second limit of the interval (In/Out)
- **stepSize** (*float*) – Step size

Returns

tuple (x1, x2, stepSize)

See also:

[`setAttribute\(\)`](#)

divideScale(*x1*, *x2*, *maxMajorSteps*, *maxMinorSteps*, *stepSize=0.0*)

Calculate a scale division for an interval

Parameters

- **x1** (*float*) – First interval limit
- **x2** (*float*) – Second interval limit
- **maxMajorSteps** (*int*) – Maximum for the number of major steps
- **maxMinorSteps** (*int*) – Maximum number of minor steps
- **stepSize** (*float*) – Step size. If `stepSize == 0.0`, the scaleEngine calculates one

Returns

Calculated scale division

buildTicks(*interval*, *stepSize*, *maxMinorSteps*)

Calculate ticks for an interval

Parameters

- **interval** (`qwt.interval.QwtInterval`) – Interval

- **stepSize** (*float*) – Step size
- **maxMinorSteps** (*int*) – Maximum number of minor steps

Returns

Calculated ticks

buildMajorTicks(*interval*, *stepSize*)

Calculate major ticks for an interval

Parameters

- **interval** (`qwt.interval.QwtInterval`) – Interval
- **stepSize** (*float*) – Step size

Returns

Calculated ticks

buildMinorTicks(*ticks*, *maxMinorSteps*, *stepSize*)

Calculate minor ticks for an interval

Parameters

- **ticks** (*list*) – Major ticks (returned)
- **maxMinorSteps** (*int*) – Maximum number of minor steps
- **stepSize** (*float*) – Step size

align(*interval*, *stepSize*)

Align an interval to a step size

The limits of an interval are aligned that both are integer multiples of the step size.

Parameters

- **interval** (`qwt.interval.QwtInterval`) – Interval
- **stepSize** (*float*) – Step size

Returns

Aligned interval

5.4.6 QwtLogScaleEngine

class `qwt.scale_engine.QwtLogScaleEngine`(*base=10*)

A scale engine for logarithmic scales

The step size is measured in *decades* and the major step size will be adjusted to fit the pattern $\{1,2,3,5\} \cdot 10^{*n}$, where *n* is a natural number including zero.

Warning: The step size as well as the margins are measured in *decades*.

autoScale(*maxNumSteps*, *x1*, *x2*, *stepSize*)

Align and divide an interval

Parameters

- **maxNumSteps** (*int*) – Max. number of steps

- **x1** (*float*) – First limit of the interval (In/Out)
- **x2** (*float*) – Second limit of the interval (In/Out)
- **stepSize** (*float*) – Step size

Returns

tuple (x1, x2, stepSize)

See also:

setAttribute()

divideScale(*x1*, *x2*, *maxMajorSteps*, *maxMinorSteps*, *stepSize=0.0*)

Calculate a scale division for an interval

Parameters

- **x1** (*float*) – First interval limit
- **x2** (*float*) – Second interval limit
- **maxMajorSteps** (*int*) – Maximum for the number of major steps
- **maxMinorSteps** (*int*) – Maximum number of minor steps
- **stepSize** (*float*) – Step size. If stepSize == 0.0, the scaleEngine calculates one

Returns

Calculated scale division

buildTicks(*interval*, *stepSize*, *maxMinorSteps*)

Calculate ticks for an interval

Parameters

- **interval** (`qwt.interval.QwtInterval`) – Interval
- **stepSize** (*float*) – Step size
- **maxMinorSteps** (*int*) – Maximum number of minor steps

Returns

Calculated ticks

buildMajorTicks(*interval*, *stepSize*)

Calculate major ticks for an interval

Parameters

- **interval** (`qwt.interval.QwtInterval`) – Interval
- **stepSize** (*float*) – Step size

Returns

Calculated ticks

buildMinorTicks(*ticks*, *maxMinorSteps*, *stepSize*)

Calculate minor ticks for an interval

Parameters

- **ticks** (*list*) – Major ticks (returned)
- **maxMinorSteps** (*int*) – Maximum number of minor steps
- **stepSize** (*float*) – Step size

align(*interval*, *stepSize*)

Align an interval to a step size

The limits of an interval are aligned that both are integer multiples of the step size.

Parameters

- **interval** (`qwt.interval.QwtInterval`) – Interval
- **stepSize** (`float`) – Step size

Returns

Aligned interval

5.4.7 QwtAbstractScaleDraw

class `qwt.scale_draw.QwtAbstractScaleDraw`

A abstract base class for drawing scales

QwtAbstractScaleDraw can be used to draw linear or logarithmic scales.

After a scale division has been specified as a *QwtScaleDiv* object using *setScaleDiv()*, the scale can be drawn with the *draw()* member.

Scale components:

- *QwtAbstractScaleDraw.Backbone*: Backbone = the line where the ticks are located
- *QwtAbstractScaleDraw.Ticks*: Ticks
- *QwtAbstractScaleDraw.Labels*: Labels

class `QwtAbstractScaleDraw`

The range of the scale is initialized to [0, 100], The spacing (distance between ticks and labels) is set to 4, the tick lengths are set to 4,6 and 8 pixels

extent(*font*)

Calculate the extent

The extent is the distance from the baseline to the outermost pixel of the scale draw in opposite to its orientation. It is at least *minimumExtent()* pixels.

Parameters

font (`QFont`) – Font used for drawing the tick labels

Returns

Number of pixels

See also:

`setMinimumExtent()`, `minimumExtent()`

drawTick(*painter*, *value*, *len_*)

Draw a tick

Parameters

- **painter** (`QPainter`) – Painter
- **value** (`float`) – Value of the tick
- **len** (`float`) – Length of the tick

See also:

[`drawBackbone\(\)`](#), [`drawLabel\(\)`](#)

drawBackbone(*painter*)

Draws the baseline of the scale

Parameters

painter (*QPainter*) – Painter

See also:

[`drawTick\(\)`](#), [`drawLabel\(\)`](#)

drawLabel(*painter*, *value*)

Draws the label for a major scale tick

Parameters

- **painter** (*QPainter*) – Painter
- **value** (*float*) – Value

See also:

[`drawTick\(\)`](#), [`drawBackbone\(\)`](#)

enableComponent(*component*, *enable*)

En/Disable a component of the scale

Parameters

- **component** (*int*) – Scale component
- **enable** (*bool*) – On/Off

See also:

[`hasComponent\(\)`](#)

hasComponent(*component*)

Check if a component is enabled

Parameters

component (*int*) – Component type

Returns

True, when component is enabled

See also:

[`enableComponent\(\)`](#)

setScaleDiv(*scaleDiv*)

Change the scale division

Parameters

scaleDiv (*qwt.scale_div.QwtScaleDiv*) – New scale division

setTransformation(*transformation*)

Change the transformation of the scale

Parameters

transformation (*qwt.transform.QwtTransform*) – New scale transformation

scaleMap()

Returns

Map how to translate between scale and pixel values

scaleDiv()

Returns

scale division

setPenWidth(*width*)

Specify the width of the scale pen

Parameters

width (*int*) – Pen width

See also:

[*penWidth\(\)*](#)

penWidth()

Returns

Scale pen width

See also:

[*setPenWidth\(\)*](#)

draw(*painter*, *palette*)

Draw the scale

Parameters

- **painter** (*QPainter*) – The painter
- **palette** (*QPalette*) – Palette, text color is used for the labels, foreground color for ticks and backbone

setSpacing(*spacing*)

Set the spacing between tick and labels

The spacing is the distance between ticks and labels. The default spacing is 4 pixels.

Parameters

spacing (*float*) – Spacing

See also:

[*spacing\(\)*](#)

spacing()

Get the spacing

The spacing is the distance between ticks and labels. The default spacing is 4 pixels.

Returns

Spacing

See also:

[*setSpacing\(\)*](#)

setMinimumExtent(*minExtent*)

Set a minimum for the extent

The extent is calculated from the components of the scale draw. In situations, where the labels are changing and the layout depends on the extent (f.e scrolling a scale), setting an upper limit as minimum extent will avoid jumps of the layout.

Parameters

minExtent (*float*) – Minimum extent

See also:

[`extent\(\)`](#), [`minimumExtent\(\)`](#)

minimumExtent()

Get the minimum extent

Returns

Minimum extent

See also:

[`extent\(\)`](#), [`setMinimumExtent\(\)`](#)

setTickLength(*tick_type*, *length*)

Set the length of the ticks

Parameters

- **tick_type** (*int*) – Tick type
- **length** (*float*) – New length

Warning: the length is limited to [0..1000]

tickLength(*tick_type*)**Parameters**

tick_type (*int*) – Tick type

Returns

Length of the ticks

See also:

[`setTickLength\(\)`](#), [`maxTickLength\(\)`](#)

maxTickLength()**Returns**

Length of the longest tick

Useful for layout calculations

See also:

[`tickLength\(\)`](#), [`setTickLength\(\)`](#)

setTickLighterFactor(*tick_type*, *factor*)

Set the color lighter factor of the ticks

Parameters

- **tick_type** (*int*) – Tick type
- **factor** (*int*) – New factor

tickLighterFactor(*tick_type*)

Parameters

tick_type (*int*) – Tick type

Returns

Color lighter factor of the ticks

See also:

[*setTickLighterFactor\(\)*](#)

label(*value*)

Convert a value into its representing label

The value is converted to a plain text using *QLocale().toString(value)*. This method is often overloaded by applications to have individual labels.

Parameters

value (*float*) – Value

Returns

Label string

tickLabel(*font, value*)

Convert a value into its representing label and cache it.

The conversion between value and label is called very often in the layout and painting code. Unfortunately the calculation of the label sizes might be slow (really slow for rich text in Qt4), so it's necessary to cache the labels.

Parameters

- **font** (*QFont*) – Font
- **value** (*float*) – Value

Returns

Tuple (tick label, text size)

invalidateCache()

Invalidate the cache used by *tickLabel()*

The cache is invalidated, when a new *QwtScaleDiv* is set. If the labels need to be changed. while the same *QwtScaleDiv* is set, *invalidateCache()* needs to be called manually.

5.4.8 QwtScaleDraw

class qwt.scale_draw.**QwtScaleDraw**

A class for drawing scales

QwtScaleDraw can be used to draw linear or logarithmic scales. A scale has a position, an alignment and a length, which can be specified. The labels can be rotated and aligned to the ticks using *setLabelRotation()* and *setLabelAlignment()*.

After a scale division has been specified as a *QwtScaleDiv* object using *QwtAbstractScaleDraw.setScaleDiv(scaleDiv)*, the scale can be drawn with the *QwtAbstractScaleDraw.draw()* member.

Alignment of the scale draw:

- *QwtScaleDraw.BottomScale*: The scale is below
- *QwtScaleDraw.TopScale*: The scale is above
- *QwtScaleDraw.LeftScale*: The scale is left
- *QwtScaleDraw.RightScale*: The scale is right

class QwtScaleDraw

The range of the scale is initialized to [0, 100], The position is at (0, 0) with a length of 100. The orientation is *QwtAbstractScaleDraw.Bottom*.

alignment()

Returns

Alignment of the scale

See also:

[*setAlignment\(\)*](#)

setAlignment(*align*)

Set the alignment of the scale

Parameters

align (*int*) – Alignment of the scale

Alignment of the scale draw:

- *QwtScaleDraw.BottomScale*: The scale is below
- *QwtScaleDraw.TopScale*: The scale is above
- *QwtScaleDraw.LeftScale*: The scale is left
- *QwtScaleDraw.RightScale*: The scale is right

The default alignment is *QwtScaleDraw.BottomScale*

See also:

[*alignment\(\)*](#)

orientation()

Return the orientation

TopScale, BottomScale are horizontal (*Qt.Horizontal*) scales, LeftScale, RightScale are vertical (*Qt.Vertical*) scales.

Returns

Orientation of the scale

See also:

[*alignment\(\)*](#)

getBorderDistHint(*font*)

Determine the minimum border distance

This member function returns the minimum space needed to draw the mark labels at the scale's endpoints.

Parameters

font (*QFont*) – Font

Returns

tuple (*start*, *end*)

Returned tuple:

- start: Start border distance
- end: End border distance

minLabelDist(*font*)

Determine the minimum distance between two labels, that is necessary that the texts don't overlap.

Parameters

font (*QFont*) – Font

Returns

The maximum width of a label

See also:

[*getBorderDistHint\(\)*](#)

extent(*font*)

Calculate the width/height that is needed for a vertical/horizontal scale.

The extent is calculated from the pen width of the backbone, the major tick length, the spacing and the maximum width/height of the labels.

Parameters

font (*QFont*) – Font used for painting the labels

Returns

Extent

See also:

[*minLength\(\)*](#)

minLength(*font*)

Calculate the minimum length that is needed to draw the scale

Parameters

font (*QFont*) – Font used for painting the labels

Returns

Minimum length that is needed to draw the scale

See also:

[*extent\(\)*](#)

labelPosition(*value*)

Find the position, where to paint a label

The position has a distance that depends on the length of the ticks in direction of the *alignment()*.

Parameters

value (*float*) – Value

Returns

Position, where to paint a label

drawTick(*painter*, *value*, *len_*)

Draw a tick

Parameters

- **painter** (*QPainter*) – Painter

- **value** (*float*) – Value of the tick
- **len** (*float*) – Length of the tick

See also:

[*drawBackbone\(\)*](#), [*drawLabel\(\)*](#)

drawBackbone(*painter*)

Draws the baseline of the scale

Parameters

painter (*QPainter*) – Painter

See also:

[*drawTick\(\)*](#), [*drawLabel\(\)*](#)

move(**args*)

Move the position of the scale

The meaning of the parameter pos depends on the alignment:

- *QwtScaleDraw.LeftScale*:

The origin is the topmost point of the backbone. The backbone is a vertical line. Scale marks and labels are drawn at the left of the backbone.

- *QwtScaleDraw.RightScale*:

The origin is the topmost point of the backbone. The backbone is a vertical line. Scale marks and labels are drawn at the right of the backbone.

- *QwtScaleDraw.TopScale*:

The origin is the leftmost point of the backbone. The backbone is a horizontal line. Scale marks and labels are drawn above the backbone.

- *QwtScaleDraw.BottomScale*:

The origin is the leftmost point of the backbone. The backbone is a horizontal line. Scale marks and labels are drawn below the backbone.

move(*x*, *y*)

Parameters

- **x** (*float*) – X coordinate
- **y** (*float*) – Y coordinate

move(*pos*)

Parameters

pos (*QPointF*) – position

See also:

[*pos\(\)*](#), [*setLength\(\)*](#)

pos()

Returns

Origin of the scale

See also:

[*pos\(\)*](#), [*setLength\(\)*](#)

setLength(*length*)

Set the length of the backbone.

The length doesn't include the space needed for overlapping labels.

Parameters

length (*float*) – Length of the backbone

See also:

[*move\(\)*](#), [*minLabelDist\(\)*](#)

length()**Returns**

the length of the backbone

See also:

[*setLength\(\)*](#), [*pos\(\)*](#)

drawLabel(*painter*, *value*)

Draws the label for a major scale tick

Parameters

- **painter** (*QPainter*) – Painter
- **value** (*float*) – Value

See also:

[*drawTick\(\)*](#), [*drawBackbone\(\)*](#), [*boundingLabelRect\(\)*](#)

boundingLabelRect(*font*, *value*)

Find the bounding rectangle for the label.

The coordinates of the rectangle are absolute (calculated from *pos()*) in direction of the tick.

Parameters

- **font** (*QFont*) – Font used for painting
- **value** (*float*) – Value

Returns

Bounding rectangle

See also:

[*labelRect\(\)*](#)

labelTransformation(*pos*, *size*)

Calculate the transformation that is needed to paint a label depending on its alignment and rotation.

Parameters

- **pos** (*QPointF*) – Position where to paint the label
- **size** (*QSizeF*) – Size of the label

Returns

Transformation matrix

See also:

[*setLabelAlignment\(\)*](#), [*setLabelRotation\(\)*](#)

labelRect(*font, value*)

Find the bounding rectangle for the label. The coordinates of the rectangle are relative to spacing + tick length from the backbone in direction of the tick.

Parameters

- **font** (*QFont*) – Font used for painting
- **value** (*float*) – Value

Returns

Bounding rectangle that is needed to draw a label

labelSize(*font, value*)

Calculate the size that is needed to draw a label

Parameters

- **font** (*QFont*) – Label font
- **value** (*float*) – Value

Returns

Size that is needed to draw a label

setLabelRotation(*rotation*)

Rotate all labels.

When changing the rotation, it might be necessary to adjust the label flags too. Finding a useful combination is often the result of try and error.

Parameters

rotation (*float*) – Angle in degrees. When changing the label rotation, the label flags often needs to be adjusted too.

See also:

[*setLabelAlignment\(\)*](#), [*labelRotation\(\)*](#), [*labelAlignment\(\)*](#)

labelRotation()**Returns**

the label rotation

See also:

[*setLabelRotation\(\)*](#), [*labelAlignment\(\)*](#)

setLabelAlignment(*alignment*)

Change the label flags

Labels are aligned to the point tick length + spacing away from the backbone.

The alignment is relative to the orientation of the label text. In case of an flags of 0 the label will be aligned depending on the orientation of the scale:

- *QwtScaleDraw.TopScale*: *Qt.AlignHCenter* | *Qt.AlignTop*
- *QwtScaleDraw.BottomScale*: *Qt.AlignHCenter* | *Qt.AlignBottom*
- *QwtScaleDraw.LeftScale*: *Qt.AlignLeft* | *Qt.AlignVCenter*
- *QwtScaleDraw.RightScale*: *Qt.AlignRight* | *Qt.AlignVCenter*

Changing the alignment is often necessary for rotated labels.

:param Qt.Alignment alignment Or'd *Qt.AlignmentFlags*

See also:

setLabelRotation(), *labelRotation()*, *labelAlignment()*

Warning: The various alignments might be confusing. The alignment of the label is not the alignment of the scale and is not the alignment of the flags (*QwtText.flags()*) returned from *QwtAbstractScale-Draw.label()*.

labelAlignment()

Returns

the label flags

See also:

setLabelAlignment(), *labelRotation()*

setLabelAutoSize(*state*)

Set label automatic size option state

When drawing text labels, if automatic size mode is enabled (default behavior), the axes are drawn in order to optimize layout space and depends on text label individual sizes. Otherwise, width and height won't change when axis range is changing.

This option is not implemented in Qwt C++ library: this may be used either as an optimization (updating plot layout is faster when this option is enabled) or as an appearance preference (with Qwt default behavior, the size of axes may change when zooming and/or panning plot canvas which in some cases may not be desired).

Parameters

state (*bool*) – On/off

See also:

labelAutoSize()

labelAutoSize()

Returns

True if automatic size option is enabled for labels

See also:

setLabelAutoSize()

maxLabelWidth(*font*)

Parameters

font (*QFont*) – Font

Returns

the maximum width of a label

maxLabelHeight(*font*)

Parameters

font (*QFont*) – Font

Returns

the maximum height of a label

5.5 QwtSymbol

class qwt.symbol.QwtSymbol(*args)

A class for drawing symbols

Symbol styles:

- *QwtSymbol.NoSymbol*: No Style. The symbol cannot be drawn.
- *QwtSymbol.Ellipse*: Ellipse or circle
- *QwtSymbol.Rect*: Rectangle
- *QwtSymbol.Diamond*: Diamond
- *QwtSymbol.Triangle*: Triangle pointing upwards
- *QwtSymbol.DTriangle*: Triangle pointing downwards
- *QwtSymbol.UTriangle*: Triangle pointing upwards
- *QwtSymbol.LTriangle*: Triangle pointing left
- *QwtSymbol.RTriangle*: Triangle pointing right
- *QwtSymbol.Cross*: Cross (+)
- *QwtSymbol.XCross*: Diagonal cross (X)
- *QwtSymbol.HLine*: Horizontal line
- *QwtSymbol.VLine*: Vertical line
- *QwtSymbol.Star1*: X combined with +
- *QwtSymbol.Star2*: Six-pointed star
- *QwtSymbol.Hexagon*: Hexagon
- *QwtSymbol.Path*: The symbol is represented by a painter path, where the origin (0, 0) of the path coordinate system is mapped to the position of the symbol

..seealso:

```
:py:meth:`setPath()`, :py:meth:`path()`
```

- *QwtSymbol.Pixmap*: The symbol is represented by a pixmap. The pixmap is centered or aligned to its pin point.

..seealso:

```
:py:meth:`setPinPoint()`
```

- *QwtSymbol.Graphic*: The symbol is represented by a graphic. The graphic is centered or aligned to its pin point.

..seealso:

```
:py:meth:`setPinPoint()`
```

- *QwtSymbol.SvgDocument*: The symbol is represented by a SVG graphic. The graphic is centered or aligned to its pin point.

..seealso:

`:py:meth: `setPinPoint()``

- *QwtSymbol.UserStyle*: Styles \geq *QwtSymbol.UserStyle* are reserved for derived classes of *QwtSymbol* that overload *drawSymbols()* with additional application specific symbol types.

Cache policies:

Depending on the render engine and the complexity of the symbol shape it might be faster to render the symbol to a pixmap and to paint this pixmap.

F.e. the raster paint engine is a pure software renderer where in cache mode a draw operation usually ends in raster operation with the the backing store, that are usually faster, than the algorithms for rendering polygons. But the opposite can be expected for graphic pipelines that can make use of hardware acceleration.

The default setting is AutoCache

..seealso:

`:py:meth: `setCachePolicy()`, :py:meth: `cachePolicy()``

Note: The policy has no effect, when the symbol is painted to a vector graphics format (PDF, SVG).

Warning: Since Qt 4.8 raster is the default backend on X11

Valid cache policies:

- *QwtSymbol.NoCache*: Don't use a pixmap cache
- *QwtSymbol.Cache*: Always use a pixmap cache
- *QwtSymbol.AutoCache*: Use a cache when the symbol is rendered with the software renderer (*QPaintEngine.Raster*)

class *QwtSymbol* ([*style=QwtSymbol.NoSymbol*])

The symbol is constructed with gray interior, black outline with zero width, no size and style 'NoSymbol'.

Parameters

style (*int*) – Symbol Style

class *QwtSymbol* (*style, brush, pen, size*)

Parameters

- **style** (*int*) – Symbol Style
- **brush** (*QBrush*) – Brush to fill the interior
- **pen** (*QPen*) – Outline pen
- **size** (*QSize*) – Size

class `QwtSymbol`(*path*, *brush*, *pen*)

Parameters

- **path** (*QPainterPath*) – Painter path
- **brush** (*QBrush*) – Brush to fill the interior
- **pen** (*QPen*) – Outline pen

See also:

[`setPath\(\)`](#), [`setBrush\(\)`](#), [`setPen\(\)`](#), [`setSize\(\)`](#)

Style

alias of `int`

classmethod `make`(*style=None*, *brush=None*, *pen=None*, *size=None*, *path=None*, *pixmap=None*, *graphic=None*, *svgdocument=None*, *pinpoint=None*)

Create and setup a new *QwtSymbol* object (convenience function).

Parameters

- **style** (*int or None*) – Symbol Style
- **brush** (*QBrush or None*) – Brush to fill the interior
- **pen** (*QPen or None*) – Outline pen
- **size** (*QSize or None*) – Size
- **path** (*QPainterPath or None*) – Painter path
- **path** – Painter path
- **pixmap** (*QPixmap or None*) – Pixmap as symbol
- **graphic** ([`qwt.graphic.QwtGraphic`](#) or *None*) – Graphic
- **svgdocument** – SVG icon as symbol

See also:

[`setPixmap\(\)`](#), [`setGraphic\(\)`](#), [`setPath\(\)`](#)

setCachePolicy(*policy*)

Change the cache policy

The default policy is `AutoCache`

Parameters

policy (*int*) – Cache policy

See also:

[`cachePolicy\(\)`](#)

cachePolicy()

Returns

Cache policy

See also:

[`setCachePolicy\(\)`](#)

setPath(path)

Set a painter path as symbol

The symbol is represented by a painter path, where the origin (0, 0) of the path coordinate system is mapped to the position of the symbol.

When the symbol has valid size the painter path gets scaled to fit into the size. Otherwise the symbol size depends on the bounding rectangle of the path.

The following code defines a symbol drawing an arrow:

```
from qtpy.QtGui import QApplication, QPen, QPainterPath, QTransform
from qtpy.QtCore import Qt, QPointF
from qwt import QwtPlot, QwtPlotCurve, QwtSymbol
import numpy as np

app = QApplication([])

# --- Construct custom symbol ---

path = QPainterPath()
path.moveTo(0, 8)
path.lineTo(0, 5)
path.lineTo(-3, 5)
path.lineTo(0, 0)
path.lineTo(3, 5)
path.lineTo(0, 5)

transform = QTransform()
transform.rotate(-30.0)
path = transform.map(path)

pen = QPen(Qt.black, 2);
pen.setJoinStyle(Qt.MiterJoin)

symbol = QwtSymbol()
symbol.setPen(pen)
symbol.setBrush(Qt.red)
symbol.setPath(path)
symbol.setPinPoint(QPointF(0., 0.))
symbol.setSize(10, 14)

# --- Test it within a simple plot ---

curve = QwtPlotCurve()
curve_pen = QPen(Qt.blue)
curve_pen.setStyle(Qt.DotLine)
curve.setPen(curve_pen)
curve.setSymbol(symbol)
x = np.linspace(0, 10, 10)
curve.setData(x, np.sin(x))

plot = QwtPlot()
curve.attach(plot)
plot.resize(600, 300)
```

(continues on next page)

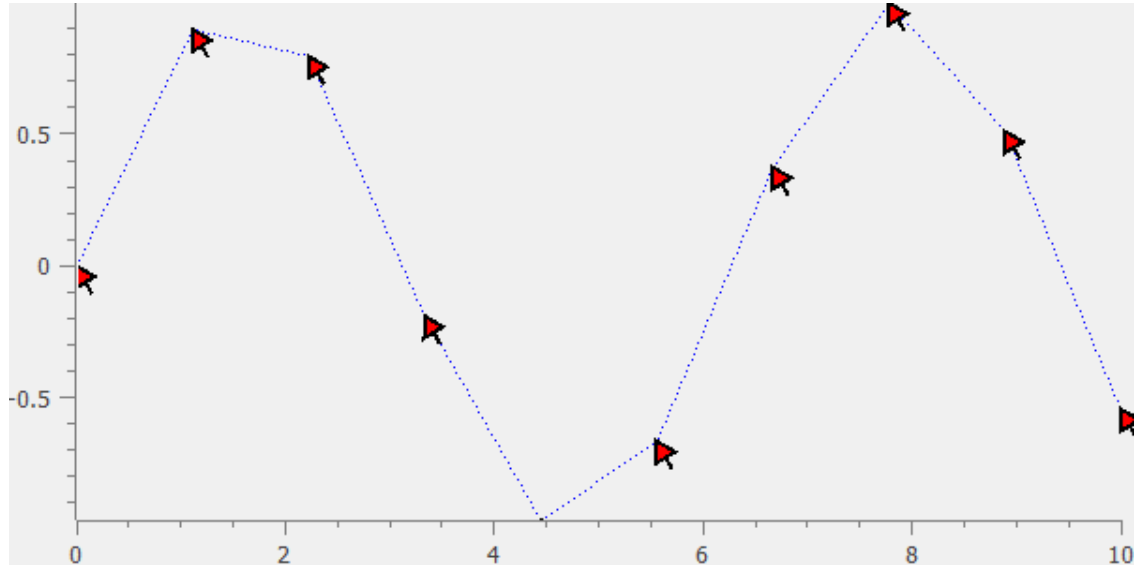
(continued from previous page)

```

plot.replot()
plot.show()

app.exec_()

```

**Parameters****path** (*QPainterPath*) – Painter path**See also:**[*path\(\)*](#), [*setSize\(\)*](#)**path()****Returns**

Painter path for displaying the symbol

See also:[*setPath\(\)*](#)**setPixmap**(*pixmap*)

Set a pixmap as symbol

Parameters**pixmap** (*QPixmap*) – Pixmap**See also:**[*pixmap\(\)*](#), [*setGraphic\(\)*](#)

Note: The *style()* is set to *QwtSymbol.Pixmap*

Note: *brush()* and *pen()* have no effect

pixmap()

Returns

Assigned pixmap

See also:

[*setPixmap\(\)*](#)

setGraphic(*graphic*)

Set a graphic as symbol

Parameters

graphic ([*qwt.graphic.QwtGraphic*](#)) – Graphic

See also:

[*graphic\(\)*](#), [*setPixmap\(\)*](#)

Note: The *style()* is set to *QwtSymbol.Graphic*

Note: *brush()* and *pen()* have no effect

graphic()

Returns

Assigned graphic

See also:

[*setGraphic\(\)*](#)

setSvgDocument(*svgDocument*)

Set a SVG icon as symbol

Parameters

svgDocument – SVG icon

See also:

[*setGraphic\(\)*](#), [*setPixmap\(\)*](#)

Note: The *style()* is set to *QwtSymbol.SvgDocument*

Note: *brush()* and *pen()* have no effect

setSize(*args)

Specify the symbol's size

setSize(*width*[, *height=-1*])

Parameters

- **width** (*int*) – Width
- **height** (*int*) – Height

setSize(*size*)

Parameters

size (*QSize*) – Size

See also:

[*size\(\)*](#)

size()

Returns

Size

See also:

[*setSize\(\)*](#)

setBrush(*brush*)

Assign a brush

The brush is used to draw the interior of the symbol.

Parameters

brush (*QBrush*) – Brush

See also:

[*brush\(\)*](#)

brush()

Returns

Brush

See also:

[*setBrush\(\)*](#)

setPen(args*)**

Build and/or assign a pen, depending on the arguments.

setPen(*color*, *width*, *style*)

Build and assign a pen

In Qt5 the default pen width is 1.0 (0.0 in Qt4) what makes it non cosmetic (see *QPen.isCosmetic()*).

This method signature has been introduced to hide this incompatibility.

Parameters

- **color** (*QColor*) – Pen color
- **width** (*float*) – Pen width
- **style** (*Qt.PenStyle*) – Pen style

setPen(*pen*)

Assign a pen

Parameters

pen (*QPen*) – New pen

See also:

[*pen\(\)*](#), [*brush\(\)*](#)

pen()

Returns

Pen

See also:

[*setPen\(\)*](#), [*brush\(\)*](#)

setColor(*color*)

Set the color of the symbol

Change the color of the brush for symbol types with a filled area. For all other symbol types the color will be assigned to the pen.

Parameters

color (*QColor*) – Color

See also:

[*setPen\(\)*](#), [*setBrush\(\)*](#), [*brush\(\)*](#), [*pen\(\)*](#)

setPinPoint(*pos*, *enable=True*)

Set and enable a pin point

The position of a complex symbol is not always aligned to its center (f.e an arrow, where the peak points to a position). The pin point defines the position inside of a QPixmap, Graphic, SvgDocument or PainterPath symbol where the represented point has to be aligned to.

Parameters

pos (*QPointF*) – Position

Enable bool enable

En/Disable the pin point alignment

See also:

[*pinPoint\(\)*](#), [*setPinPointEnabled\(\)*](#)

pinPoint()

Returns

Pin point

See also:

[*setPinPoint\(\)*](#), [*setPinPointEnabled\(\)*](#)

setPinPointEnabled(*on*)

En/Disable the pin point alignment

Parameters

on (*bool*) – Enabled, when on is true

See also:

[*setPinPoint\(\)*](#), [*isPinPointEnabled\(\)*](#)

isPinPointEnabled()

Returns

True, when the pin point translation is enabled

See also:

[*setPinPoint\(\)*](#), [*setPinPointEnabled\(\)*](#)

drawSymbols(*painter, points*)

Render an array of symbols

Painting several symbols is more effective than drawing symbols one by one, as a couple of layout calculations and setting of pen/brush can be done once for the complete array.

Parameters

- **painter** (*QPainter*) – Painter
- **points** (*QPolygonF*) – Positions of the symbols in screen coordinates

drawSymbol(*painter, point_or_rect*)

Draw the symbol into a rectangle

The symbol is painted centered and scaled into the target rectangle. It is always painted uncached and the pin point is ignored.

This method is primarily intended for drawing a symbol to the legend.

Parameters

- **painter** (*QPainter*) – Painter
- **point_or_rect** (*QPointF* or *QPoint* or *QRectF*) – Position or target rectangle of the symbol in screen coordinates

renderSymbols(*painter, points*)

Render the symbol to series of points

Parameters

- **painter** (*QPainter*) – Painter
- **point_or_rect** – Positions of the symbols

boundingRect()

Calculate the bounding rectangle for a symbol at position (0,0).

Returns

Bounding rectangle

invalidateCache()

Invalidate the cached symbol pixmap

The symbol invalidates its cache, whenever an attribute is changed that has an effect on how to display a symbol. In case of derived classes with individual styles (\geq *QwtSymbol.UserStyle*) it might be necessary to call `invalidateCache()` for attributes that are relevant for this style.

See also:

[`setCachePolicy\(\)`](#), [`drawSymbols\(\)`](#)

setStyle(*style*)

Specify the symbol style

Parameters

style (*int*) – Style

See also:

[`style\(\)`](#)

style()

Returns

Current symbol style

See also:

[*setStyle\(\)*](#)

5.6 Text widgets

5.6.1 QwtText

class `qwt.text.QwtText` (*text=None, textFormat=None, other=None*)

A class representing a text

A *QwtText* is a text including a set of attributes how to render it.

- **Format:**

A text might include control sequences (f.e tags) describing how to render it. Each format (f.e MathML, TeX, Qt Rich Text) has its own set of control sequences, that can be handles by a special *QwtTextEngine* for this format.

- **Background:**

A text might have a background, defined by a *QPen* and *QBrush* to improve its visibility. The corners of the background might be rounded.

- **Font:**

A text might have an individual font.

- **Color**

A text might have an individual color.

- **Render Flags**

Flags from *Qt.AlignmentFlag* and *Qt.TextFlag* used like in *QPainter.drawText()*.

..seealso:

```
:py:meth: `qwt.text.QwtTextEngine`,  
:py:meth: `qwt.text.QwtTextLabel`
```

Text formats:

- *QwtText.AutoText:*

The text format is determined using *QwtTextEngine.mightRender()* for all available text engines in increasing order > *PlainText*. If none of the text engines can render the text is rendered like *QwtText.PlainText*.

- *QwtText.PlainText:*

Draw the text as it is, using a *QwtPlainTextEngine*.

- *QwtText.RichText:*

Use the Scribe framework (Qt Rich Text) to render the text.

- *QwtText.OtherFormat*:

The number of text formats can be extended using *setTextEngine*. Formats \geq *QwtText.OtherFormat* are not used by Qwt.

Paint attributes:

- *QwtText.PaintUsingTextFont*: The text has an individual font.
- *QwtText.PaintUsingTextColor*: The text has an individual color.
- *QwtText.PaintBackground*: The text has an individual background.

Layout attributes:

- *QwtText.MinimumLayout*:

Layout the text without its margins. This mode is useful if a text needs to be aligned accurately, like the tick labels of a scale. If *QwtTextEngine.textMargins* is not implemented for the format of the text, *MinimumLayout* has no effect.

```
class QwtText([text=None][, textFormat=None][, other=None])
```

Parameters

- **text** (*str*) – Text content
- **textFormat** (*int*) – Text format
- **other** (*qwt.text.QwtText*) – Object to copy (text and textFormat arguments are ignored)

```
classmethod make(text=None, textformat=None, renderflags=None, font=None, family=None,
                 pointsize=None, weight=None, color=None, borderradius=None, borderpen=None,
                 brush=None)
```

Create and setup a new *QwtText* object (convenience function).

Parameters

- **text** (*str*) – Text content
- **textformat** (*int*) – Text format
- **renderflags** (*int*) – Flags from *Qt.AlignmentFlag* and *Qt.TextFlag*
- **font** (*QFont* or *None*) – Font
- **family** (*str* or *None*) – Font family (default: Helvetica)
- **pointsize** (*int* or *None*) – Font point size (default: 10)
- **weight** (*int* or *None*) – Font weight (default: *QFont.Normal*)
- **color** (*QColor* or *str* or *None*) – Pen color
- **borderradius** (*float* or *None*) – Radius for the corners of the border frame
- **borderpen** (*QPen* or *None*) – Background pen
- **brush** (*QBrush* or *None*) – Background brush

See also:

[*setText\(\)*](#)

```
isEmpty()
```

Returns

True if text is empty

setText(*text*, *textFormat=None*)

Assign a new text content

Parameters

- **text** (*str*) – Text content
- **textFormat** (*int*) – Text format

See also:

[*text\(\)*](#)

text()

Returns

Text content

See also:

[*setText\(\)*](#)

setRenderFlags(*renderFlags*)

Change the render flags

The default setting is *Qt.AlignCenter*

Parameters

renderFlags (*int*) – Bitwise OR of the flags used like in *QPainter.drawText()*

See also:

[*renderFlags\(\)*](#), [*qwt.text.QwtTextEngine.draw\(\)*](#)

renderFlags()

Returns

Render flags

See also:

[*setRenderFlags\(\)*](#)

setFont(*font*)

Set the font.

Parameters

font (*QFont*) – Font

Note: Setting the font might have no effect, when the text contains control sequences for setting fonts.

See also:

[*font\(\)*](#), [*usedFont\(\)*](#)

font()

Returns

Return the font

See also:

[*setFont\(\)*](#), [*usedFont\(\)*](#)

usedFont(*defaultFont*)

Return the font of the text, if it has one. Otherwise return defaultFont.

Parameters

defaultFont (*QFont*) – Default font

Returns

Font used for drawing the text

See also:

[*setFont\(\)*](#), [*font\(\)*](#)

setColor(*color*)

Set the pen color used for drawing the text.

Parameters

color (*QColor*) – Color

Note: Setting the color might have no effect, when the text contains control sequences for setting colors.

See also:

[*color\(\)*](#), [*usedColor\(\)*](#)

color()**Returns**

Return the pen color, used for painting the text

See also:

[*setColor\(\)*](#), [*usedColor\(\)*](#)

usedColor(*defaultColor*)

Return the color of the text, if it has one. Otherwise return defaultColor.

Parameters

defaultColor (*QColor*) – Default color

Returns

Color used for drawing the text

See also:

[*setColor\(\)*](#), [*color\(\)*](#)

setBorderRadius(*radius*)

Set the radius for the corners of the border frame

Parameters

radius (*float*) – Radius of a rounded corner

See also:

[*borderRadius\(\)*](#), [*setBorderPen\(\)*](#), [*setBackgroundBrush\(\)*](#)

borderRadius()**Returns**

Radius for the corners of the border frame

See also:

[*setBorderRadius\(\)*](#), [*borderPen\(\)*](#), [*backgroundBrush\(\)*](#)

setBorderPen(*pen*)

Set the background pen

Parameters

pen (*QPen*) – Background pen

See also:

[*borderPen\(\)*](#), [*setBackgroundBrush\(\)*](#)

borderPen()

Returns

Background pen

See also:

[*setBorderPen\(\)*](#), [*backgroundBrush\(\)*](#)

setBackgroundBrush(*brush*)

Set the background brush

Parameters

brush (*QBrush*) – Background brush

See also:

[*backgroundBrush\(\)*](#), [*setBorderPen\(\)*](#)

backgroundBrush()

Returns

Background brush

See also:

[*setBackgroundBrush\(\)*](#), [*borderPen\(\)*](#)

setPaintAttribute(*attribute*, *on=True*)

Change a paint attribute

Parameters

- **attribute** (*int*) – Paint attribute
- **on** (*bool*) – On/Off

Note: Used by [*setFont\(\)*](#), [*setColor\(\)*](#), [*setBorderPen\(\)*](#) and [*setBackgroundBrush\(\)*](#)

See also:

[*testPaintAttribute\(\)*](#)

testPaintAttribute(*attribute*)

Test a paint attribute

Parameters

attribute (*int*) – Paint attribute

Returns

True, if attribute is enabled

See also:

[`setPaintAttribute\(\)`](#)

setLayoutAttribute(*attribute*, *on=True*)

Change a layout attribute

Parameters

- **attribute** (*int*) – Layout attribute
- **on** (*bool*) – On/Off

See also:

[`testLayoutAttribute\(\)`](#)

testLayoutAttribute(*attribute*)

Test a layout attribute

Parameters

attribute (*int*) – Layout attribute

Returns

True, if attribute is enabled

See also:

[`setLayoutAttribute\(\)`](#)

heightForWidth(*width*, *defaultFont=None*)

Find the height for a given width

Parameters

- **width** (*float*) – Width
- **defaultFont** (*QFont*) – Font, used for the calculation if the text has no font

Returns

Calculated height

textSize(*defaultFont*)

Returns the size, that is needed to render text

:param QFont defaultFont Font, used for the calculation if the text has no font :return: Calculated size

draw(*painter*, *rect*)

Draw a text into a rectangle

Parameters

- **painter** (*QPainter*) – Painter
- **rect** (*QRectF*) – Rectangle

textEngine(*text=None*, *format_=None*)

Find the text engine for a text format

In case of *QwtText.AutoText* the first text engine (beside *QwtPlainTextEngine*) is returned, where *QwtTextEngine.mightRender* returns true. If there is none *QwtPlainTextEngine* is returned.

If no text engine is registered for the format *QwtPlainTextEngine* is returned.

Parameters

- **text** (*str*) – Text, needed in case of `AutoText`
- **format** (*int*) – Text format

Returns

Corresponding text engine

setTextEngine(*format_*, *engine*)

Assign/Replace a text engine for a text format

With `setTextEngine` it is possible to extend *PythonQwt* with other types of text formats.

For *QwtText.PlainText* it is not allowed to assign a engine to `None`.

Parameters

- **format** (*int*) – Text format
- **engine** (`qwt.text.QwtTextEngine`) – Text engine

See also:

[*setPaintAttribute\(\)*](#)

Warning: Using *QwtText.AutoText* does nothing.

5.6.2 QwtTextLabel

class `qwt.text.QwtTextLabel(*args)`

A Widget which displays a `QwtText`

class `QwtTextLabel(parent)`

Parameters

parent (*QWidget*) – Parent widget

class `QwtTextLabel([text=None][, parent=None])`

Parameters

- **text** (*str*) – Text
- **parent** (*QWidget*) – Parent widget

setPlainText(*text*)

Interface for the designer plugin - does the same as `setText()`

Parameters

text (*str*) – Text

See also:

[*plainText\(\)*](#)

plainText()

Interface for the designer plugin

Returns

Text as plain text

See also:

`setPlainText()`

setText(*text*, *textFormat*=0)

Change the label's text, keeping all other QwtText attributes

Parameters

- **text** (`qwt.text.QwtText` or `str`) – New text
- **textFormat** (`int`) – Format of text

See also:

`text()`

text()

Returns

Return the text

See also:

`setText()`

clear()

Clear the text and all *QwtText* attributes

indent()

Returns

Label's text indent in pixels

See also:

`setIndent()`

setIndent(*indent*)

Set label's text indent in pixels

Parameters

indent (`int`) – Indentation in pixels

See also:

`indent()`

margin()

Returns

Label's text indent in pixels

See also:

`setMargin()`

setMargin(*margin*)

Set label's margin in pixels

Parameters

margin (`int`) – Margin in pixels

See also:

`margin()`

sizeHint()

Return a size hint

minimumSizeHint()

Return a minimum size hint

heightForWidth(*width*)

Parameters

width (*int*) – Width

Returns

Preferred height for this widget, given the width.

paintEvent(*self*, *a0*: *QPaintEvent* | *None*)

drawContents(*painter*)

Redraw the text and focus indicator

Parameters

painter (*QPainter*) – Painter

drawText(*painter*, *textRect*)

Redraw the text

Parameters

- **painter** (*QPainter*) – Painter
- **textRect** (*QRectF*) – Text rectangle

textRect()

Calculate geometry for the text in widget coordinates

Returns

Geometry for the text

5.7 Text engines

5.7.1 QwtTextEngine

class `qwt.text.QwtTextEngine`

Abstract base class for rendering text strings

A text engine is responsible for rendering texts for a specific text format. They are used by *QwtText* to render a text.

QwtPlainTextEngine and *QwtRichTextEngine* are part of the *PythonQwt* library.

See also:

`qwt.text.QwtText.setTextEngine()`

heightForWidth(*font*, *flags*, *text*, *width*)

Find the height for a given width

Parameters

- **font** (*QFont*) – Font of the text

- **flags** (*int*) – Bitwise OR of the flags used like in QPainter::drawText
- **text** (*str*) – Text to be rendered
- **width** (*float*) – Width

Returns

Calculated height

textSize(*font, flags, text*)

Returns the size, that is needed to render text

Parameters

- **font** (*QFont*) – Font of the text
- **flags** (*int*) – Bitwise OR of the flags like in for QPainter::drawText
- **text** (*str*) – Text to be rendered

Returns

Calculated size

mightRender(*text*)

Test if a string can be rendered by this text engine

Parameters

text (*str*) – Text to be tested

Returns

True, if it can be rendered

textMargins(*font*)

Return margins around the texts

The textSize might include margins around the text, like QFontMetrics::descent(). In situations where texts need to be aligned in detail, knowing these margins might improve the layout calculations.

Parameters

font (*QFont*) – Font of the text

Returns

tuple (left, right, top, bottom) representing margins

draw(*painter, rect, flags, text*)

Draw the text in a clipping rectangle

Parameters

- **painter** (*QPainter*) – Painter
- **rect** (*QRectF*) – Clipping rectangle
- **flags** (*int*) – Bitwise OR of the flags like in for QPainter::drawText()
- **text** (*str*) – Text to be rendered

5.7.2 QwtPlainTextEngine

class `qwt.text.QwtPlainTextEngine`

A text engine for plain texts

QwtPlainTextEngine renders texts using the basic *Qt* classes *QPainter* and *QFontMetrics*.

heightForWidth(*font*, *flags*, *text*, *width*)

Find the height for a given width

Parameters

- **font** (*QFont*) – Font of the text
- **flags** (*int*) – Bitwise OR of the flags used like in *QPainter::drawText*
- **text** (*str*) – Text to be rendered
- **width** (*float*) – Width

Returns

Calculated height

textSize(*font*, *flags*, *text*)

Returns the size, that is needed to render text

Parameters

- **font** (*QFont*) – Font of the text
- **flags** (*int*) – Bitwise OR of the flags like in for *QPainter::drawText*
- **text** (*str*) – Text to be rendered

Returns

Calculated size

textMargins(*font*)

Return margins around the texts

The *textSize* might include margins around the text, like *QFontMetrics::descent()*. In situations where texts need to be aligned in detail, knowing these margins might improve the layout calculations.

Parameters

font (*QFont*) – Font of the text

Returns

tuple (left, right, top, bottom) representing margins

draw(*painter*, *rect*, *flags*, *text*)

Draw the text in a clipping rectangle

Parameters

- **painter** (*QPainter*) – Painter
- **rect** (*QRectF*) – Clipping rectangle
- **flags** (*int*) – Bitwise OR of the flags like in for *QPainter::drawText()*
- **text** (*str*) – Text to be rendered

mightRender(*text*)

Test if a string can be rendered by this text engine

Parameters

text (*str*) – Text to be tested

Returns

True, if it can be rendered

5.7.3 QwtRichTextEngine

class qwt.text.QwtRichTextEngine

A text engine for *Qt* rich texts

QwtRichTextEngine renders *Qt* rich texts using the classes of the Scribe framework of *Qt*.

heightForWidth(*font, flags, text, width*)

Find the height for a given width

Parameters

- **font** (*QFont*) – Font of the text
- **flags** (*int*) – Bitwise OR of the flags used like in *QPainter::drawText*
- **text** (*str*) – Text to be rendered
- **width** (*float*) – Width

Returns

Calculated height

textSize(*font, flags, text*)

Returns the size, that is needed to render text

Parameters

- **font** (*QFont*) – Font of the text
- **flags** (*int*) – Bitwise OR of the flags like in for *QPainter::drawText*
- **text** (*str*) – Text to be rendered

Returns

Calculated size

draw(*painter, rect, flags, text*)

Draw the text in a clipping rectangle

Parameters

- **painter** (*QPainter*) – Painter
- **rect** (*QRectF*) – Clipping rectangle
- **flags** (*int*) – Bitwise OR of the flags like in for *QPainter::drawText()*
- **text** (*str*) – Text to be rendered

mightRender(*text*)

Test if a string can be rendered by this text engine

Parameters

text (*str*) – Text to be tested

Returns

True, if it can be rendered

textMargins(font)

Return margins around the texts

The textSize might include margins around the text, like QFontMetrics::descent(). In situations where texts need to be aligned in detail, knowing these margins might improve the layout calculations.

Parameters

font (QFont) – Font of the text

Returns

tuple (left, right, top, bottom) representing margins

5.8 NumPy array to QImage

`qwt.toqimage.array_to_qimage(arr, copy=False)`

Convert NumPy array to QImage object

Parameters

- **arr** (*numpy.array*) – NumPy array
- **copy** (*bool*) – if True, make a copy of the array

Returns

QImage object

Private API:

5.9 QwtGraphic

class `qwt.graphic.QwtGraphic(*args)`

A paint device for scalable graphics

QwtGraphic is the representation of a graphic that is tailored for scalability. Like *QPicture* it will be initialized by *QPainter* operations and can be replayed later to any target paint device.

While the usual image representations *QImage* and *QPixmap* are not scalable *Qt* offers two paint devices, that might be candidates for representing a vector graphic:

- *QPicture*:

Unfortunately *QPicture* had been forgotten, when Qt4 introduced floating point based render engines. Its API is still on integers, what make it unusable for proper scaling.

- *QSvgRenderer*, *QSvgGenerator*:

Unfortunately *QSvgRenderer* hides too much information about its nodes in internal APIs, that are necessary for proper layout calculations. Also it is derived from *QObject* and can't be copied like *QImage/QPixmap*.

QwtGraphic maps all scalable drawing primitives to a *QPainterPath* and stores them together with the painter state changes (pen, brush, transformation ...) in a list of *QwtPaintCommands*. For being a complete *QPaintDevice* it also stores pixmaps or images, what is somehow against the idea of the class, because these objects can't be scaled without a loss in quality.

The main issue about scaling a *QwtGraphic* object are the pens used for drawing the outlines of the painter paths. While non cosmetic pens (*QPen.isCosmetic()*) are scaled with the same ratio as the path, cosmetic pens have a fixed width. A graphic might have paths with different pens - cosmetic and non-cosmetic.

QwtGraphic caches 2 different rectangles:

- control point rectangle:

The control point rectangle is the bounding rectangle of all control point rectangles of the painter paths, or the target rectangle of the pixmaps/images.

- bounding rectangle:

The bounding rectangle extends the control point rectangle by what is needed for rendering the outline with an unscaled pen.

Because the offset for drawing the outline depends on the shape of the painter path (the peak of a triangle is different than the flat side) scaling with a fixed aspect ratio always needs to be calculated from the control point rectangle.

class QwtGraphic

Initializes a null graphic

class QwtGraphic(*other*)

Copy constructor

Parameters

other ([qwt.graphic.QwtGraphic](#)) – Source

reset()

Clear all stored commands

isNull()

Return True, when no painter commands have been stored

isEmpty()

Return True, when the bounding rectangle is empty

setRenderHint(*hint*, *on=True*)

Toggle an render hint

testRenderHint(*hint*)

Test a render hint

boundingRect()

The bounding rectangle is the [controlPointRect\(\)](#) extended by the areas needed for rendering the outlines with unscaled pens.

Returns

Bounding rectangle of the graphic

See also:

[controlPointRect\(\)](#), [scaledBoundingRect\(\)](#)

controlPointRect()

The control point rectangle is the bounding rectangle of all control points of the paths and the target rectangles of the images/pixmaps.

Returns

Control point rectangle

See also:

boundingRect(), *scaledBoundingRect()*

scaledBoundingRect(*sx*, *sy*)

Calculate the target rectangle for scaling the graphic

Parameters

- **sx** (*float*) – Horizontal scaling factor
- **sy** (*float*) – Vertical scaling factor

Returns

Scaled bounding rectangle

Note: In case of paths that are painted with a cosmetic pen (see `QPen.isCosmetic()`) the target rectangle is different to multiplying the bounding rectangle.

See also:

boundingRect(), *controlPointRect()*

sizeMetrics()

Return Ceiled *defaultSize()*

setDefaultSize(*size*)

The default size is used in all methods rendering the graphic, where no size is explicitly specified. Assigning an empty size means, that the default size will be calculated from the bounding rectangle.

Parameters

size (*QSizeF*) – Default size

See also:

defaultSize(), *boundingRect()*

defaultSize()

When a non empty size has been assigned by `setDefaultSize()` this size will be returned. Otherwise the default size is the size of the bounding rectangle.

The default size is used in all methods rendering the graphic, where no size is explicitly specified.

Returns

Default size

See also:

setDefaultSize(), *boundingRect()*

render(**args*)

render(*painter*)

Replay all recorded painter commands

Parameters

painter (*QPainter*) – Qt painter

render(*painter*, *size*, *aspectRatioMode*)

Replay all recorded painter commands

The graphic is scaled to fit into the rectangle of the given size starting at (0, 0).

Parameters

- **painter** (*QPainter*) – Qt painter
- **size** (*QSizeF*) – Size for the scaled graphic
- **aspectRatioMode** (*Qt.AspectRatioMode*) – Mode how to scale

render(*painter, rect, aspectRatioMode*)

Replay all recorded painter commands

The graphic is scaled to fit into the given rectangle

Parameters

- **painter** (*QPainter*) – Qt painter
- **rect** (*QRectF*) – Rectangle for the scaled graphic
- **aspectRatioMode** (*Qt.AspectRatioMode*) – Mode how to scale

render(*painter, pos, aspectRatioMode*)

Replay all recorded painter commands

The graphic is scaled to the [defaultSize\(\)](#) and aligned to a position.

Parameters

- **painter** (*QPainter*) – Qt painter
- **pos** (*QPointF*) – Reference point, where to render
- **aspectRatioMode** (*Qt.AspectRatioMode*) – Mode how to scale

toPixmap(*args)

Convert the graphic to a *QPixmap*

All pixels of the pixmap get initialized by *Qt.transparent* before the graphic is scaled and rendered on it.

The size of the pixmap is the default size (ceiled to integers) of the graphic.

Returns

The graphic as pixmap in default size

See also:

[defaultSize\(\)](#), [toImage\(\)](#), [render\(\)](#)

toImage(*args)

toImage()

Convert the graphic to a *QImage*

All pixels of the image get initialized by 0 (transparent) before the graphic is scaled and rendered on it.

The format of the image is *QImage.Format_ARGB32_Premultiplied*.

The size of the image is the default size (ceiled to integers) of the graphic.

Returns

The graphic as image in default size

toImage(*size*[, *aspectRatioMode=Qt.IgnoreAspectRatio*])

Convert the graphic to a *QImage*

All pixels of the image get initialized by 0 (transparent) before the graphic is scaled and rendered on it.

The format of the image is *QImage.Format_ARGB32_Premultiplied*.

Parameters

- **size** (*QSize*) – Size of the image
- **aspectRatioMode** (*Qt.AspectRatioMode*) – Aspect ratio how to scale the graphic

Returns

The graphic as image

See also:

toPixmap(), *render()*

drawPath(*path*)

Store a path command in the command list

Parameters

path (*QPainterPath*) – Painter path

See also:

QPaintEngine.drawPath()

drawPixmap(*rect*, *pixmap*, *subRect*)

Store a pixmap command in the command list

Parameters

- **rect** (*QRectF*) – target rectangle
- **pixmap** (*QPixmap*) – Pixmap to be painted
- **subRect** (*QRectF*) – Rectangle of the pixmap to be painted

See also:

QPaintEngine.drawPixmap()

drawImage(*rect*, *image*, *subRect*, *flags*)

Store a image command in the command list

Parameters

- **rect** (*QRectF*) – target rectangle
- **image** (*QImage*) – Pixmap to be painted
- **subRect** (*QRectF*) – Rectangle of the pixmap to be painted
- **flags** (*Qt.ImageConversionFlags*) – Pixmap to be painted

See also:

QPaintEngine.drawImage()

updateState(*state*)

Store a state command in the command list

Parameters

state (*QPaintEngineState*) – State to be stored

See also:

QPaintEngine.updateState()

5.10 QwtInterval

class qwt.interval.**QwtInterval**(*minValue=0.0, maxValue=-1.0, borderFlags=None*)

A class representing an interval

The interval is represented by 2 doubles, the lower and the upper limit.

class **QwtInterval**(*minValue=0., maxValue=-1., borderFlags=None*)

Build an interval with from min/max values

Parameters

- **minValue** (*float*) – Minimum value
- **maxValue** (*float*) – Maximum value
- **borderFlags** (*int*) – Include/Exclude borders

setInterval(*minValue, maxValue, borderFlags=None*)

Assign the limits of the interval

Parameters

- **minValue** (*float*) – Minimum value
- **maxValue** (*float*) – Maximum value
- **borderFlags** (*int*) – Include/Exclude borders

setBorderFlags(*borderFlags*)

Change the border flags

Parameters

- borderFlags** (*int*) – Include/Exclude borders

See also:

[*borderFlags\(\)*](#)

borderFlags()

Returns

Border flags

See also:

[*setBorderFlags\(\)*](#)

setMinValue(*minValue*)

Assign the lower limit of the interval

Parameters

- minValue** (*float*) – Minimum value

setMaxValue(*maxValue*)

Assign the upper limit of the interval

Parameters

- maxValue** (*float*) – Maximum value

minValue()

Returns

Lower limit of the interval

maxValue()

Returns

Upper limit of the interval

isValid()

A interval is valid when `minValue() <= maxValue()`. In case of *QwtInterval.ExcludeBorders* it is true when `minValue() < maxValue()`

Returns

True, when the interval is valid

width()

The width of invalid intervals is 0.0, otherwise the result is `maxValue() - minValue()`.

Returns

the width of an interval

isNull()

Returns

true, if `isValid() && (minValue() >= maxValue())`

invalidate()

The limits are set to interval [0.0, -1.0]

See also:

[*isValid\(\)*](#)

normalized()

Normalize the limits of the interval

If `maxValue() < minValue()` the limits will be inverted.

Returns

Normalized interval

See also:

[*isValid\(\)*](#), [*inverted\(\)*](#)

inverted()

Invert the limits of the interval

Returns

Inverted interval

See also:

[*normalized\(\)*](#)

contains(*value*)

Test if a value is inside an interval

Parameters

value (*float*) – Value

Returns

true, if `value >= minValue() && value <= maxValue()`

unite(*other*)

Unite two intervals

Parameters

other (`qwt.interval.QwtInterval`) – other interval to united with

Returns

united interval

intersect(*other*)

Intersect two intervals

Parameters

other (`qwt.interval.QwtInterval`) – other interval to intersect with

Returns

intersected interval

intersects(*other*)

Test if two intervals overlap

Parameters

other (`qwt.interval.QwtInterval`) – other interval

Returns

True, when the intervals are intersecting

symmetrize(*value*)

Adjust the limit that is closer to value, so that value becomes the center of the interval.

Parameters

value (`float`) – Center

Returns

Interval with value as center

limited(*lowerBound*, *upperBound*)

Limit the interval, keeping the border modes

Parameters

- **lowerBound** (`float`) – Lower limit
- **upperBound** (`float`) – Upper limit

Returns

Limited interval

extend(*value*)

Extend the interval

If value is below `minValue()`, value becomes the lower limit. If value is above `maxValue()`, value becomes the upper limit.

`extend()` has no effect for invalid intervals

Parameters

value (`float`) – Value

Returns

extended interval

extend_fraction(*value*)

Extend the interval by a fraction of its width

Parameters**value** (*float*) – Fraction**Returns**

extended interval

5.11 QwtPlotDirectPainter

class qwt.plot_directpainter.**QwtPlotDirectPainter**(*parent=None*)

Painter object trying to paint incrementally

Often applications want to display samples while they are collected. When there are too many samples complete replots will be expensive to be processed in a collection cycle.

QwtPlotDirectPainter offers an API to paint subsets (f.e all additions points) without erasing/repainting the plot canvas.

On certain environments it might be important to calculate a proper clip region before painting. F.e. for Qt Embedded only the clipped part of the backing store will be copied to a (maybe unaccelerated) frame buffer.

Warning: Incremental painting will only help when no replot is triggered by another operation (like changing scales) and nothing needs to be erased.

Paint attributes:

- *QwtPlotDirectPainter.AtomicPainter*:

Initializing a *QPainter* is an expensive operation. When *AtomicPainter* is set each call of *drawSeries()* opens/closes a temporary *QPainter*. Otherwise *QwtPlotDirectPainter* tries to use the same *QPainter* as long as possible.

- *QwtPlotDirectPainter.FullRepaint*:

When *FullRepaint* is set the plot canvas is explicitly repainted after the samples have been rendered.

- *QwtPlotDirectPainter.CopyBackingStore*:

When *QwtPlotCanvas.BackingStore* is enabled the painter has to paint to the backing store and the widget. In certain situations/environments it might be faster to paint to the backing store only and then copy the backing store to the canvas. This flag can also be useful for settings, where Qt fills the the clip region with the widget background.

setAttribute(*attribute, on=True*)

Change an attribute

Parameters

- **attribute** (*int*) – Attribute to change
- **on** (*bool*) – On/Off

See also:*testAttribute()*

testAttribute(*attribute*)

Parameters

attribute (*int*) – Attribute to be tested

Returns

True, when attribute is enabled

See also:

[`setAttribute\(\)`](#)

setClipping(*enable*)

En/Disables clipping

Parameters

enable (*bool*) – Enables clipping is true, disable it otherwise

See also:

[`hasClipping\(\)`](#), [`clipRegion\(\)`](#), [`setClipRegion\(\)`](#)

hasClipping()

Returns

Return true, when clipping is enabled

See also:

[`setClipping\(\)`](#), [`clipRegion\(\)`](#), [`setClipRegion\(\)`](#)

setClipRegion(*region*)

Assign a clip region and enable clipping

Depending on the environment setting a proper clip region might improve the performance heavily. F.e. on Qt embedded only the clipped part of the backing store will be copied to a (maybe unaccelerated) frame buffer device.

Parameters

region (*QRegion*) – Clip region

See also:

[`hasClipping\(\)`](#), [`setClipping\(\)`](#), [`clipRegion\(\)`](#)

clipRegion()

Returns

Return Currently set clip region.

See also:

[`hasClipping\(\)`](#), [`setClipping\(\)`](#), [`setClipRegion\(\)`](#)

drawSeries(*seriesItem*, *from_*, *to*)

Draw a set of points of a seriesItem.

When observing a measurement while it is running, new points have to be added to an existing seriesItem. drawSeries() can be used to display them avoiding a complete redraw of the canvas.

Setting `plot().canvas().setAttribute(Qt.WA_PaintOutsidePaintEvent, True)` will result in faster painting, if the paint engine of the canvas widget supports this feature.

Parameters

- **seriesItem** (`qwt.plot_series.QwtPlotSeriesItem`) – Item to be painted

- **from** (*int*) – Index of the first point to be painted
- **to** (*int*) – Index of the last point to be painted. If *to* < 0 the series will be painted to its last point.

reset()

Close the internal QPainter

eventFilter(*self*, *a0*: *QObject* | *None*, *a1*: *QEvent* | *None*) → bool

5.12 QwtPlotLayout

class qwt.plot_layout.**QwtPlotLayout**

Layout engine for QwtPlot.

It is used by the *QwtPlot* widget to organize its internal widgets or by *QwtPlot.print()* to render its content to a QPaintDevice like a QPrinter, QPixmap/QImage or QSvgRenderer.

See also:

[*qwt.plot.QwtPlot.setPlotLayout\(\)*](#)

Valid options:

- *QwtPlotLayout.AlignScales*: Unused
- *QwtPlotLayout.IgnoreScrollbars*: Ignore the dimension of the scrollbars. There are no scrollbars, when the plot is not rendered to widgets.
- *QwtPlotLayout.IgnoreFrames*: Ignore all frames.
- *QwtPlotLayout.IgnoreLegend*: Ignore the legend.
- *QwtPlotLayout.IgnoreTitle*: Ignore the title.
- *QwtPlotLayout.IgnoreFooter*: Ignore the footer.

setCanvasMargin(*margin*, *axis=-1*)

Change a margin of the canvas. The margin is the space above/below the scale ticks. A negative margin will be set to -1, excluding the borders of the scales.

Parameters

- **margin** (*int*) – New margin
- **axisId** (*int*) – Axis index

See also:

[*canvasMargin\(\)*](#)

Warning: The margin will have no effect when *alignCanvasToScale()* is True

canvasMargin(*axisId*)

Parameters

- **axisId** (*int*) – Axis index

Returns

Margin around the scale tick borders

See also:

[`setCanvasMargin\(\)`](#)

setAlignCanvasToScales(*args)

Change the align-canvas-to-axis-scales setting.

setAlignCanvasToScales(on):

Set the align-canvas-to-axis-scales flag for all axes

Parameters

on (*bool*) – True/False

setAlignCanvasToScales(axisId, on):

Change the align-canvas-to-axis-scales setting. The canvas may:

- extend beyond the axis scale ends to maximize its size,
- align with the axis scale ends to control its size.

The *axisId* parameter is somehow confusing as it identifies a border of the plot and not the axes, that are aligned. F.e when *QwtPlot.yLeft* is set, the left end of the the x-axes (*QwtPlot.xTop*, *QwtPlot.xBottom*) is aligned.

Parameters

- **axisId** (*int*) – Axis index
- **on** (*bool*) – True/False

See also:

[`setAlignCanvasToScale\(\)`](#), [`alignCanvasToScale\(\)`](#)

alignCanvasToScale(axisId)

Return the align-canvas-to-axis-scales setting. The canvas may:

- extend beyond the axis scale ends to maximize its size
- align with the axis scale ends to control its size.

Parameters

axisId (*int*) – Axis index

Returns

align-canvas-to-axis-scales setting

See also:

[`setAlignCanvasToScale\(\)`](#), [`setCanvasMargin\(\)`](#)

setSpacing(spacing)

Change the spacing of the plot. The spacing is the distance between the plot components.

Parameters

spacing (*int*) – New spacing

See also:

[`setCanvasMargin\(\)`](#), [`spacing\(\)`](#)

spacing()

Returns

Spacing

See also:

[`margin\(\)`](#), [`setSpacing\(\)`](#)

setLegendPosition(*args)

Specify the position of the legend

setLegendPosition(pos, [ratio=0.]):

Specify the position of the legend

Parameters

- **pos** (*QwtPlot.LegendPosition*) – Legend position
- **ratio** (*float*) – Ratio between legend and the bounding rectangle of title, footer, canvas and axes

The legend will be shrunk if it would need more space than the given ratio. The ratio is limited to]0.0 .. 1.0]. In case of ≤ 0.0 it will be reset to the default ratio. The default vertical/horizontal ratio is 0.33/0.5.

Valid position values:

- *QwtPlot.LeftLegend*,
- *QwtPlot.RightLegend*,
- *QwtPlot.TopLegend*,
- *QwtPlot.BottomLegend*

See also:

[*setLegendPosition\(\)*](#)

legendPosition()**Returns**

Position of the legend

See also:

[*legendPosition\(\)*](#)

setLegendRatio(ratio)

Specify the relative size of the legend in the plot

Parameters

- **ratio** (*float*) – Ratio between legend and the bounding rectangle of title, footer, canvas and axes

The legend will be shrunk if it would need more space than the given ratio. The ratio is limited to]0.0 .. 1.0]. In case of ≤ 0.0 it will be reset to the default ratio. The default vertical/horizontal ratio is 0.33/0.5.

See also:

[*legendRatio\(\)*](#)

legendRatio()**Returns**

The relative size of the legend in the plot.

See also:

[*setLegendRatio\(\)*](#)

setTitleRect(rect)

Set the geometry for the title

This method is intended to be used from derived layouts overloading *activate()*

Parameters

- **rect** (*QRectF*) – Rectangle

See also:

titleRect(), *activate()*

titleRect()

Returns

Geometry for the title

See also:

invalidate(), *activate()*

setFooterRect(*rect*)

Set the geometry for the footer

This method is intended to be used from derived layouts overloading *activate()*

Parameters

rect (*QRectF*) – Rectangle

See also:

footerRect(), *activate()*

footerRect()

Returns

Geometry for the footer

See also:

invalidate(), *activate()*

setLegendRect(*rect*)

Set the geometry for the legend

This method is intended to be used from derived layouts overloading *activate()*

Parameters

rect (*QRectF*) – Rectangle for the legend

See also:

footerRect(), *activate()*

legendRect()

Returns

Geometry for the legend

See also:

invalidate(), *activate()*

setScaleRect(*axis*, *rect*)

Set the geometry for an axis

This method is intended to be used from derived layouts overloading *activate()*

Parameters

- **axisId** (*int*) – Axis index
- **rect** (*QRectF*) – Rectangle for the scale

See also:

`scaleRect()`, `activate()`

scaleRect(*axis*)

Parameters

axisId (*int*) – Axis index

Returns

Geometry for the scale

See also:

`invalidate()`, `activate()`

setCanvasRect(*rect*)

Set the geometry for the canvas

This method is intended to be used from derived layouts overloading `activate()`

Parameters

rect (*QRectF*) – Rectangle

See also:

`canvasRect()`, `activate()`

canvasRect()

Returns

Geometry for the canvas

See also:

`invalidate()`, `activate()`

invalidate()

Invalidate the geometry of all components.

See also:

`activate()`

minimumSizeHint(*plot*)

Parameters

plot (*qwt.plot.QwtPlot*) – Plot widget

Returns

Minimum size hint

See also:

`qwt.plot.QwtPlot.minimumSizeHint()`

layoutLegend(*options*, *rect*)

Find the geometry for the legend

Parameters

- **options** – Options how to layout the legend
- **rect** (*QRectF*) – Rectangle where to place the legend

Returns

Geometry for the legend

alignLegend(*canvasRect*, *legendRect*)

Align the legend to the canvas

Parameters

- **canvasRect** (*QRectF*) – Geometry of the canvas
- **legendRect** (*QRectF*) – Maximum geometry for the legend

Returns

Geometry for the aligned legend

expandLineBreaks(*options*, *rect*)

Expand all line breaks in text labels, and calculate the height of their widgets in orientation of the text.

Parameters

- **options** – Options how to layout the legend
- **rect** (*QRectF*) – Bounding rectangle for title, footer, axes and canvas.

Returns

tuple (*dimTitle*, *dimFooter*, *dimAxes*)

Returns:

- *dimTitle*: Expanded height of the title widget
- *dimFooter*: Expanded height of the footer widget
- *dimAxes*: Expanded heights of the axis in axis orientation.

alignScales(*options*, *canvasRect*, *scaleRect*)

Align the ticks of the axis to the canvas borders using the empty corners.

Parameters

- **options** – Options how to layout the legend
- **canvasRect** (*QRectF*) – Geometry of the canvas (IN/OUT)
- **scaleRect** (*QRectF*) – Geometry of the scales (IN/OUT)

activate(*plot*, *plotRect*, *options=0*)

Recalculate the geometry of all components.

Parameters

- **plot** ([qwt.plot.QwtPlot](#)) – Plot to be layout
- **plotRect** (*QRectF*) – Rectangle where to place the components
- **options** – Layout options

5.13 Plotting series item

5.13.1 QwtPlotSeriesItem

class `qwt.plot_series.QwtPlotSeriesItem`(*title*)

Base class for plot items representing a series of samples

setOrientation(*orientation*)

Set the orientation of the item. Default is *Qt.Horizontal*.

The *orientation()* might be used in specific way by a plot item. F.e. a *QwtPlotCurve* uses it to identify how to display the curve int *QwtPlotCurve.Steps* or *QwtPlotCurve.Sticks* style.

See also:

:py:meth`orientation()`

orientation()

Returns

Orientation of the plot item

See also:

:py:meth`setOrientation()`

draw(*painter*, *xMap*, *yMap*, *canvasRect*)

Draw the complete series

Parameters

- **painter** (*QPainter*) – Painter
- **xMap** (`qwt.scale_map.QwtScaleMap`) – Maps x-values into pixel coordinates.
- **yMap** (`qwt.scale_map.QwtScaleMap`) – Maps y-values into pixel coordinates.
- **canvasRect** (*QRectF*) – Contents rectangle of the canvas

drawSeries(*painter*, *xMap*, *yMap*, *canvasRect*, *from_*, *to*)

Draw a subset of the samples

Parameters

- **painter** (*QPainter*) – Painter
- **xMap** (`qwt.scale_map.QwtScaleMap`) – Maps x-values into pixel coordinates.
- **yMap** (`qwt.scale_map.QwtScaleMap`) – Maps y-values into pixel coordinates.
- **canvasRect** (*QRectF*) – Contents rectangle of the canvas
- **from** (*int*) – Index of the first point to be painted
- **to** (*int*) – Index of the last point to be painted. If *to* < 0 the curve will be painted to its last point.

See also:

This method is implemented in *qwt.plot_curve.QwtPlotCurve*

boundingRect()**Returns**

An invalid bounding rect: QRectF(1.0, 1.0, -2.0, -2.0)

Note: A width or height < 0.0 is ignored by the autoscaler

5.13.2 QwtSeriesData**class qwt.plot_series.QwtSeriesData**

Abstract interface for iterating over samples

PythonQwt offers several implementations of the QwtSeriesData API, but in situations, where data of an application specific format needs to be displayed, without having to copy it, it is recommended to implement an individual data access.

A subclass of *QwtSeriesData* must implement:

- **size():**
Should return number of data points.
- **sample()**
Should return values x and y values of the sample at specific position as QPointF object.
- **boundingRect()**
Should return the bounding rectangle of the data series. It is used for autoscaling and might help certain algorithms for displaying the data. The member *_boundingRect* is intended for caching the calculated rectangle.

setRectOfInterest(rect)

Set a the “rect of interest”

QwtPlotSeriesItem defines the current area of the plot canvas as “rectangle of interest” (*QwtPlotSeriesItem::updateScaleDiv()*). It can be used to implement different levels of details.

The default implementation does nothing.

Parameters

rect (*QRectF*) – Rectangle of interest

size()**Returns**

Number of samples

sample(i)

Return a sample

Parameters

i (*int*) – Index

Returns

Sample at position i

boundingRect()

Calculate the bounding rect of all samples

The bounding rect is necessary for autoscaling and can be used for a couple of painting optimizations.

Returns

Bounding rectangle

5.13.3 QwtPointArrayData

class qwt.plot_series.QwtPointArrayData(*x=None, y=None, size=None, finite=None*)

Interface for iterating over two array objects

class QwtCQwtPointArrayDataolorMap(*x, y[, size=None]*)

Parameters

- **x** (*list or tuple or numpy.array*) – Array of x values
- **y** (*list or tuple or numpy.array*) – Array of y values
- **size** (*int*) – Size of the x and y arrays
- **finite** (*bool*) – if True, keep only finite array elements (remove all infinity and not a number values), otherwise do not filter array elements

boundingRect()

Calculate the bounding rectangle

The bounding rectangle is calculated once by iterating over all points and is stored for all following requests.

Returns

Bounding rectangle

size()**Returns**

Size of the data set

sample(index)**Parameters**

index (*int*) – Index

Returns

Sample at position *index*

xData()**Returns**

Array of the x-values

yData()**Returns**

Array of the y-values

5.13.4 QwtSeriesStore

class `qwt.plot_series.QwtSeriesStore`

Class storing a *QwtSeriesData* object

QwtSeriesStore and *QwtPlotSeriesItem* are intended as base classes for all plot items iterating over a series of samples.

setData(*series*)

Assign a series of samples

Parameters

series (`qwt.plot_series.QwtSeriesData`) – Data

Warning: The item takes ownership of the data object, deleting it when its not used anymore.

data()

Returns

the series data

sample(*index*)

Parameters

index (*int*) – Index

Returns

Sample at position index

dataSize()

Returns

Number of samples of the series

See also:

`setData()`, `qwt.plot_series.QwtSeriesData.size()`

dataRect()

Returns

Bounding rectangle of the series or an invalid rectangle, when no series is stored

See also:

`qwt.plot_series.QwtSeriesData.boundingRect()`

setRectOfInterest(*rect*)

Set a the “rect of interest” for the series

Parameters

rect (*QRectF*) – Rectangle of interest

See also:

`qwt.plot_series.QwtSeriesData.setRectOfInterest()`

swapData(*series*)

Replace a series without deleting the previous one

Parameters**series** (`qwt.plot_series.QwtSeriesData`) – New series**Returns**

Previously assigned series

5.14 Coordinate transformations

5.14.1 QwtTransform

class `qwt.transform.QwtTransform`

A transformation between coordinate systems

QwtTransform manipulates values, when being mapped between the scale and the paint device coordinate system.

A transformation consists of 2 methods:

- `transform`
- `invTransform`

where one is the inverse function of the other.

When `p1`, `p2` are the boundaries of the paint device coordinates and `s1`, `s2` the boundaries of the scale, `QwtScaleMap` uses the following calculations:

$$p = p1 + (p2 - p1) * (T(s) - T(s1) / (T(s2) - T(s1)))$$
$$s = invT(T(s1) + (T(s2) - T(s1)) * (p - p1) / (p2 - p1))$$

bounded(*value*)

Modify value to be a valid value for the transformation. The default implementation does nothing.

transform(*value*)

Transformation function

Parameters**value** (*float*) – Value**Returns**

Modified value

See also:[`invTransform\(\)`](#)**invTransform**(*value*)

Inverse transformation function

Parameters**value** (*float*) – Value**Returns**

Modified value

See also:[`transform\(\)`](#)

copy()

Returns

Clone of the transformation

The default implementation does nothing.

5.14.2 QwtNullTransform

class qwt.transform.QwtNullTransform

transform(*value*)

Transformation function

Parameters

value (*float*) – Value

Returns

Modified value

See also:

[*invTransform\(\)*](#)

invTransform(*value*)

Inverse transformation function

Parameters

value (*float*) – Value

Returns

Modified value

See also:

[*transform\(\)*](#)

copy()

Returns

Clone of the transformation

5.14.3 QwtLogTransform

class qwt.transform.QwtLogTransform

Logarithmic transformation

QwtLogTransform modifies the values using *numpy.log()* and *numpy.exp()*.

Note: In the calculations of *QwtScaleMap* the base of the log function has no effect on the mapping. So *QwtLogTransform* can be used for logarithmic scale in base 2 or base 10 or any other base.

Extremum values:

- *QwtLogTransform.LogMin*: Smallest allowed value for logarithmic scales: 1.0e-150
- *QwtLogTransform.LogMax*: Largest allowed value for logarithmic scales: 1.0e150

bounded(*value*)

Modify value to be a valid value for the transformation.

Parameters

value (*float*) – Value to be bounded

Returns

Value modified

transform(*value*)

Transformation function

Parameters

value (*float*) – Value

Returns

Modified value

See also:

[*invTransform\(\)*](#)

invTransform(*value*)

Inverse transformation function

Parameters

value (*float*) – Value

Returns

Modified value

See also:

[*transform\(\)*](#)

copy()

Returns

Clone of the transformation

5.14.4 QwtPowerTransform

class `qwt.transform.QwtPowerTransform`(*exponent*)

A transformation using *numpy.pow()*

QwtPowerTransform preserves the sign of a value. F.e. a transformation with a factor of 2 transforms a value of -3 to -9 and v.v. Thus *QwtPowerTransform* can be used for scales including negative values.

transform(*value*)

Transformation function

Parameters

value (*float*) – Value

Returns

Modified value

See also:

[*invTransform\(\)*](#)

invTransform(*value*)

Inverse transformation function

Parameters

value (*float*) – Value

Returns

Modified value

See also:

[*transform\(\)*](#)

copy()**Returns**

Clone of the transformation

Indices and tables:

- [genindex](#)
- [search](#)

PYTHON MODULE INDEX

C

`qwt.color_map`, 126

g

`qwt.graphic`, 184

i

`qwt.interval`, 189

l

`qwt.legend`, 119

p

`qwt.plot`, 75

`qwt.plot_canvas`, 96

`qwt.plot_curve`, 104

`qwt.plot_directpainter`, 192

`qwt.plot_grid`, 100

`qwt.plot_layout`, 194

`qwt.plot_marker`, 113

`qwt.plot_renderer`, 128

`qwt.plot_series`, 200

q

`qwt`, ??

S

`qwt.scale_div`, 142

`qwt.scale_draw`, 152

`qwt.scale_engine`, 145

`qwt.scale_map`, 132

`qwt.scale_widget`, 135

`qwt.symbol`, 163

t

`qwt.text`, 172

`qwt.toqimage`, 184

`qwt.transform`, 204

A

activate() (*qwt.plot_layout.QwtPlotLayout method*), 199
 align() (*qwt.scale_engine.QwtLinearScaleEngine method*), 150
 align() (*qwt.scale_engine.QwtLogScaleEngine method*), 151
 alignCanvasToScale() (*qwt.plot_layout.QwtPlotLayout method*), 195
 alignLegend() (*qwt.plot_layout.QwtPlotLayout method*), 199
 alignment() (*qwt.scale_draw.QwtScaleDraw method*), 157
 alignment() (*qwt.scale_widget.QwtScaleWidget method*), 136
 alignScales() (*qwt.plot_layout.QwtPlotLayout method*), 199
 array_to_qimage() (*in module qwt.toqimage*), 184
 attach() (*qwt.plot.QwtPlotItem method*), 90
 attachItem() (*qwt.plot.QwtPlot method*), 89
 attributes() (*qwt.scale_engine.QwtScaleEngine method*), 148
 autoRefresh() (*qwt.plot.QwtPlot method*), 84
 autoReplot() (*qwt.plot.QwtPlot method*), 84
 autoScale() (*qwt.scale_engine.QwtLinearScaleEngine method*), 149
 autoScale() (*qwt.scale_engine.QwtLogScaleEngine method*), 150
 autoScale() (*qwt.scale_engine.QwtScaleEngine method*), 145
 axisAutoScale() (*qwt.plot.QwtPlot method*), 78
 axisEnabled() (*qwt.plot.QwtPlot method*), 78
 axisFont() (*qwt.plot.QwtPlot method*), 78
 axisInterval() (*qwt.plot.QwtPlot method*), 80
 axisMargin() (*qwt.plot.QwtPlot method*), 79
 axisMaxMajor() (*qwt.plot.QwtPlot method*), 78
 axisMaxMinor() (*qwt.plot.QwtPlot method*), 79
 axisScaleDiv() (*qwt.plot.QwtPlot method*), 79
 axisScaleDraw() (*qwt.plot.QwtPlot method*), 79
 axisScaleEngine() (*qwt.plot.QwtPlot method*), 78
 axisStepSize() (*qwt.plot.QwtPlot method*), 79

axisTitle() (*qwt.plot.QwtPlot method*), 80
 axisValid() (*qwt.plot.QwtPlot method*), 88
 axisWidget() (*qwt.plot.QwtPlot method*), 78

B

backgroundBrush() (*qwt.text.QwtText method*), 176
 backingStore() (*qwt.plot_canvas.QwtPlotCanvas method*), 98
 base() (*qwt.scale_engine.QwtScaleEngine method*), 149
 baseline() (*qwt.plot_curve.QwtPlotCurve method*), 112
 borderFlags() (*qwt.interval.QwtInterval method*), 189
 borderPath() (*qwt.plot_canvas.QwtPlotCanvas method*), 99
 borderPen() (*qwt.text.QwtText method*), 176
 borderRadius() (*qwt.plot_canvas.QwtPlotCanvas method*), 99
 borderRadius() (*qwt.text.QwtText method*), 175
 bounded() (*qwt.scale_div.QwtScaleDiv method*), 144
 bounded() (*qwt.transform.QwtLogTransform method*), 205
 bounded() (*qwt.transform.QwtTransform method*), 204
 boundingLabelRect() (*qwt.scale_draw.QwtScaleDraw method*), 160
 boundingRect() (*qwt.graphic.QwtGraphic method*), 185
 boundingRect() (*qwt.plot.QwtPlotItem method*), 95
 boundingRect() (*qwt.plot_marker.QwtPlotMarker method*), 118
 boundingRect() (*qwt.plot_series.QwtPlotSeriesItem method*), 200
 boundingRect() (*qwt.plot_series.QwtPointArrayData method*), 202
 boundingRect() (*qwt.plot_series.QwtSeriesData method*), 201
 boundingRect() (*qwt.symbol.QwtSymbol method*), 171
 brush() (*qwt.plot_curve.QwtPlotCurve method*), 108
 brush() (*qwt.symbol.QwtSymbol method*), 169
 buildCanvasMaps() (*qwt.plot_renderer.QwtPlotRenderer method*), 132

- buildInterval() (*qwt.scale_engine.QwtScaleEngine* method), 147
 buildMajorTicks() (*qwt.scale_engine.QwtLinearScaleEngine* method), 150
 buildMajorTicks() (*qwt.scale_engine.QwtLogScaleEngine* method), 151
 buildMinorTicks() (*qwt.scale_engine.QwtLinearScaleEngine* method), 150
 buildMinorTicks() (*qwt.scale_engine.QwtLogScaleEngine* method), 151
 buildTicks() (*qwt.scale_engine.QwtLinearScaleEngine* method), 149
 buildTicks() (*qwt.scale_engine.QwtLogScaleEngine* method), 151
- ## C
- cachePolicy() (*qwt.symbol.QwtSymbol* method), 165
 canvas() (*qwt.plot.QwtPlot* method), 86
 canvasBackground() (*qwt.plot.QwtPlot* method), 87
 canvasMap() (*qwt.plot.QwtPlot* method), 87
 canvasMargin() (*qwt.plot_layout.QwtPlotLayout* method), 194
 canvasRect() (*qwt.plot_layout.QwtPlotLayout* method), 198
 clear() (*qwt.text.QwtTextLabel* method), 179
 clipRegion() (*qwt.plot_directpainter.QwtPlotDirectPainter* method), 193
 closePolyline() (*qwt.plot_curve.QwtPlotCurve* method), 111
 closestPoint() (*qwt.plot_curve.QwtPlotCurve* method), 112
 color() (*qwt.color_map.QwtAlphaColorMap* method), 128
 color() (*qwt.color_map.QwtColorMap* method), 126
 color() (*qwt.text.QwtText* method), 175
 colorBarInterval() (*qwt.scale_widget.QwtScaleWidget* method), 141
 colorBarRect() (*qwt.scale_widget.QwtScaleWidget* method), 138
 colorBarWidth() (*qwt.scale_widget.QwtScaleWidget* method), 141
 colorMap() (*qwt.scale_widget.QwtScaleWidget* method), 141
 colorTable() (*qwt.color_map.QwtColorMap* method), 126
 contains() (*qwt.interval.QwtInterval* method), 190
 contains() (*qwt.scale_div.QwtScaleDiv* method), 144
 contains() (*qwt.scale_engine.QwtScaleEngine* method), 147
 contentsWidget() (*qwt.legend.QwtLegend* method), 124
 controlPointRect() (*qwt.graphic.QwtGraphic* method), 185
 copy() (*qwt.transform.QwtLogTransform* method), 206
 copy() (*qwt.transform.QwtNullTransform* method), 205
 copy() (*qwt.transform.QwtPowerTransform* method), 207
 copy() (*qwt.transform.QwtTransform* method), 204
 createWidget() (*qwt.legend.QwtLegend* method), 124
- ## D
- data() (*qwt.legend.QwtLegendLabel* method), 120
 data() (*qwt.plot_series.QwtSeriesStore* method), 203
 dataRect() (*qwt.plot_series.QwtSeriesStore* method), 203
 dataSize() (*qwt.plot_series.QwtSeriesStore* method), 203
 defaultItemMode() (*qwt.legend.QwtLegend* method), 123
 defaultSize() (*qwt.graphic.QwtGraphic* method), 186
 detach() (*qwt.plot.QwtPlotItem* method), 90
 detachItems() (*qwt.plot.QwtPlot* method), 77
 dimForLength() (*qwt.scale_widget.QwtScaleWidget* method), 139
 directPaint() (*qwt.plot_curve.QwtPlotCurve* method), 108
 discardFlags() (*qwt.plot_renderer.QwtPlotRenderer* method), 129
 divideInterval() (*qwt.scale_engine.QwtScaleEngine* method), 147
 divideScale() (*qwt.scale_engine.QwtLinearScaleEngine* method), 149
 divideScale() (*qwt.scale_engine.QwtLogScaleEngine* method), 151
 divideScale() (*qwt.scale_engine.QwtScaleEngine* method), 145
 draw() (*qwt.plot_grid.QwtPlotGrid* method), 102
 draw() (*qwt.plot_marker.QwtPlotMarker* method), 115
 draw() (*qwt.plot_series.QwtPlotSeriesItem* method), 200
 draw() (*qwt.scale_draw.QwtAbstractScaleDraw* method), 154
 draw() (*qwt.scale_widget.QwtScaleWidget* method), 138
 draw() (*qwt.text.QwtPlainTextEngine* method), 182
 draw() (*qwt.text.QwtRichTextEngine* method), 183
 draw() (*qwt.text.QwtText* method), 177
 draw() (*qwt.text.QwtTextEngine* method), 181
 drawBackbone() (*qwt.scale_draw.QwtAbstractScaleDraw* method), 153
 drawBackbone() (*qwt.scale_draw.QwtScaleDraw* method), 159
 drawBorder() (*qwt.plot_canvas.QwtPlotCanvas* method), 99
 drawCanvas() (*qwt.plot.QwtPlot* method), 87
 drawColorBar() (*qwt.scale_widget.QwtScaleWidget* method), 138
 drawContents() (*qwt.text.QwtTextLabel* method), 180
 drawCurve() (*qwt.plot_curve.QwtPlotCurve* method), 108

- `drawDots()` (*qwt.plot_curve.QwtPlotCurve method*), 109
`drawFocusIndicator()` (*qwt.plot_canvas.QwtPlotCanvas method*), 99
`drawImage()` (*qwt.graphic.QwtGraphic method*), 188
`drawItems()` (*qwt.plot.QwtPlot method*), 87
`drawLabel()` (*qwt.plot_marker.QwtPlotMarker method*), 115
`drawLabel()` (*qwt.scale_draw.QwtAbstractScaleDraw method*), 153
`drawLabel()` (*qwt.scale_draw.QwtScaleDraw method*), 160
`drawLines()` (*qwt.plot_curve.QwtPlotCurve method*), 109
`drawLines()` (*qwt.plot_marker.QwtPlotMarker method*), 115
`drawPath()` (*qwt.graphic.QwtGraphic method*), 188
`drawPixmap()` (*qwt.graphic.QwtGraphic method*), 188
`drawSeries()` (*qwt.plot_curve.QwtPlotCurve method*), 108
`drawSeries()` (*qwt.plot_directpainter.QwtPlotDirectPainter method*), 193
`drawSeries()` (*qwt.plot_series.QwtPlotSeriesItem method*), 200
`drawSteps()` (*qwt.plot_curve.QwtPlotCurve method*), 110
`drawSticks()` (*qwt.plot_curve.QwtPlotCurve method*), 109
`drawSymbol()` (*qwt.symbol.QwtSymbol method*), 171
`drawSymbols()` (*qwt.plot_curve.QwtPlotCurve method*), 111
`drawSymbols()` (*qwt.symbol.QwtSymbol method*), 170
`drawText()` (*qwt.text.QwtTextLabel method*), 180
`drawTick()` (*qwt.scale_draw.QwtAbstractScaleDraw method*), 152
`drawTick()` (*qwt.scale_draw.QwtScaleDraw method*), 158
`drawTitle()` (*qwt.scale_widget.QwtScaleWidget method*), 139
- ## E
- `enableAxis()` (*qwt.plot.QwtPlot method*), 80
`enableComponent()` (*qwt.scale_draw.QwtAbstractScaleDraw method*), 153
`enableX()` (*qwt.plot_grid.QwtPlotGrid method*), 100
`enableXMin()` (*qwt.plot_grid.QwtPlotGrid method*), 101
`enableY()` (*qwt.plot_grid.QwtPlotGrid method*), 100
`enableYMin()` (*qwt.plot_grid.QwtPlotGrid method*), 101
`endBorderDist()` (*qwt.scale_widget.QwtScaleWidget method*), 138
`event()` (*qwt.plot.QwtPlot method*), 84
`event()` (*qwt.plot_canvas.QwtPlotCanvas method*), 99
`eventFilter()` (*qwt.legend.QwtLegend method*), 125
`eventFilter()` (*qwt.plot.QwtPlot method*), 84
`eventFilter()` (*qwt.plot_directpainter.QwtPlotDirectPainter method*), 194
`expandLineBreaks()` (*qwt.plot_layout.QwtPlotLayout method*), 199
`exportTo()` (*qwt.plot.QwtPlot method*), 89
`exportTo()` (*qwt.plot_renderer.QwtPlotRenderer method*), 132
`extend()` (*qwt.interval.QwtInterval method*), 191
`extend_fraction()` (*qwt.interval.QwtInterval method*), 191
`extent()` (*qwt.scale_draw.QwtAbstractScaleDraw method*), 152
`extent()` (*qwt.scale_draw.QwtScaleDraw method*), 158
- ## F
- `fillCurve()` (*qwt.plot_curve.QwtPlotCurve method*), 111
`flatStyle()` (*qwt.plot.QwtPlot method*), 77
`focusIndicator()` (*qwt.plot_canvas.QwtPlotCanvas method*), 98
`font()` (*qwt.text.QwtText method*), 174
`footer()` (*qwt.plot.QwtPlot method*), 85
`footerLabel()` (*qwt.plot.QwtPlot method*), 85
`footerRect()` (*qwt.plot_layout.QwtPlotLayout method*), 197
- ## G
- `getBorderDistHint()` (*qwt.scale_draw.QwtScaleDraw method*), 157
`getBorderDistHint()` (*qwt.scale_widget.QwtScaleWidget method*), 139
`getCanvasMarginHint()` (*qwt.plot.QwtPlotItem method*), 95
`getCanvasMarginsHint()` (*qwt.plot.QwtPlot method*), 86
`getMinBorderDist()` (*qwt.scale_widget.QwtScaleWidget method*), 140
`graphic()` (*qwt.symbol.QwtSymbol method*), 168
- ## H
- `hasClipping()` (*qwt.plot_directpainter.QwtPlotDirectPainter method*), 193
`hasComponent()` (*qwt.scale_draw.QwtAbstractScaleDraw method*), 153
`hasRole()` (*qwt.legend.QwtLegendData method*), 119
`heightForWidth()` (*qwt.legend.QwtLegend method*), 125
`heightForWidth()` (*qwt.text.QwtPlainTextEngine method*), 182

- heightForWidth() (*qwt.text.QwtRichTextEngine* method), 183
 heightForWidth() (*qwt.text.QwtText* method), 177
 heightForWidth() (*qwt.text.QwtTextEngine* method), 180
 heightForWidth() (*qwt.text.QwtTextLabel* method), 180
 hide() (*qwt.plot.QwtPlotItem* method), 93
 horizontalScrollBar() (*qwt.legend.QwtLegend* method), 124
I
 icon() (*qwt.legend.QwtLegendData* method), 120
 icon() (*qwt.legend.QwtLegendLabel* method), 121
 icon() (*qwt.plot.QwtPlotItem* method), 91
 indent() (*qwt.text.QwtTextLabel* method), 179
 init() (*qwt.plot_curve.QwtPlotCurve* method), 105
 initAxesData() (*qwt.plot.QwtPlot* method), 78
 initScale() (*qwt.scale_widget.QwtScaleWidget* method), 135
 insertItem() (*qwt.plot.QwtPlot* method), 76
 insertLegend() (*qwt.plot.QwtPlot* method), 88
 intersect() (*qwt.interval.QwtInterval* method), 191
 intersects() (*qwt.interval.QwtInterval* method), 191
 interval() (*qwt.scale_div.QwtScaleDiv* method), 143
 invalidate() (*qwt.interval.QwtInterval* method), 190
 invalidate() (*qwt.plot_layout.QwtPlotLayout* method), 198
 invalidateBackingStore() (*qwt.plot_canvas.QwtPlotCanvas* method), 98
 invalidateCache() (*qwt.scale_draw.QwtAbstractScaleDraw* method), 156
 invalidateCache() (*qwt.symbol.QwtSymbol* method), 171
 invert() (*qwt.scale_div.QwtScaleDiv* method), 144
 inverted() (*qwt.interval.QwtInterval* method), 190
 inverted() (*qwt.scale_div.QwtScaleDiv* method), 144
 invTransform() (*qwt.plot.QwtPlot* method), 80
 invTransform() (*qwt.scale_map.QwtScaleMap* method), 135
 invTransform() (*qwt.transform.QwtLogTransform* method), 206
 invTransform() (*qwt.transform.QwtNullTransform* method), 205
 invTransform() (*qwt.transform.QwtPowerTransform* method), 206
 invTransform() (*qwt.transform.QwtTransform* method), 204
 invTransform_scalar() (*qwt.scale_map.QwtScaleMap* method), 133
 isChecked() (*qwt.legend.QwtLegendLabel* method), 121
 isColorBarEnabled() (*qwt.scale_widget.QwtScaleWidget* method), 141
 isDown() (*qwt.legend.QwtLegendLabel* method), 122
 isEmpty() (*qwt.graphic.QwtGraphic* method), 185
 isEmpty() (*qwt.scale_div.QwtScaleDiv* method), 144
 isEmpty() (*qwt.text.QwtText* method), 173
 isIncreasing() (*qwt.scale_div.QwtScaleDiv* method), 144
 isInverting() (*qwt.scale_map.QwtScaleMap* method), 133
 isNull() (*qwt.graphic.QwtGraphic* method), 185
 isNull() (*qwt.interval.QwtInterval* method), 190
 isPinPointEnabled() (*qwt.symbol.QwtSymbol* method), 170
 isValid() (*qwt.interval.QwtInterval* method), 190
 isValid() (*qwt.legend.QwtLegendData* method), 120
 isVisible() (*qwt.plot.QwtPlotItem* method), 93
 itemChanged() (*qwt.plot.QwtPlotItem* method), 93
 itemInfo() (*qwt.legend.QwtLegend* method), 126
 itemList() (*qwt.plot.QwtPlot* method), 77
 itemMode() (*qwt.legend.QwtLegendLabel* method), 121
K
 keyPressEvent() (*qwt.legend.QwtLegendLabel* method), 122
 keyReleaseEvent() (*qwt.legend.QwtLegendLabel* method), 122
L
 label() (*qwt.plot_marker.QwtPlotMarker* method), 116
 label() (*qwt.scale_draw.QwtAbstractScaleDraw* method), 156
 labelAlignment() (*qwt.plot_marker.QwtPlotMarker* method), 117
 labelAlignment() (*qwt.scale_draw.QwtScaleDraw* method), 162
 labelAutoSize() (*qwt.scale_draw.QwtScaleDraw* method), 162
 labelOrientation() (*qwt.plot_marker.QwtPlotMarker* method), 117
 labelPosition() (*qwt.scale_draw.QwtScaleDraw* method), 158
 labelRect() (*qwt.scale_draw.QwtScaleDraw* method), 160
 labelRotation() (*qwt.scale_draw.QwtScaleDraw* method), 161
 labelSize() (*qwt.scale_draw.QwtScaleDraw* method), 161
 labelTransformation() (*qwt.scale_draw.QwtScaleDraw* method), 160
 layoutFlags() (*qwt.plot_renderer.QwtPlotRenderer* method), 130

- layoutLegend() (*qwt.plot_layout.QwtPlotLayout method*), 198
 layoutScale() (*qwt.scale_widget.QwtScaleWidget method*), 138
 legend() (*qwt.plot.QwtPlot method*), 85
 legendChanged() (*qwt.plot.QwtPlotItem method*), 94
 legendData() (*qwt.plot.QwtPlotItem method*), 95
 legendIcon() (*qwt.plot.QwtPlotItem method*), 93
 legendIcon() (*qwt.plot_curve.QwtPlotCurve method*), 112
 legendIcon() (*qwt.plot_marker.QwtPlotMarker method*), 118
 legendIconSize() (*qwt.plot.QwtPlotItem method*), 93
 legendPosition() (*qwt.plot_layout.QwtPlotLayout method*), 196
 legendRatio() (*qwt.plot_layout.QwtPlotLayout method*), 196
 legendRect() (*qwt.plot_layout.QwtPlotLayout method*), 197
 legendWidget() (*qwt.legend.QwtLegend method*), 125
 legendWidgets() (*qwt.legend.QwtLegend method*), 125
 length() (*qwt.scale_draw.QwtScaleDraw method*), 160
 limited() (*qwt.interval.QwtInterval method*), 191
 linePen() (*qwt.plot_marker.QwtPlotMarker method*), 118
 lineStyle() (*qwt.plot_marker.QwtPlotMarker method*), 116
 lowerBound() (*qwt.scale_div.QwtScaleDiv method*), 143
 lowerMargin() (*qwt.scale_engine.QwtScaleEngine method*), 146
- ## M
- majorPen() (*qwt.plot_grid.QwtPlotGrid method*), 102
 make() (*qwt.plot_curve.QwtPlotCurve class method*), 105
 make() (*qwt.plot_grid.QwtPlotGrid class method*), 100
 make() (*qwt.plot_marker.QwtPlotMarker class method*), 113
 make() (*qwt.symbol.QwtSymbol class method*), 165
 make() (*qwt.text.QwtText class method*), 173
 margin() (*qwt.scale_widget.QwtScaleWidget method*), 138
 margin() (*qwt.text.QwtTextLabel method*), 179
 maxColumns() (*qwt.legend.QwtLegend method*), 123
 maxLabelHeight() (*qwt.scale_draw.QwtScaleDraw method*), 162
 maxLabelWidth() (*qwt.scale_draw.QwtScaleDraw method*), 162
 maxTickLength() (*qwt.scale_draw.QwtAbstractScaleDraw method*), 155
 maxValue() (*qwt.interval.QwtInterval method*), 189
 mightRender() (*qwt.text.QwtPlainTextEngine method*), 182
 mightRender() (*qwt.text.QwtRichTextEngine method*), 183
 mightRender() (*qwt.text.QwtTextEngine method*), 181
 minimumExtent() (*qwt.scale_draw.QwtAbstractScaleDraw method*), 155
 minimumSizeHint() (*qwt.plot.QwtPlot method*), 86
 minimumSizeHint() (*qwt.plot_layout.QwtPlotLayout method*), 198
 minimumSizeHint() (*qwt.scale_widget.QwtScaleWidget method*), 139
 minimumSizeHint() (*qwt.text.QwtTextLabel method*), 180
 minLabelDist() (*qwt.scale_draw.QwtScaleDraw method*), 158
 minLength() (*qwt.scale_draw.QwtScaleDraw method*), 158
 minorPen() (*qwt.plot_grid.QwtPlotGrid method*), 103
 minValue() (*qwt.interval.QwtInterval method*), 189
 mode() (*qwt.color_map.QwtLinearColorMap method*), 127
 mode() (*qwt.legend.QwtLegendData method*), 120
 module
 - qwt, 1
 - qwt.color_map, 126
 - qwt.graphic, 184
 - qwt.interval, 188
 - qwt.legend, 119
 - qwt.plot, 75
 - qwt.plot_canvas, 96
 - qwt.plot_curve, 104
 - qwt.plot_directpainter, 192
 - qwt.plot_grid, 100
 - qwt.plot_layout, 194
 - qwt.plot_marker, 113
 - qwt.plot_renderer, 128
 - qwt.plot_series, 199
 - qwt.scale_div, 141
 - qwt.scale_draw, 152
 - qwt.scale_engine, 145
 - qwt.scale_map, 132
 - qwt.scale_widget, 135
 - qwt.symbol, 163
 - qwt.text, 172
 - qwt.toqimage, 184
 - qwt.transform, 204
- mousePressEvent() (*qwt.legend.QwtLegendLabel method*), 122
 mouseReleaseEvent() (*qwt.legend.QwtLegendLabel method*), 122
 move() (*qwt.scale_draw.QwtScaleDraw method*), 159
- ## N
- normalized() (*qwt.interval.QwtInterval method*), 190

O

`orientation()` (*qwt.plot_series.QwtPlotSeriesItem* method), 200

`orientation()` (*qwt.scale_draw.QwtScaleDraw* method), 157

P

`p1()` (*qwt.scale_map.QwtScaleMap* method), 133

`p2()` (*qwt.scale_map.QwtScaleMap* method), 133

`paintEvent()` (*qwt.legend.QwtLegendLabel* method), 122

`paintEvent()` (*qwt.plot_canvas.QwtPlotCanvas* method), 99

`paintEvent()` (*qwt.scale_widget.QwtScaleWidget* method), 138

`paintEvent()` (*qwt.text.QwtTextLabel* method), 180

`paintRect()` (*qwt.plot.QwtPlotItem* method), 96

`path()` (*qwt.symbol.QwtSymbol* method), 167

`pDist()` (*qwt.scale_map.QwtScaleMap* method), 133

`pen()` (*qwt.plot_curve.QwtPlotCurve* method), 107

`pen()` (*qwt.symbol.QwtSymbol* method), 169

`penWidth()` (*qwt.scale_draw.QwtAbstractScaleDraw* method), 154

`pinPoint()` (*qwt.symbol.QwtSymbol* method), 170

`pixmap()` (*qwt.symbol.QwtSymbol* method), 167

`plainText()` (*qwt.text.QwtTextLabel* method), 178

`plot()` (*qwt.plot.QwtPlotItem* method), 90

`plot()` (*qwt.plot_canvas.QwtPlotCanvas* method), 97

`plotLayout()` (*qwt.plot.QwtPlot* method), 85

`pos()` (*qwt.scale_draw.QwtScaleDraw* method), 159

`print_()` (*qwt.plot.QwtPlot* method), 89

Q

`qwt`
 module, 1

`qwt.color_map`
 module, 126

`qwt.graphic`
 module, 184

`qwt.interval`
 module, 188

`qwt.legend`
 module, 119

`qwt.plot`
 module, 75

`qwt.plot_canvas`
 module, 96

`qwt.plot_curve`
 module, 104

`qwt.plot_directpainter`
 module, 192

`qwt.plot_grid`
 module, 100

`qwt.plot_layout`
 module, 194

`qwt.plot_marker`
 module, 113

`qwt.plot_renderer`
 module, 128

`qwt.plot_series`
 module, 199

`qwt.scale_div`
 module, 141

`qwt.scale_draw`
 module, 152

`qwt.scale_engine`
 module, 145

`qwt.scale_map`
 module, 132

`qwt.scale_widget`
 module, 135

`qwt.symbol`
 module, 163

`qwt.text`
 module, 172

`qwt.toqimage`
 module, 184

`qwt.transform`
 module, 204

`QwtAbstractScaleDraw` (class in *qwt.scale_draw*), 152

`QwtAbstractScaleDraw.QwtAbstractScaleDraw`
 (class in *qwt.scale_draw*), 152

`QwtAlphaColorMap` (class in *qwt.color_map*), 128

`QwtAlphaColorMap.QwtAlphaColorMap` (class in *qwt.color_map*), 128

`QwtColorMap` (class in *qwt.color_map*), 126

`QwtColorMap.QwtColorMap` (class in *qwt.color_map*), 126

`QwtGraphic` (class in *qwt.graphic*), 184

`QwtGraphic.QwtGraphic` (class in *qwt.graphic*), 185

`QwtInterval` (class in *qwt.interval*), 189

`QwtInterval.QwtInterval` (class in *qwt.interval*), 189

`QwtLegend` (class in *qwt.legend*), 122

`QwtLegend.checked` (in module *qwt.legend*), 123

`QwtLegend.clicked` (in module *qwt.legend*), 122

`QwtLegend.QwtLegend` (class in *qwt.legend*), 122

`QwtLegendData` (class in *qwt.legend*), 119

`QwtLegendLabel` (class in *qwt.legend*), 120

`QwtLinearColorMap` (class in *qwt.color_map*), 127

`QwtLinearColorMap.QwtLinearColorMap` (class in *qwt.color_map*), 127

`QwtLinearScaleEngine` (class in *qwt.scale_engine*), 149

`QwtLogScaleEngine` (class in *qwt.scale_engine*), 150

`QwtLogTransform` (class in *qwt.transform*), 205

`QwtNullTransform` (class in *qwt.transform*), 205

`QwtPlainTextEngine` (class in *qwt.text*), 182

- QwtPlot (class in *qwt.plot*), 75
 QwtPlot.itemAttached (in module *qwt.plot*), 76
 QwtPlot.legendDataChanged (in module *qwt.plot*), 76
 QwtPlot.QwtPlot (class in *qwt.plot*), 76
 QwtPlotCanvas (class in *qwt.plot_canvas*), 96
 QwtPlotCanvas.QwtPlotCanvas (class in *qwt.plot_canvas*), 97
 QwtPlotCurve (class in *qwt.plot_curve*), 104
 QwtPlotCurve.QwtPlotCurve (class in *qwt.plot_curve*), 105
 QwtPlotDirectPainter (class in *qwt.plot_directpainter*), 192
 QwtPlotGrid (class in *qwt.plot_grid*), 100
 QwtPlotItem (class in *qwt.plot*), 89
 QwtPlotItem.QwtPlotItem (class in *qwt.plot*), 90
 QwtPlotLayout (class in *qwt.plot_layout*), 194
 QwtPlotMarker (class in *qwt.plot_marker*), 113
 QwtPlotRenderer (class in *qwt.plot_renderer*), 128
 QwtPlotSeriesItem (class in *qwt.plot_series*), 200
 QwtPointArrayData (class in *qwt.plot_series*), 202
 QwtPointArrayData.QwtCQwtPointArrayDataolorMap (class in *qwt.plot_series*), 202
 QwtPowerTransform (class in *qwt.transform*), 206
 QwtRichTextEngine (class in *qwt.text*), 183
 QwtScaleDiv (class in *qwt.scale_div*), 142
 QwtScaleDiv.QwtScaleDiv (class in *qwt.scale_div*), 142
 QwtScaleDraw (class in *qwt.scale_draw*), 156
 QwtScaleDraw.QwtScaleDraw (class in *qwt.scale_draw*), 157
 QwtScaleEngine (class in *qwt.scale_engine*), 145
 QwtScaleMap (class in *qwt.scale_map*), 132
 QwtScaleMap.QwtScaleMap (class in *qwt.scale_map*), 132
 QwtScaleWidget (class in *qwt.scale_widget*), 135
 QwtScaleWidget.QwtScaleWidget (class in *qwt.scale_widget*), 135
 QwtSeriesData (class in *qwt.plot_series*), 201
 QwtSeriesStore (class in *qwt.plot_series*), 203
 QwtSymbol (class in *qwt.symbol*), 163
 QwtSymbol.QwtSymbol (class in *qwt.symbol*), 164
 QwtText (class in *qwt.text*), 172
 QwtText.QwtText (class in *qwt.text*), 173
 QwtTextEngine (class in *qwt.text*), 180
 QwtTextLabel (class in *qwt.text*), 178
 QwtTextLabel.QwtTextLabel (class in *qwt.text*), 178
 QwtTransform (class in *qwt.transform*), 204
 render() (*qwt.plot_renderer.QwtPlotRenderer* method), 130
 renderCanvas() (*qwt.plot_renderer.QwtPlotRenderer* method), 131
 renderDocument() (*qwt.plot_renderer.QwtPlotRenderer* method), 130
 renderFlags() (*qwt.text.QwtText* method), 174
 renderFooter() (*qwt.plot_renderer.QwtPlotRenderer* method), 131
 renderItem() (*qwt.legend.QwtLegend* method), 125
 renderLegend() (*qwt.legend.QwtLegend* method), 125
 renderLegend() (*qwt.plot_renderer.QwtPlotRenderer* method), 131
 renderScale() (*qwt.plot_renderer.QwtPlotRenderer* method), 131
 renderSymbols() (*qwt.symbol.QwtSymbol* method), 171
 renderTitle() (*qwt.plot_renderer.QwtPlotRenderer* method), 131
 renderTo() (*qwt.plot_renderer.QwtPlotRenderer* method), 130
 replot() (*qwt.plot.QwtPlot* method), 86
 replot() (*qwt.plot_canvas.QwtPlotCanvas* method), 99
 reset() (*qwt.graphic.QwtGraphic* method), 185
 reset() (*qwt.plot_directpainter.QwtPlotDirectPainter* method), 194
 resizeEvent() (*qwt.plot.QwtPlot* method), 86
 resizeEvent() (*qwt.plot_canvas.QwtPlotCanvas* method), 99
 resizeEvent() (*qwt.scale_widget.QwtScaleWidget* method), 138
 rtti() (*qwt.plot.QwtPlotItem* method), 90
 rtti() (*qwt.plot_curve.QwtPlotCurve* method), 105
 rtti() (*qwt.plot_grid.QwtPlotGrid* method), 100
 rtti() (*qwt.plot_marker.QwtPlotMarker* method), 114
S
 s1() (*qwt.scale_map.QwtScaleMap* method), 133
 s2() (*qwt.scale_map.QwtScaleMap* method), 133
 sample() (*qwt.plot_series.QwtPointArrayData* method), 202
 sample() (*qwt.plot_series.QwtSeriesData* method), 201
 sample() (*qwt.plot_series.QwtSeriesStore* method), 203
 scaleChange() (*qwt.scale_widget.QwtScaleWidget* method), 139
 scaledBoundingRect() (*qwt.graphic.QwtGraphic* method), 186
 scaleDiv() (*qwt.scale_draw.QwtAbstractScaleDraw* method), 154
 scaleDraw() (*qwt.scale_widget.QwtScaleWidget* method), 137
 scaleMap() (*qwt.scale_draw.QwtAbstractScaleDraw* method), 153
 scaleRect() (*qwt.plot.QwtPlotItem* method), 96
R
 range() (*qwt.scale_div.QwtScaleDiv* method), 144
 reference() (*qwt.scale_engine.QwtScaleEngine* method), 148
 removeItem() (*qwt.plot.QwtPlot* method), 76
 render() (*qwt.graphic.QwtGraphic* method), 186

- `scaleRect()` (*qwt.plot_layout.QwtPlotLayout* method), 198
`sDist()` (*qwt.scale_map.QwtScaleMap* method), 133
`setAlignCanvasToScales()` (*qwt.plot_layout.QwtPlotLayout* method), 195
`setAlignment()` (*qwt.scale_draw.QwtScaleDraw* method), 157
`setAlignment()` (*qwt.scale_widget.QwtScaleWidget* method), 136
`setAttribute()` (*qwt.plot_directpainter.QwtPlotDirectPainter* method), 192
`setAttribute()` (*qwt.scale_engine.QwtScaleEngine* method), 147
`setAttributes()` (*qwt.scale_engine.QwtScaleEngine* method), 148
`setAutoReplot()` (*qwt.plot.QwtPlot* method), 84
`setAxes()` (*qwt.plot.QwtPlotItem* method), 94
`setAxis()` (*qwt.plot.QwtPlotItem* method), 94
`setAxisAutoScale()` (*qwt.plot.QwtPlot* method), 81
`setAxisFont()` (*qwt.plot.QwtPlot* method), 81
`setAxisLabelAlignment()` (*qwt.plot.QwtPlot* method), 82
`setAxisLabelAutoSize()` (*qwt.plot.QwtPlot* method), 82
`setAxisLabelRotation()` (*qwt.plot.QwtPlot* method), 82
`setAxisMargin()` (*qwt.plot.QwtPlot* method), 83
`setAxisMaxMajor()` (*qwt.plot.QwtPlot* method), 83
`setAxisMaxMinor()` (*qwt.plot.QwtPlot* method), 83
`setAxisScale()` (*qwt.plot.QwtPlot* method), 81
`setAxisScaleDiv()` (*qwt.plot.QwtPlot* method), 81
`setAxisScaleDraw()` (*qwt.plot.QwtPlot* method), 82
`setAxisScaleEngine()` (*qwt.plot.QwtPlot* method), 78
`setAxisTitle()` (*qwt.plot.QwtPlot* method), 83
`setBackgroundBrush()` (*qwt.text.QwtText* method), 176
`setBase()` (*qwt.scale_engine.QwtScaleEngine* method), 148
`setBaseline()` (*qwt.plot_curve.QwtPlotCurve* method), 111
`setBorderDist()` (*qwt.scale_widget.QwtScaleWidget* method), 136
`setBorderFlags()` (*qwt.interval.QwtInterval* method), 189
`setBorderPen()` (*qwt.text.QwtText* method), 176
`setBorderRadius()` (*qwt.plot_canvas.QwtPlotCanvas* method), 98
`setBorderRadius()` (*qwt.text.QwtText* method), 175
`setBrush()` (*qwt.plot_curve.QwtPlotCurve* method), 107
`setBrush()` (*qwt.symbol.QwtSymbol* method), 169
`setCachePolicy()` (*qwt.symbol.QwtSymbol* method), 165
`setCanvas()` (*qwt.plot.QwtPlot* method), 84
`setCanvasBackground()` (*qwt.plot.QwtPlot* method), 87
`setCanvasMargin()` (*qwt.plot_layout.QwtPlotLayout* method), 194
`setCanvasRect()` (*qwt.plot_layout.QwtPlotLayout* method), 198
`setChecked()` (*qwt.legend.QwtLegendLabel* method), 121
`setClipping()` (*qwt.plot_directpainter.QwtPlotDirectPainter* method), 193
`setClipRegion()` (*qwt.plot_directpainter.QwtPlotDirectPainter* method), 193
`setColor()` (*qwt.color_map.QwtAlphaColorMap* method), 128
`setColor()` (*qwt.symbol.QwtSymbol* method), 170
`setColor()` (*qwt.text.QwtText* method), 175
`setColorBarEnabled()` (*qwt.scale_widget.QwtScaleWidget* method), 140
`setColorBarWidth()` (*qwt.scale_widget.QwtScaleWidget* method), 141
`setColorMap()` (*qwt.scale_widget.QwtScaleWidget* method), 141
`setCurveAttribute()` (*qwt.plot_curve.QwtPlotCurve* method), 110
`setData()` (*qwt.legend.QwtLegendLabel* method), 120
`setData()` (*qwt.plot_curve.QwtPlotCurve* method), 112
`setData()` (*qwt.plot_series.QwtSeriesStore* method), 203
`setDefaultItemMode()` (*qwt.legend.QwtLegend* method), 123
`setDefaultSize()` (*qwt.graphic.QwtGraphic* method), 186
`setDiscardFlag()` (*qwt.plot_renderer.QwtPlotRenderer* method), 128
`setDiscardFlags()` (*qwt.plot_renderer.QwtPlotRenderer* method), 129
`setDown()` (*qwt.legend.QwtLegendLabel* method), 122
`setFlatStyle()` (*qwt.plot.QwtPlot* method), 77
`setFocusIndicator()` (*qwt.plot_canvas.QwtPlotCanvas* method), 98
`setFont()` (*qwt.text.QwtText* method), 174
`setFooter()` (*qwt.plot.QwtPlot* method), 85
`setFooterRect()` (*qwt.plot_layout.QwtPlotLayout* method), 197
`setGraphic()` (*qwt.symbol.QwtSymbol* method), 168
`setIcon()` (*qwt.legend.QwtLegendLabel* method), 121
`setIcon()` (*qwt.plot.QwtPlotItem* method), 91
`setIndent()` (*qwt.text.QwtTextLabel* method), 179
`setInterval()` (*qwt.interval.QwtInterval* method), 189
`setInterval()` (*qwt.scale_div.QwtScaleDiv* method), 143

setItemAttribute() (*qwt.plot.QwtPlotItem* method), 91
 setItemInterest() (*qwt.plot.QwtPlotItem* method), 92
 setItemMode() (*qwt.legend.QwtLegendLabel* method), 120
 setLabel() (*qwt.plot_marker.QwtPlotMarker* method), 116
 setLabelAlignment() (*qwt.plot_marker.QwtPlotMarker* method), 116
 setLabelAlignment() (*qwt.scale_draw.QwtScaleDraw* method), 161
 setLabelAlignment() (*qwt.scale_widget.QwtScaleWidget* method), 137
 setLabelAutoSize() (*qwt.scale_draw.QwtScaleDraw* method), 162
 setLabelAutoSize() (*qwt.scale_widget.QwtScaleWidget* method), 137
 setLabelOrientation() (*qwt.plot_marker.QwtPlotMarker* method), 117
 setLabelRotation() (*qwt.scale_draw.QwtScaleDraw* method), 161
 setLabelRotation() (*qwt.scale_widget.QwtScaleWidget* method), 137
 setLayoutAttribute() (*qwt.text.QwtText* method), 177
 setLayoutFlag() (*qwt.plot_renderer.QwtPlotRenderer* method), 129
 setLayoutFlag() (*qwt.scale_widget.QwtScaleWidget* method), 135
 setLayoutFlags() (*qwt.plot_renderer.QwtPlotRenderer* method), 129
 setLegendAttribute() (*qwt.plot_curve.QwtPlotCurve* method), 106
 setLegendIconSize() (*qwt.plot.QwtPlotItem* method), 92
 setLegendPosition() (*qwt.plot_layout.QwtPlotLayout* method), 195
 setLegendRatio() (*qwt.plot_layout.QwtPlotLayout* method), 196
 setLegendRect() (*qwt.plot_layout.QwtPlotLayout* method), 197
 setLength() (*qwt.scale_draw.QwtScaleDraw* method), 159
 setLinePen() (*qwt.plot_marker.QwtPlotMarker* method), 118
 setLineStyle() (*qwt.plot_marker.QwtPlotMarker* method), 115
 setLowerBound() (*qwt.scale_div.QwtScaleDiv* method), 143
 setMajorPen() (*qwt.plot_grid.QwtPlotGrid* method), 102
 setMargin() (*qwt.scale_widget.QwtScaleWidget* method), 136
 setMargin() (*qwt.text.QwtTextLabel* method), 179
 setMargins() (*qwt.scale_engine.QwtScaleEngine* method), 146
 setMaxColumns() (*qwt.legend.QwtLegend* method), 123
 setMaxValue() (*qwt.interval.QwtInterval* method), 189
 setMinBorderDist() (*qwt.scale_widget.QwtScaleWidget* method), 140
 setMinimumExtent() (*qwt.scale_draw.QwtAbstractScaleDraw* method), 154
 setMinorPen() (*qwt.plot_grid.QwtPlotGrid* method), 102
 setMinValue() (*qwt.interval.QwtInterval* method), 189
 setMode() (*qwt.color_map.QwtLinearColorMap* method), 127
 setOrientation() (*qwt.plot_series.QwtPlotSeriesItem* method), 200
 setPaintAttribute() (*qwt.plot_canvas.QwtPlotCanvas* method), 97
 setPaintAttribute() (*qwt.text.QwtText* method), 176
 setPaintInterval() (*qwt.scale_map.QwtScaleMap* method), 134
 setPath() (*qwt.symbol.QwtSymbol* method), 165
 setPen() (*qwt.plot_curve.QwtPlotCurve* method), 107
 setPen() (*qwt.plot_grid.QwtPlotGrid* method), 101
 setPen() (*qwt.symbol.QwtSymbol* method), 169
 setPenWidth() (*qwt.scale_draw.QwtAbstractScaleDraw* method), 154
 setPinPoint() (*qwt.symbol.QwtSymbol* method), 170
 setPinPointEnabled() (*qwt.symbol.QwtSymbol* method), 170
 setPixmap() (*qwt.symbol.QwtSymbol* method), 167
 setPlainText() (*qwt.text.QwtTextLabel* method), 178
 setPlotLayout() (*qwt.plot.QwtPlot* method), 85
 setRectOfInterest() (*qwt.plot_series.QwtSeriesData* method), 201
 setRectOfInterest() (*qwt.plot_series.QwtSeriesStore* method), 203
 setReference() (*qwt.scale_engine.QwtScaleEngine* method), 148
 setRenderFlags() (*qwt.text.QwtText* method), 174
 setRenderHint() (*qwt.graphic.QwtGraphic* method), 185
 setRenderHint() (*qwt.plot.QwtPlotItem* method), 92
 setSamples() (*qwt.plot_curve.QwtPlotCurve* method), 113
 setScaleDiv() (*qwt.scale_draw.QwtAbstractScaleDraw* method), 153
 setScaleDiv() (*qwt.scale_widget.QwtScaleWidget* method), 140

- setScaleDraw() (*qwt.scale_widget.QwtScaleWidget method*), 137
 setScaleInterval() (*qwt.scale_map.QwtScaleMap method*), 134
 setScaleRect() (*qwt.plot_layout.QwtPlotLayout method*), 197
 setSize() (*qwt.symbol.QwtSymbol method*), 168
 setSpacing() (*qwt.legend.QwtLegendLabel method*), 121
 setSpacing() (*qwt.plot_layout.QwtPlotLayout method*), 195
 setSpacing() (*qwt.plot_marker.QwtPlotMarker method*), 117
 setSpacing() (*qwt.scale_draw.QwtAbstractScaleDraw method*), 154
 setSpacing() (*qwt.scale_widget.QwtScaleWidget method*), 136
 setStyle() (*qwt.plot_curve.QwtPlotCurve method*), 106
 setStyle() (*qwt.symbol.QwtSymbol method*), 171
 setSvgDocument() (*qwt.symbol.QwtSymbol method*), 168
 setSymbol() (*qwt.plot_curve.QwtPlotCurve method*), 107
 setSymbol() (*qwt.plot_marker.QwtPlotMarker method*), 116
 setText() (*qwt.legend.QwtLegendLabel method*), 120
 setText() (*qwt.text.QwtText method*), 173
 setText() (*qwt.text.QwtTextLabel method*), 179
 setTextEngine() (*qwt.text.QwtText method*), 178
 setTickLength() (*qwt.scale_draw.QwtAbstractScaleDraw method*), 155
 setTickLighterFactor() (*qwt.scale_draw.QwtAbstractScaleDraw method*), 155
 setTicks() (*qwt.scale_div.QwtScaleDiv method*), 144
 setTitle() (*qwt.plot.QwtPlot method*), 84
 setTitle() (*qwt.plot.QwtPlotItem method*), 91
 setTitle() (*qwt.scale_widget.QwtScaleWidget method*), 136
 setTitleRect() (*qwt.plot_layout.QwtPlotLayout method*), 196
 setTransformation() (*qwt.scale_draw.QwtAbstractScaleDraw method*), 153
 setTransformation() (*qwt.scale_engine.QwtScaleEngine method*), 146
 setTransformation() (*qwt.scale_map.QwtScaleMap method*), 134
 setTransformation() (*qwt.scale_widget.QwtScaleWidget method*), 140
 setUpperBound() (*qwt.scale_div.QwtScaleDiv method*), 143
 setValue() (*qwt.legend.QwtLegendData method*), 119
 setValue() (*qwt.plot_marker.QwtPlotMarker method*), 114
 setValues() (*qwt.legend.QwtLegendData method*), 119
 setVisible() (*qwt.plot.QwtPlotItem method*), 93
 setXAxis() (*qwt.plot.QwtPlotItem method*), 94
 setXDiv() (*qwt.plot_grid.QwtPlotGrid method*), 101
 setXValue() (*qwt.plot_marker.QwtPlotMarker method*), 115
 setYAxis() (*qwt.plot.QwtPlotItem method*), 94
 setYDiv() (*qwt.plot_grid.QwtPlotGrid method*), 101
 setYValue() (*qwt.plot_marker.QwtPlotMarker method*), 115
 setZ() (*qwt.plot.QwtPlotItem method*), 91
 show() (*qwt.plot.QwtPlotItem method*), 93
 size() (*qwt.plot_series.QwtPointArrayData method*), 202
 size() (*qwt.plot_series.QwtSeriesData method*), 201
 size() (*qwt.symbol.QwtSymbol method*), 169
 sizeHint() (*qwt.legend.QwtLegend method*), 125
 sizeHint() (*qwt.legend.QwtLegendLabel method*), 122
 sizeHint() (*qwt.plot.QwtPlot method*), 86
 sizeHint() (*qwt.scale_widget.QwtScaleWidget method*), 139
 sizeHint() (*qwt.text.QwtTextLabel method*), 179
 sizeMetrics() (*qwt.graphic.QwtGraphic method*), 186
 spacing() (*qwt.legend.QwtLegendLabel method*), 121
 spacing() (*qwt.plot_layout.QwtPlotLayout method*), 195
 spacing() (*qwt.plot_marker.QwtPlotMarker method*), 117
 spacing() (*qwt.scale_draw.QwtAbstractScaleDraw method*), 154
 spacing() (*qwt.scale_widget.QwtScaleWidget method*), 138
 startBorderDist() (*qwt.scale_widget.QwtScaleWidget method*), 137
 strip() (*qwt.scale_engine.QwtScaleEngine method*), 147
 Style (*qwt.symbol.QwtSymbol attribute*), 165
 style() (*qwt.plot_curve.QwtPlotCurve method*), 106
 style() (*qwt.symbol.QwtSymbol method*), 171
 swapData() (*qwt.plot_series.QwtSeriesStore method*), 203
 symbol() (*qwt.plot_curve.QwtPlotCurve method*), 107
 symbol() (*qwt.plot_marker.QwtPlotMarker method*), 116
 symmetrize() (*qwt.interval.QwtInterval method*), 191

T

- testAttribute() (*qwt.plot_directpainter.QwtPlotDirectPainter method*), 192

- testAttribute() (*qwt.scale_engine.QwtScaleEngine method*), 148
- testCurveAttribute() (*qwt.plot_curve.QwtPlotCurve method*), 110
- testDiscardFlag() (*qwt.plot_renderer.QwtPlotRenderer method*), 129
- testItemAttribute() (*qwt.plot.QwtPlotItem method*), 91
- testItemInterest() (*qwt.plot.QwtPlotItem method*), 92
- testLayoutAttribute() (*qwt.text.QwtText method*), 177
- testLayoutFlag() (*qwt.plot_renderer.QwtPlotRenderer method*), 129
- testLayoutFlag() (*qwt.scale_widget.QwtScaleWidget method*), 135
- testLegendAttribute() (*qwt.plot_curve.QwtPlotCurve method*), 106
- testPaintAttribute() (*qwt.plot_canvas.QwtPlotCanvas method*), 98
- testPaintAttribute() (*qwt.text.QwtText method*), 176
- testRenderHint() (*qwt.graphic.QwtGraphic method*), 185
- testRenderHint() (*qwt.plot.QwtPlotItem method*), 92
- text() (*qwt.text.QwtText method*), 174
- text() (*qwt.text.QwtTextLabel method*), 179
- textEngine() (*qwt.text.QwtText method*), 177
- textMargins() (*qwt.text.QwtPlainTextEngine method*), 182
- textMargins() (*qwt.text.QwtRichTextEngine method*), 184
- textMargins() (*qwt.text.QwtTextEngine method*), 181
- textRect() (*qwt.text.QwtTextLabel method*), 180
- textSize() (*qwt.text.QwtPlainTextEngine method*), 182
- textSize() (*qwt.text.QwtRichTextEngine method*), 183
- textSize() (*qwt.text.QwtText method*), 177
- textSize() (*qwt.text.QwtTextEngine method*), 181
- tickLabel() (*qwt.scale_draw.QwtAbstractScaleDraw method*), 156
- tickLength() (*qwt.scale_draw.QwtAbstractScaleDraw method*), 155
- tickLighterFactor() (*qwt.scale_draw.QwtAbstractScaleDraw method*), 156
- ticks() (*qwt.scale_div.QwtScaleDiv method*), 144
- title() (*qwt.legend.QwtLegendData method*), 120
- title() (*qwt.plot.QwtPlot method*), 85
- title() (*qwt.plot.QwtPlotItem method*), 91
- title() (*qwt.scale_widget.QwtScaleWidget method*), 137
- titleHeightForWidth() (*qwt.scale_widget.QwtScaleWidget method*), 139
- titleLabel() (*qwt.plot.QwtPlot method*), 85
- titleRect() (*qwt.plot_layout.QwtPlotLayout method*), 197
- toImage() (*qwt.graphic.QwtGraphic method*), 187
- toPixmap() (*qwt.graphic.QwtGraphic method*), 187
- transform() (*qwt.plot.QwtPlot method*), 80
- transform() (*qwt.scale_map.QwtScaleMap method*), 134
- transform() (*qwt.transform.QwtLogTransform method*), 206
- transform() (*qwt.transform.QwtNullTransform method*), 205
- transform() (*qwt.transform.QwtPowerTransform method*), 206
- transform() (*qwt.transform.QwtTransform method*), 204
- transform_scalar() (*qwt.scale_map.QwtScaleMap method*), 133
- transformation() (*qwt.scale_engine.QwtScaleEngine method*), 146
- transformation() (*qwt.scale_map.QwtScaleMap method*), 134
- ## U
- unite() (*qwt.interval.QwtInterval method*), 190
- updateAxes() (*qwt.plot.QwtPlot method*), 83
- updateCanvasMargins() (*qwt.plot.QwtPlot method*), 86
- updateLayout() (*qwt.plot.QwtPlot method*), 86
- updateLegend() (*qwt.legend.QwtLegend method*), 124
- updateLegend() (*qwt.plot.QwtPlot method*), 88
- updateLegend() (*qwt.plot.QwtPlotItem method*), 95
- updateLegendItems() (*qwt.plot.QwtPlot method*), 89
- updateScaleDiv() (*qwt.plot_grid.QwtPlotGrid method*), 103
- updateState() (*qwt.graphic.QwtGraphic method*), 188
- updateStyleSheetInfo() (*qwt.plot_canvas.QwtPlotCanvas method*), 99
- updateWidget() (*qwt.legend.QwtLegend method*), 124
- upperBound() (*qwt.scale_div.QwtScaleDiv method*), 143
- upperMargin() (*qwt.scale_engine.QwtScaleEngine method*), 146
- usedColor() (*qwt.text.QwtText method*), 175
- usedFont() (*qwt.text.QwtText method*), 174
- ## V
- value() (*qwt.legend.QwtLegendData method*), 119
- value() (*qwt.plot_marker.QwtPlotMarker method*), 114
- values() (*qwt.legend.QwtLegendData method*), 119

`verticalScrollBar()` (*qwt.legend.QwtLegend*
method), 124

W

`width()` (*qwt.interval.QwtInterval method*), 190

X

`xAxis()` (*qwt.plot.QwtPlotItem method*), 94

`xData()` (*qwt.plot_series.QwtPointArrayData method*),
202

`xEnabled()` (*qwt.plot_grid.QwtPlotGrid method*), 103

`xMinEnabled()` (*qwt.plot_grid.QwtPlotGrid method*),
103

`xScaleDiv()` (*qwt.plot_grid.QwtPlotGrid method*), 103

`xValue()` (*qwt.plot_marker.QwtPlotMarker method*),
114

Y

`yAxis()` (*qwt.plot.QwtPlotItem method*), 94

`yData()` (*qwt.plot_series.QwtPointArrayData method*),
202

`yEnabled()` (*qwt.plot_grid.QwtPlotGrid method*), 103

`yMinEnabled()` (*qwt.plot_grid.QwtPlotGrid method*),
103

`yScaleDiv()` (*qwt.plot_grid.QwtPlotGrid method*), 103

`yValue()` (*qwt.plot_marker.QwtPlotMarker method*),
114

Z

`z()` (*qwt.plot.QwtPlotItem method*), 90