Qwt Polar User's Guide

1.1.1

Generated by Doxygen 1.8.5

Fri Sep 19 2014 12:28:25

Contents

1 QwtPolar - A Qwt/Qt Polar Plot Library

The QwtPolar library contains classes for displaying values on a polar coordinate system.

1.1 License

QwtPolar is distributed under the terms of the Qwt License, Version 1.0.

1.2 Platforms

QwtPolar depends on the Qt and Qwt frameworks and might be usable in all environments supported by Qt. It is compatible with Qt >= 4.4 and Qwt >= 6.1.

1.3 Downloads

Stable releases, prereleases and snapshots are available at the QwtPolar project page.

For getting a snapshot with all bugfixes for the latest 1.1 release:

```
svn checkout svn://svn.code.sf.net/p/qwtpolar/code/branches/qwtpolar-1.1
```

For getting a development snapshot from the SVN repository:

```
svn checkout svn://svn.code.sf.net/p/qwtpolar/code/trunk/qwtpolar
```

1.4 Support

- Mailing list

  QwtPolar doesn't have its own mailing list, but you can ask on the Qwt mailing list.

  If you prefer newsgroups use the mail to news gateway of Gmane.

- Forum

  Qt Centre is a great resource for Qt related questions. It has a sub forum, that is dedicated to Qwt related questions.

- Individual support

  If you are looking for individual support, or need someone who implements your Qwt component/application contact support@qwt-project.org. Sending requests to this address without a good reason for not using public support channels might be silently ignored.

1.5 Related Projects

Qwt, Qt Widgets for Technical Applications.

QwtPlot3D, an OpenGL 3D plot widget.

1.6 Donations

Sourceforge offers a Donation System via PayPal. You can use it, if you like to support the development of Qwt.

1.7 Credits:

Authors:

Uwe Rathmann

Project admin:

Uwe Rathmann <rathmann@users.sourceforge.net>

2 Qwt License, Version 1.0

```
  0. This License Agreement applies to any software library or other
program which contains a notice placed by the copyright holder or
other authorized party saying it may be distributed under the terms of
this Lesser General Public License (also called "this License").
Each licensee is addressed as "you".

  A "library" means a collection of software functions and/or data
prepared so as to be conveniently linked with application programs
(which use some of those functions and data) to form executables.

  The "Library", below, refers to any such software library or work
which has been distributed under these terms.  A "work based on the
Library" means either the Library or any derivative work under
copyright law: that is to say, a work containing the Library or a
portion of it, either verbatim or with modifications and/or translated
straightforwardly into another language.  (Hereinafter, translation is
included without limitation in the term "modification".)

  "Source code" for a work means the preferred form of the work for
making modifications to it.  For a library, complete source code means
all the source code for all modules it contains, plus any associated
interface definition files, plus the scripts used to control compilation
and installation of the library.

  Activities other than copying, distribution and modification are not
covered by this License; they are outside its scope.  The act of
running a program using the Library is not restricted, and output from
such a program is covered only if its contents constitute a work based
on the Library (independent of the use of the Library in a tool for
writing it).  Whether that is true depends on what the Library does
and what the program that uses the Library does.

  1. You may copy and distribute verbatim copies of the Library's
complete source code as you receive it, in any medium, provided that
you conspicuously and appropriately publish on each copy an
appropriate copyright notice and disclaimer of warranty; keep intact
all the notices that refer to this License and to the absence of any
warranty; and distribute a copy of this License along with the
Library.

  You may charge a fee for the physical act of transferring a copy,
and you may at your option offer warranty protection in exchange for a
fee.

  2. You may modify your copy or copies of the Library or any portion
of it, thus forming a work based on the Library, and copy and
distribute such modifications or work under the terms of Section 1
above, provided that you also meet all of these conditions:

    a) The modified work must itself be a software library.

    b) You must cause the files modified to carry prominent notices
    stating that you changed the files and the date of any change.

    c) You must cause the whole of the work to be licensed at no
    charge to all third parties under the terms of this License.

    d) If a facility in the modified Library refers to a function or a
    table of data to be supplied by an application program that uses
    the facility, other than as an argument passed when the facility
    is invoked, then you must make a good faith effort to ensure that,
    in the event an application does not supply such function or
    table, the facility still operates, and performs whatever part of
    its purpose remains meaningful.

    (For example, a function in a library to compute square roots has
    a purpose that is entirely well-defined independent of the
    application.  Therefore, Subsection 2d requires that any
    application-supplied function or table used by this function must
    be optional: if the application does not supply it, the square
    root function must still compute square roots.)

These requirements apply to the modified work as a whole.  If
identifiable sections of that work are not derived from the Library,
and can be reasonably considered independent and separate works in
themselves, then this License, and its terms, do not apply to those
sections when you distribute them as separate works.  But when you
distribute the same sections as part of a whole which is a work based
on the Library, the distribution of the whole must be on the terms of
this License, whose permissions for other licensees extend to the
entire whole, and thus to each and every part regardless of who wrote
it.

Thus, it is not the intent of this section to claim rights or contest
your rights to work written entirely by you; rather, the intent is to
exercise the right to control the distribution of derivative or
collective works based on the Library.
```

user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot

3 Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

4 Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

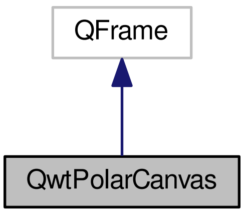
5 Class Documentation

5.1 QwtPolarCanvas Class Reference

Canvas of a QwtPolarPlot.

```
#include <qwt_polar_canvas.h>
```

Inheritance diagram for QwtPolarCanvas:



Public Types

- enum PaintAttribute { BackingStore = 0x01 }

  Paint attributes.
- typedef QFlags< PaintAttribute > PaintAttributes

  Paint attributes.

Public Member Functions

- QwtPolarCanvas (QwtPolarPlot ∗)

  Constructor.
- virtual ∼QwtPolarCanvas ()

  Destructor.
- QwtPolarPlot ∗ plot ()
- const QwtPolarPlot ∗ plot () const
- void setPaintAttribute (PaintAttribute, bool on=true)

  Changing the paint attributes.
- bool testPaintAttribute (PaintAttribute) const
- const QPixmap ∗ backingStore () const
- void invalidateBackingStore ()

  Invalidate the internal backing store.
- QwtPointPolar invTransform (const QPoint &) const
- QPoint transform (const QwtPointPolar &) const

Protected Member Functions

- virtual void paintEvent (QPaintEvent ∗)
- virtual void resizeEvent (QResizeEvent ∗)

5.1.1 Detailed Description

Canvas of a QwtPolarPlot.

The canvas is the widget, where all polar items are painted to.

Note

In opposite to QwtPlot all axes are painted on the canvas.

See Also

QwtPolarPlot

5.1.2 Member Enumeration Documentation

5.1.2.1 enum QwtPolarCanvas::PaintAttribute

Paint attributes.

The default setting enables BackingStore

See Also

setPaintAttribute(), testPaintAttribute(), backingStore()

Enumerator

BackingStore Paint double buffered and reuse the content of the pixmap buffer for some spontaneous repaints that happen when a plot gets unhidden, deiconified or changes the focus.

5.1.3 Member Function Documentation

5.1.3.1 const QPixmap ∗ QwtPolarCanvas::backingStore ( ) const

Returns

Backing store, might be null

5.1.3.2 QwtPointPolar QwtPolarCanvas::invTransform (const QPoint & *pos*) const

Translate a point from widget into plot coordinates

Parameters

pos	Point in widget coordinates of the plot canvas

Returns

Point in plot coordinates

See Also

transform()

5.1.3.3 void QwtPolarCanvas::paintEvent (QPaintEvent $*$ *event*) `[protected]`, `[virtual]`

Paint event

Parameters

event	Paint event

5.1.3.4 QwtPolarPlot ∗ QwtPolarCanvas::plot ( )

Returns

Parent plot widget

5.1.3.5 const QwtPolarPlot ∗ QwtPolarCanvas::plot ( ) const

Returns

Parent plot widget

5.1.3.6 void QwtPolarCanvas::resizeEvent (QResizeEvent ∗ *event*) `[protected]`, `[virtual]`

Resize event

Parameters

event	Resize event

5.1.3.7 void QwtPolarCanvas::setPaintAttribute (PaintAttribute *attribute*, bool *on* = `true`)

Changing the paint attributes.

Parameters

attribute	Paint attribute
on	On/Off

The default setting enables BackingStore

See Also

testPaintAttribute(), paintCache()

5.1.3.8 bool QwtPolarCanvas::testPaintAttribute (PaintAttribute *attribute*) const

Test wether a paint attribute is enabled

Parameters

attribute	Paint attribute

Returns

true if the attribute is enabled

See Also

setPaintAttribute()

5.1.3.9 QPoint QwtPolarCanvas::transform (const QwtPointPolar & *polarPos*) const

Translate a point from plot into widget coordinates

Parameters

polarPos	Point in plot coordinates

Returns

Point in widget coordinates

See Also

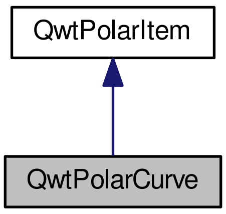
transform()

5.2 QwtPolarCurve Class Reference

An item, that represents a series of points.

`#include <qwt_polar_curve.h>`

Inheritance diagram for QwtPolarCurve:



Public Types

- enum CurveStyle { NoCurve, Lines, UserCurve = 100 }
- enum LegendAttribute { LegendShowLine = 0x01, LegendShowSymbol = 0x02 }

  Attributes how to represent the curve on the legend.
- typedef QFlags< LegendAttribute > LegendAttributes

  Legend attributes.

Public Member Functions

- QwtPolarCurve ()

  Constructor.
- QwtPolarCurve (const QwtText &title)
- QwtPolarCurve (const QString &title)
- virtual ∼QwtPolarCurve ()

  Destructor.
- virtual int rtti () const
- void setLegendAttribute (LegendAttribute, bool on=true)
- bool testLegendAttribute (LegendAttribute) const

  Test if a lefend attribute is enables.

- void setData (QwtSeriesData< QwtPointPolar > ∗data)
- const QwtSeriesData
  < QwtPointPolar > ∗ data () const
- size_t dataSize () const
- QwtPointPolar sample (int i) const
- void setPen (const QPen &)

  Assign a pen.
- const QPen & pen () const
- void setStyle (CurveStyle style)
- CurveStyle style () const
- void setSymbol (QwtSymbol ∗)

  Assign a symbol.
- const QwtSymbol ∗ symbol () const
- void setCurveFitter (QwtCurveFitter ∗)

  Insert a curve fitter.
- QwtCurveFitter ∗ curveFitter () const
- virtual void draw (QPainter ∗p, const QwtScaleMap &azimuthMap, const QwtScaleMap &radialMap, const QPointF &pole, double radius, const QRectF &canvasRect) const
- virtual void draw (QPainter ∗p, const QwtScaleMap &azimuthMap, const QwtScaleMap &radialMap, const QPointF &pole, int from, int to) const

  Draw an interval of the curve.
- virtual QwtInterval boundingInterval (int scaleId) const
- virtual QwtGraphic legendIcon (int index, const QSizeF &) const

Protected Member Functions

- void init ()

  Initialize data members.
- virtual void drawCurve (QPainter ∗, int style, const QwtScaleMap &azimuthMap, const QwtScaleMap &radial-Map, const QPointF &pole, int from, int to) const
- virtual void drawSymbols (QPainter ∗, const QwtSymbol &, const QwtScaleMap &azimuthMap, const Qwt-ScaleMap &radialMap, const QPointF &pole, int from, int to) const
- void drawLines (QPainter ∗, const QwtScaleMap &azimuthMap, const QwtScaleMap &radialMap, const Q-PointF &pole, int from, int to) const

5.2.1 Detailed Description

An item, that represents a series of points.

A curve is the representation of a series of points in polar coordinates. The points are connected to the curve using the abstract QwtData interface.

See Also

QwtPolarPlot, QwtSymbol, QwtScaleMap

5.2.2 Member Enumeration Documentation

5.2.2.1 enum **QwtPolarCurve::CurveStyle**

Curve styles.

See Also

setStyle(), style()

Enumerator

NoCurve Don't draw a curve. Note: This doesn't affect the symbols.

Lines Connect the points with straight lines. The lines might be interpolated depending on the 'Fitted' attribute. Curve fitting can be configured using setCurveFitter().

UserCurve Values $>$ 100 are reserved for user specific curve styles.

5.2.2.2 enum QwtPolarCurve::LegendAttribute

Attributes how to represent the curve on the legend.

If none of the flags is activated QwtPlotCurve tries to find a color representing the curve and paints a rectangle with it. In the default setting all attributes are off.

See Also

setLegendAttribute(), testLegendAttribute()

Enumerator

LegendShowLine If the curveStyle() is not NoCurve a line is painted with the curvePen().

LegendShowSymbol If the curve has a valid symbol it is painted.

5.2.3 Constructor & Destructor Documentation

5.2.3.1 QwtPolarCurve::QwtPolarCurve (const QwtText & *title*) [explicit]

Constructor

Parameters

title	title of the curve

5.2.3.2 QwtPolarCurve::QwtPolarCurve (const QString & *title*) [explicit]

Constructor

Parameters

title	title of the curve

5.2.4 Member Function Documentation

5.2.4.1 QwtInterval QwtPolarCurve::boundingInterval (int *scaleId*) const [virtual]

Interval, that is necessary to display the item This interval can be useful for operations like clipping or autoscaling

Parameters

scaleId	Scale index

Returns

bounding interval

See Also

QwtData::boundingRect()

Reimplemented from QwtPolarItem.

5.2.4.2 QwtCurveFitter $*$ QwtPolarCurve::curveFitter ( ) const

Returns

The curve fitter

See Also

setCurveFitter()

5.2.4.3 const QwtSeriesData$<$ QwtPointPolar $>$ $*$ QwtPolarCurve::data ( ) const `[inline]`

Returns

the the curve data

5.2.4.4 size_t QwtPolarCurve::dataSize ( ) const

Returns

Number of points

See Also

setData()

5.2.4.5 void QwtPolarCurve::draw (QPainter $*$ *painter,* const QwtScaleMap & *azimuthMap,* const QwtScaleMap & *radialMap,* const QPointF & *pole,* double *radius,* const QRectF & *canvasRect*) const `[virtual]`

Draw the curve

Parameters

painter	Painter
azimuthMap	Maps azimuth values to values related to 0.0, M_2PI
radialMap	Maps radius values into painter coordinates.
pole	Position of the pole in painter coordinates
radius	Radius of the complete plot area in painter coordinates
canvasRect	Contents rect of the canvas in painter coordinates

Implements QwtPolarItem.

5.2.4.6 void QwtPolarCurve::draw (QPainter $*$ *painter,* const QwtScaleMap & *azimuthMap,* const QwtScaleMap & *radialMap,* const QPointF & *pole,* int *from,* int *to*) const `[virtual]`

Draw an interval of the curve.

Parameters

painter	Painter
azimuthMap	Maps azimuth values to values related to 0.0, M_2PI

radialMap	Maps radius values into painter coordinates.
pole	Position of the pole in painter coordinates
from	index of the first point to be painted
to	index of the last point to be painted. If to $<$ 0 the curve will be painted to its last point.

See Also

drawCurve(), drawSymbols(),

5.2.4.7 void QwtPolarCurve::drawCurve (QPainter $*$ *painter,* int *style,* const QwtScaleMap & *azimuthMap,* const QwtScaleMap & *radialMap,* const QPointF & *pole,* int *from,* int *to*) const [protected], [virtual]

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

Parameters

painter	Painter
style	Curve style, see QwtPolarCurve::CurveStyle
azimuthMap	Maps azimuth values to values related to 0.0, M_2PI
radialMap	Maps radius values into painter coordinates.
pole	Position of the pole in painter coordinates
from	index of the first point to be painted
to	index of the last point to be painted.

See Also

draw(), drawLines()

5.2.4.8 void QwtPolarCurve::drawLines (QPainter $*$ *painter,* const QwtScaleMap & *azimuthMap,* const QwtScaleMap & *radialMap,* const QPointF & *pole,* int *from,* int *to*) const [protected]

Draw lines

Parameters

painter	Painter
azimuthMap	Maps azimuth values to values related to 0.0, M_2PI
radialMap	Maps radius values into painter coordinates.
pole	Position of the pole in painter coordinates
from	index of the first point to be painted
to	index of the last point to be painted.

See Also

draw(), drawLines(), setCurveFitter()

5.2.4.9 void QwtPolarCurve::drawSymbols (QPainter $*$ *painter,* const QwtSymbol & *symbol,* const QwtScaleMap & *azimuthMap,* const QwtScaleMap & *radialMap,* const QPointF & *pole,* int *from,* int *to*) const [protected], [virtual]

Draw symbols

Parameters

painter	Painter

symbol	Curve symbol
azimuthMap	Maps azimuth values to values related to 0.0, M_2PI
radialMap	Maps radius values into painter coordinates.
pole	Position of the pole in painter coordinates
from	index of the first point to be painted
to	index of the last point to be painted.

See Also

setSymbol(), draw(), drawCurve()

5.2.4.10 QwtGraphic QwtPolarCurve::legendIcon (int *index*, const QSizeF & *size*) const [virtual]

Returns

Icon representing the curve on the legend

Parameters

index	Index of the legend entry (ignored as there is only one)
size	Icon size

See Also

QwtPolarItem::setLegendIconSize(), QwtPolarItem::legendData()

Reimplemented from QwtPolarItem.

5.2.4.11 const QPen & QwtPolarCurve::pen ( ) const

Returns

Pen used to draw the lines

See Also

setPen()

5.2.4.12 int QwtPolarCurve::rtti ( ) const [virtual]

Returns

QwtPolarCurve::Rtti_PolarCurve

Reimplemented from QwtPolarItem.

5.2.4.13 QwtPointPolar QwtPolarCurve::sample (int *i*) const [inline]

Parameters

i	index

Returns

point at position i

5.2.4.14 void QwtPolarCurve::setCurveFitter (QwtCurveFitter ∗ *curveFitter*)

Insert a curve fitter.

Parameters

curveFitter	Curve fitter

A curve fitter interpolates the curve points. F.e QwtPolarFitter adds equidistant points so that the connection gets rounded instead of having straight lines. If curveFitter is NULL fitting is disabled.

See Also

curveFitter()

5.2.4.15 void QwtPolarCurve::setData (QwtSeriesData< QwtPointPolar > ∗ *data*)

Initialize data with a pointer to QwtSeriesData<QwtPointPolar>.

The x-values of the data object represent the azimuth, the y-value respresent the radius.

Parameters

data	Data

5.2.4.16 void QwtPolarCurve::setLegendAttribute (LegendAttribute *attribute*, bool *on* = `true`)

Specify an attribute how to draw the legend identifier

Parameters

attribute	Attribute
on	On/Off /sa LegendAttribute, testLegendAttribute()

5.2.4.17 void QwtPolarCurve::setPen (const QPen & *pen*)

Assign a pen.

Parameters

pen	New pen

See Also

pen()

5.2.4.18 void QwtPolarCurve::setStyle (CurveStyle *style*)

Set the curve's drawing style

Parameters

style	Curve style

See Also

CurveStyle, style()

5.2.4.19 void QwtPolarCurve::setSymbol (QwtSymbol ∗ *symbol*)

Assign a symbol.

Parameters

symbol	Symbol

See Also

symbol()

5.2.4.20 QwtPolarCurve::CurveStyle QwtPolarCurve::style ( ) const

Returns

Current style

See Also

CurveStyle, setStyle()

5.2.4.21 const QwtSymbol ∗ QwtPolarCurve::symbol ( ) const

Returns

The current symbol

See Also

setSymbol()

5.2.4.22 bool QwtPolarCurve::testLegendAttribute (LegendAttribute *attribute*) const

Test if a lefend attribute is enables.

Parameters

attribute	Legend attribute

Returns

True if attribute is enabled

See Also

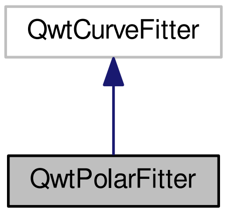
LegendAttribute, setLegendAttribute()

5.3 QwtPolarFitter Class Reference

A simple curve fitter for polar points.

```
#include <qwt_polar_fitter.h>
```

Inheritance diagram for QwtPolarFitter:



Public Member Functions

- QwtPolarFitter (int stepCount=5)
- virtual ∼QwtPolarFitter ()

  Destructor.
- void setStepCount (int size)
- int stepCount () const
- virtual QPolygonF fitCurve (const QPolygonF &) const

5.3.1 Detailed Description

A simple curve fitter for polar points.

QwtPolarFitter adds equidistant points between 2 curve points, so that the connection gets rounded according to the nature of a polar plot.

See Also

QwtPolarCurve::setCurveFitter()

5.3.2 Constructor & Destructor Documentation

5.3.2.1 QwtPolarFitter::QwtPolarFitter (int *stepCount* = 5)

Constructor

Parameters

stepCount	Number of points, that will be inserted between 2 points

See Also

setStepCount()

5.3.3 Member Function Documentation

5.3.3.1 QPolygonF QwtPolarFitter::fitCurve (const QPolygonF & *points*) const [virtual]

Insert stepCount() number of additional points between 2 elements of points.

Parameters

points	Array of points

Returns

Array of points including the additional points

5.3.3.2 void QwtPolarFitter::setStepCount (int *stepCount*)

Assign the number of points, that will be inserted between 2 points The default value is 5.

Parameters

stepCount	Number of steps

See Also

stepCount()

5.3.3.3 int QwtPolarFitter::stepCount ( ) const

Returns

Number of points, that will be inserted between 2 points

See Also

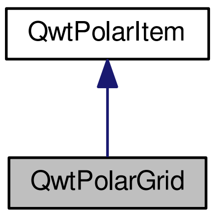
setStepCount()

5.4 QwtPolarGrid Class Reference

An item which draws scales and grid lines on a polar plot.

`#include <qwt_polar_grid.h>`

Inheritance diagram for QwtPolarGrid:



Public Types

- enum DisplayFlag {
  SmartOriginLabel = 1, HideMaxRadiusLabel = 2, ClipAxisBackground = 4, SmartScaleDraw = 8,
  ClipGridLines = 16 }

- enum GridAttribute { AutoScaling = 0x01 }

  Grid attributes.
- typedef QFlags< DisplayFlag > DisplayFlags

  Display flags.
- typedef QFlags< GridAttribute > GridAttributes

  Grid attributes.

Public Member Functions

- QwtPolarGrid ()

  Constructor.
- virtual ∼QwtPolarGrid ()

  Destructor.
- virtual int rtti () const
- void setDisplayFlag (DisplayFlag, bool on=true)
- bool testDisplayFlag (DisplayFlag) const
- void setGridAttribute (GridAttribute, bool on=true)

  Specify an attribute for the grid.
- bool testGridAttribute (GridAttribute) const
- void showGrid (int scaleId, bool show=true)
- bool isGridVisible (int scaleId) const
- void showMinorGrid (int scaleId, bool show=true)
- bool isMinorGridVisible (int scaleId) const
- void showAxis (int axisId, bool show=true)
- bool isAxisVisible (int axisId) const
- void setPen (const QPen &p)
- void setFont (const QFont &)
- void setMajorGridPen (const QPen &p)
- void setMajorGridPen (int scaleId, const QPen &p)
- QPen majorGridPen (int scaleId) const
- void setMinorGridPen (const QPen &p)
- void setMinorGridPen (int scaleId, const QPen &p)
- QPen minorGridPen (int scaleId) const
- void setAxisPen (int axisId, const QPen &p)
- QPen axisPen (int axisId) const
- void setAxisFont (int axisId, const QFont &p)
- QFont axisFont (int axisId) const
- void setScaleDraw (int axisId, QwtScaleDraw ∗)

  Set a scale draw.
- const QwtScaleDraw ∗ scaleDraw (int axisId) const
- QwtScaleDraw ∗ scaleDraw (int axisId)
- void setAzimuthScaleDraw (QwtRoundScaleDraw ∗)

  Set a scale draw for the azimuth scale.
- const QwtRoundScaleDraw ∗ azimuthScaleDraw () const
- QwtRoundScaleDraw ∗ azimuthScaleDraw ()
- virtual void draw (QPainter ∗p, const QwtScaleMap &azimuthMap, const QwtScaleMap &radialMap, const QPointF &pole, double radius, const QRectF &rect) const
- virtual void updateScaleDiv (const QwtScaleDiv &azimuthMap, const QwtScaleDiv &radialMap, const Qwt-Interval &)

  Update the item to changes of the axes scale division.
- virtual int marginHint () const

Protected Member Functions

- void drawRays (QPainter ∗, const QRectF &, const QPointF &pole, double radius, const QwtScaleMap &azimuthMap, const QList< double > &) const
- void drawCircles (QPainter ∗, const QRectF &, const QPointF &pole, const QwtScaleMap &radialMap, const QList< double > &) const
- void drawAxis (QPainter ∗, int axisId) const

5.4.1 Detailed Description

An item which draws scales and grid lines on a polar plot.

The QwtPolarGrid class can be used to draw a coordinate grid. A coordinate grid consists of major and minor gridlines. The locations of the gridlines are determined by the azimuth and radial scale divisions.

QwtPolarGrid is also responsible for drawing the axis representing the scales. It is possible to display 4 radial and one azimuth axis.

Whenever the scale divisions of the plot widget changes the grid is synchronized by updateScaleDiv().

See Also

QwtPolarPlot, QwtPolar::Axis

5.4.2 Member Enumeration Documentation

5.4.2.1 enum QwtPolarGrid::DisplayFlag

Mysterious flags trying to avoid conflicts, when painting the scales and grid lines.

The default setting enables all flags.

See Also

setDisplayFlag(), testDisplayFlag()

Enumerator

SmartOriginLabel Try to avoid situations, where the label of the origin is painted over another axis.

HideMaxRadiusLabel Often the outermost tick of the radial scale is close to the canvas border. With HideMaxRadiusLabel enabled it is not painted.

ClipAxisBackground The tick labels of the radial scales might be hard to read, when they are painted on top of the radial grid lines (or on top of a curve/spectrogram). When ClipAxisBackground the bounding rect of each label is added to the clip region.

SmartScaleDraw Don't paint the backbone of the radial axes, when they are very close to a line of the azimuth grid.

ClipGridLines All grid lines are clipped against the plot area before being painted. When the plot is zoomed in this will have an significant impact on the performance of the painting cde.

5.4.2.2 enum QwtPolarGrid::GridAttribute

Grid attributes.

See Also

setGridAttributes(), testGridAttributes()

Enumerator

AutoScaling When AutoScaling is enabled, the radial axes will be adjusted to the interval, that is currently visible on the canvas plot.

5.4.3 Constructor & Destructor Documentation

5.4.3.1 QwtPolarGrid::QwtPolarGrid ( ) [explicit]

Constructor.

Enables major and disables minor grid lines. The azimuth and right radial axis are visible. all other axes are hidden. Autoscaling is enabled.

5.4.4 Member Function Documentation

5.4.4.1 QFont QwtPolarGrid::axisFont (int *axisId*) const

Returns

Font for the tick labels of a specific axis

Parameters

axisId	Axis id (QwtPolar::Axis)

5.4.4.2 QPen QwtPolarGrid::axisPen (int *axisId*) const

Returns

Pen for painting a specific axis

Parameters

axisId	Axis id (QwtPolar::Axis)

See Also

setAxisPen()

5.4.4.3 const QwtRoundScaleDraw ∗ QwtPolarGrid::azimuthScaleDraw ( ) const

Returns

Scale draw for the azimuth scale

See Also

setAzimuthScaleDraw(), scaleDraw()

5.4.4.4 QwtRoundScaleDraw ∗ QwtPolarGrid::azimuthScaleDraw ( )

Returns

Scale draw for the azimuth scale

See Also

setAzimuthScaleDraw(), scaleDraw()

5.4.4.5 void QwtPolarGrid::draw (QPainter ∗ *painter,* const QwtScaleMap & *azimuthMap,* const QwtScaleMap & *radialMap,* const QPointF & *pole,* double *radius,* const QRectF & *canvasRect*) const [virtual]

Draw the grid and axes

Parameters

painter	Painter
azimuthMap	Maps azimuth values to values related to 0.0, M_2PI
radialMap	Maps radius values into painter coordinates.
pole	Position of the pole in painter coordinates
radius	Radius of the complete plot area in painter coordinates
canvasRect	Contents rect of the canvas in painter coordinates

Implements QwtPolarItem.

5.4.4.6 void QwtPolarGrid::drawAxis (QPainter ∗ *painter,* int *axisId*) const [protected]

Paint an axis

Parameters

painter	Painter
axisId	Axis id (QwtPolar::Axis)

5.4.4.7 void QwtPolarGrid::drawCircles (QPainter ∗ *painter,* const QRectF & *canvasRect,* const QPointF & *pole,* const QwtScaleMap & *radialMap,* const QList< double > & *values*) const [protected]

Draw circles

Parameters

painter	Painter
canvasRect	Contents rect of the canvas in painter coordinates
pole	Position of the pole in painter coordinates
radialMap	Maps radius values into painter coordinates.
values	Radial values, indicating the distances from the pole

5.4.4.8 void QwtPolarGrid::drawRays (QPainter ∗ *painter,* const QRectF & *canvasRect,* const QPointF & *pole,* double *radius,* const QwtScaleMap & *azimuthMap,* const QList< double > & *values*) const [protected]

Draw lines from the pole

Parameters

painter	Painter
canvasRect	Contents rect of the canvas in painter coordinates
pole	Position of the pole in painter coordinates
radius	Length of the lines in painter coordinates
azimuthMap	Maps azimuth values to values related to 0.0, M_2PI
values	Azimuth values, indicating the direction of the lines

5.4.4.9 bool QwtPolarGrid::isAxisVisible (int *axisId*) const

Returns

true if the axis is visible

Parameters

axisId	Axis id (QwtPolar::Axis)

See Also

showAxis()

5.4.4.10 bool QwtPolarGrid::isGridVisible (int *scaleId*) const

Returns

true if grid lines are enabled

Parameters

scaleId	Scale id (QwtPolar::Scale)

See Also

QwtPolar::Scale, showGrid()

5.4.4.11 bool QwtPolarGrid::isMinorGridVisible (int *scaleId*) const

Returns

true if minor grid lines are enabled

Parameters

scaleId	Scale id (QwtPolar::Scale)

See Also

showMinorGrid()

5.4.4.12 QPen QwtPolarGrid::majorGridPen (int *scaleId*) const

Returns

Pen for painting the major grid lines of a specific scale

Parameters

scaleId	Scale id (QwtPolar::Scale)

See Also

setMajorGridPen(), minorGridPen()

5.4.4.13 int QwtPolarGrid::marginHint ( ) const [virtual]

Returns

Number of pixels, that are necessary to paint the azimuth scale

See Also

QwtRoundScaleDraw::extent()

Reimplemented from QwtPolarItem.

5.4.4.14 QPen QwtPolarGrid::minorGridPen (int *scaleId*) const

Returns

Pen for painting the minor grid lines of a specific scale

Parameters

scaleId	Scale id (QwtPolar::Scale)

5.4.4.15 int QwtPolarGrid::rtti ( ) const [virtual]

Returns

QwtPlotItem::Rtti_PolarGrid

Reimplemented from QwtPolarItem.

5.4.4.16 const QwtScaleDraw ∗ QwtPolarGrid::scaleDraw (int *axisId*) const

Returns the scale draw of a specified axis

Parameters

axisId	axis index (QwtPolar::AxisLeft <= axisId <= QwtPolar::AxisBottom)

Returns

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

See Also

azimuthScaleDraw()

5.4.4.17 QwtScaleDraw ∗ QwtPolarGrid::scaleDraw (int *axisId*)

Returns the scale draw of a specified axis

Parameters

axisId	axis index (QwtPolar::AxisLeft <= axisId <= QwtPolar::AxisBottom)

Returns

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

See Also

setScaleDraw(), azimuthScaleDraw()

5.4.4.18 void QwtPolarGrid::setAxisFont (int *axisId,* const QFont & *font*)

Assign a font for the tick labels of a specific axis

Parameters

axisId	Axis id (QwtPolar::Axis)
font	new Font

5.4.4.19 void QwtPolarGrid::setAxisPen (int *axisId,* const QPen & *pen*)

Assign a pen for painting an axis

Parameters

axisId	Axis id (QwtPolar::Axis)
pen	Pen

See Also

axisPen()

5.4.4.20 void QwtPolarGrid::setAzimuthScaleDraw (QwtRoundScaleDraw ∗ *scaleDraw*)

Set a scale draw for the azimuth scale.

Parameters

scaleDraw	object responsible for drawing scales.

See Also

azimuthScaleDraw(), setScaleDraw()

5.4.4.21 void QwtPolarGrid::setDisplayFlag (DisplayFlag *flag*, bool *on* = `true`)

Change the display flags

Parameters

flag	See DisplayFlag
on	true/false

5.4.4.22 void QwtPolarGrid::setFont (const QFont & *font*)

Assign a font for all scale tick labels

Parameters

font	Font

See Also

setAxisFont()

5.4.4.23 void QwtPolarGrid::setGridAttribute (GridAttribute *attribute*, bool *on* = `true`)

Specify an attribute for the grid.

Parameters

attribute	Grid attribute
on	On/Off

/sa GridAttribute, testGridAttribute(), updateScaleDiv(), QwtPolarPlot::zoom(), QwtPolarPlot::scaleDiv()

5.4.4.24 void QwtPolarGrid::setMajorGridPen (const QPen & *pen*)

Assign a pen for the major grid lines

Parameters

pen	Pen

See Also

setPen(), setMinorGridPen(), majorGridPen

5.4.4.25 void QwtPolarGrid::setMajorGridPen (int *scaleId,* const QPen & *pen*)

Assign a pen for the major grid lines of a specific scale

Parameters

scaleId	Scale id (QwtPolar::Scale)
pen	Pen

See Also

setPen(), setMinorGridPen(), majorGridPen

5.4.4.26 void QwtPolarGrid::setMinorGridPen (const QPen & *pen*)

Assign a pen for the minor grid lines

Parameters

pen	Pen

See Also

setPen(), setMajorGridPen(), minorGridPen()

5.4.4.27 void QwtPolarGrid::setMinorGridPen (int *scaleId,* const QPen & *pen*)

Assign a pen for the minor grid lines of a specific scale

Parameters

scaleId	Scale id (QwtPolar::Scale)
pen	Pen

See Also

setPen(), setMajorGridPen(), minorGridPen

5.4.4.28 void QwtPolarGrid::setPen (const QPen & *pen*)

Assign a pen for all axes and grid lines

Parameters

pen	Pen

See Also

setMajorGridPen(), setMinorGridPen(), setAxisPen()

5.4.4.29 void QwtPolarGrid::setScaleDraw (int *axisId,* QwtScaleDraw ∗ *scaleDraw*)

Set a scale draw.

Parameters

axisId	axis index (QwtPolar::AxisLeft <= axisId <= QwtPolar::AxisBottom)
scaleDraw	object responsible for drawing scales.

See Also

scaleDraw(), setAzimuthScaleDraw()

5.4.4.30 void QwtPolarGrid::showAxis (int *axisId*, bool *show* = `true`)

Show/Hide an axis

Parameters

axisId	Axis id (QwtPolar::Axis)
show	true/false

See Also

isAxisVisible()

5.4.4.31 void QwtPolarGrid::showGrid (int *scaleId*, bool *show* = `true`)

Show/Hide grid lines for a scale

Parameters

scaleId	Scale id (QwtPolar::Scale)
show	true/false

5.4.4.32 void QwtPolarGrid::showMinorGrid (int *scaleId*, bool *show* = `true`)

Show/Hide minor grid lines for a scale

To display minor grid lines. showGrid() needs to be enabled too.

Parameters

scaleId	Scale id (QwtPolar::Scale)
show	true/false

See Also

showGrid

5.4.4.33 bool QwtPolarGrid::testDisplayFlag (DisplayFlag *flag*) const

Returns

true, if flag is enabled

Parameters

flag	See DisplayFlag

5.4.4.34 bool QwtPolarGrid::testGridAttribute (GridAttribute *attribute*) const

Returns

true, if attribute is enabled

See Also

GridAttribute, setGridAttribute()

5.4.4.35 void QwtPolarGrid::updateScaleDiv (const QwtScaleDiv & *azimuthScaleDiv*, const QwtScaleDiv & *radialScaleDiv*, const QwtInterval & *interval*) [virtual]

Update the item to changes of the axes scale division.

If AutoScaling is enabled the radial scale is calculated from the interval, otherwise the scales are adopted to the plot scales.

Parameters

azimuthScaleDiv	Scale division of the azimuth-scale
radialScaleDiv	Scale division of the radius-axis
interval	The interval of the radius-axis, that is visible on the canvas

See Also

QwtPolarPlot::setGridAttributes()

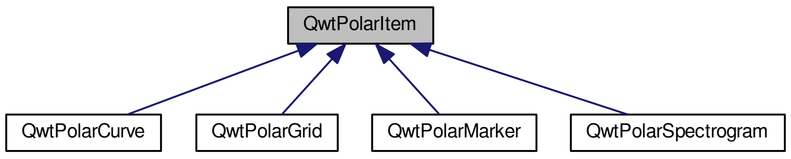
Reimplemented from QwtPolarItem.

5.5 QwtPolarItem Class Reference

Base class for items on a polar plot.

```
#include <qwt_polar_item.h>
```

Inheritance diagram for QwtPolarItem:



Public Types

- enum RttiValues {
  Rtti_PolarItem = 0, Rtti_PolarGrid, Rtti_PolarMarker, Rtti_PolarCurve,
  Rtti_PolarSpectrogram, Rtti_PolarUserItem = 1000 }

  Runtime type information.
- enum ItemAttribute { Legend = 0x01, AutoScale = 0x02 }

  Plot Item Attributes.
- enum RenderHint { RenderAntialiased = 0x01 }

  Render hints.
- typedef QFlags< ItemAttribute > ItemAttributes

  Item attributes.
- typedef QFlags< RenderHint > RenderHints

  Item attributes.

Public Member Functions

- QwtPolarItem (const QwtText &title=QwtText())
- virtual ∼QwtPolarItem ()

  Destroy the QwtPolarItem.
- void attach (QwtPolarPlot ∗plot)

  Attach the item to a plot.
- void detach ()

  This method detaches a QwtPolarItem from the QwtPolarPlot it has been associated with.
- QwtPolarPlot ∗ plot () const
- void setTitle (const QString &title)
- void setTitle (const QwtText &title)
- const QwtText & title () const
- virtual int rtti () const
- void setItemAttribute (ItemAttribute, bool on=true)
- bool testItemAttribute (ItemAttribute) const
- void setRenderHint (RenderHint, bool on=true)
- bool testRenderHint (RenderHint) const
- void setRenderThreadCount (uint numThreads)
- uint renderThreadCount () const
- double z () const
- void setZ (double z)

  Set the z value.
- void show ()

  Show the item.
- void hide ()

  Hide the item.
- virtual void setVisible (bool)
- bool isVisible () const
- virtual void itemChanged ()
- virtual void legendChanged ()
- virtual void draw (QPainter ∗painter, const QwtScaleMap &azimuthMap, const QwtScaleMap &radialMap, const QPointF &pole, double radius, const QRectF &canvasRect) const =0

  Draw the item.
- virtual QwtInterval boundingInterval (int scaleId) const
- virtual void updateScaleDiv (const QwtScaleDiv &, const QwtScaleDiv &, const QwtInterval &)

  Update the item to changes of the axes scale division.
- virtual int marginHint () const
- void setLegendIconSize (const QSize &)
- QSize legendIconSize () const
- virtual QList< QwtLegendData > legendData () const

  Return all information, that is needed to represent the item on the legend.
- virtual QwtGraphic legendIcon (int index, const QSizeF &) const

5.5.1 Detailed Description

Base class for items on a polar plot.

A QwtPolarItem is "something that can be painted on the canvas". It is connected to the QwtPolar framework by a couple of virtual methods, that are individually implemented in derived item classes.

QwtPolar offers an implementation of the most common types of items, but deriving from QwtPolarItem makes it easy to implement additional types of items.

5.5.2 Member Enumeration Documentation

5.5.2.1 enum QwtPolarItem::ItemAttribute

Plot Item Attributes.

See Also

setItemAttribute(), testItemAttribute()

Enumerator

Legend The item is represented on the legend.

AutoScale The boundingRect() of the item is included in the autoscaling calculation.

5.5.2.2 enum QwtPolarItem::RenderHint

Render hints.

See Also

setRenderHint(), testRenderHint()

Enumerator

RenderAntialiased Enable antialiasing.

5.5.2.3 enum QwtPolarItem::RttiValues

Runtime type information.

RttiValues is used to cast plot items, without having to enable runtime type information of the compiler.

Enumerator

Rtti_PolarItem Unspecific value, that can be used, when it doesn't matter.

Rtti_PolarGrid For QwtPolarGrid.

Rtti_PolarMarker For QwtPolarMarker.

Rtti_PolarCurve For QwtPolarCurve.

Rtti_PolarSpectrogram For QwtPolarSpectrogram.

Rtti_PolarUserItem Values >= Rtti_PolarUserItem are reserved for plot items not implemented in the Qwt-Polar library.

5.5.3 Constructor & Destructor Documentation

5.5.3.1 QwtPolarItem::QwtPolarItem (const QwtText & *title* = `QwtText()`) `[explicit]`

Constructor

Parameters

title	Item title, f.e used on a legend

See Also

setTitle()

5.5.4 Member Function Documentation

5.5.4.1 void QwtPolarItem::attach (QwtPolarPlot ∗ *plot*)

Attach the item to a plot.

This method will attach a QwtPolarItem to the QwtPolarPlot argument. It will first detach the QwtPolarItem from any plot from a previous call to attach (if necessary). If a NULL argument is passed, it will detach from any QwtPolarPlot it was attached to.

Parameters

plot	Plot widget

See Also

QwtPolarItem::detach()

5.5.4.2 QwtInterval QwtPolarItem::boundingInterval (int *scaleId*) const [virtual]

Interval, that is necessary to display the item

This interval can be useful for operations like clipping or autoscaling For items (like the grid), where a bounding interval makes no sense an invalid interval is returned.

Parameters

scaleId	Scale id (QwtPolar::Scale)

Returns

Bounding interval of the plot item for a specific scale

Reimplemented in QwtPolarCurve, QwtPolarSpectrogram, and QwtPolarMarker.

5.5.4.3 void QwtPolarItem::detach ( )

This method detaches a QwtPolarItem from the QwtPolarPlot it has been associated with.

detach() is equivalent to calling attach(NULL)

See Also

attach()

5.5.4.4 virtual void QwtPolarItem::draw (QPainter ∗ *painter,* const QwtScaleMap & *azimuthMap,* const QwtScaleMap & *radialMap,* const QPointF & *pole,* double *radius,* const QRectF & *canvasRect*) const [pure virtual]

Draw the item.

Parameters

painter	Painter
azimuthMap	Maps azimuth values to values related to 0.0, M_2PI
radialMap	Maps radius values into painter coordinates.
pole	Position of the pole in painter coordinates
radius	Radius of the complete plot area in painter coordinates

canvasRect	Contents rect of the canvas in painter coordinates

Implemented in QwtPolarGrid, QwtPolarCurve, QwtPolarSpectrogram, and QwtPolarMarker.

5.5.4.5 bool QwtPolarItem::isVisible ( ) const

Returns

true if visible

See Also

setVisible(), show(), hide()

5.5.4.6 void QwtPolarItem::itemChanged ( ) [virtual]

Update the legend and call QwtPolarPlot::autoRefresh for the parent plot.

See Also

updateLegend()

5.5.4.7 void QwtPolarItem::legendChanged ( ) [virtual]

Update the legend of the parent plot.

See Also

QwtPolarPlot::updateLegend(), itemChanged()

5.5.4.8 QList< QwtLegendData > QwtPolarItem::legendData ( ) const [virtual]

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

Most items are represented by one entry on the legend showing an icon and a text.

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().

See Also

title(), legendIcon(), QwtLegend

5.5.4.9 QwtGraphic QwtPolarItem::legendIcon (int *index*, const QSizeF & *size*) const [virtual]

Returns

Icon representing the item on the legend

The default implementation returns an invalid icon

Parameters

index	Index of the legend entry (usually there is only one)
size	Icon size

See Also

setLegendIconSize(), legendData()

Reimplemented in QwtPolarCurve.

5.5.4.10 QSize QwtPolarItem::legendIconSize ( ) const

Returns

Legend icon size

See Also

setLegendIconSize(), legendIcon()

5.5.4.11 int QwtPolarItem::marginHint ( ) const [virtual]

Some items like to display something (f.e. the azimuth axis) outside of the area of the interval of the radial scale. The default implementation returns 0 pixels

Returns

Hint for the margin

Reimplemented in QwtPolarGrid.

5.5.4.12 QwtPolarPlot * QwtPolarItem::plot ( ) const

Returns

Attached plot

5.5.4.13 uint QwtPolarItem::renderThreadCount ( ) const

Returns

Number of threads to be used for rendering. If numThreads() is set to 0, the system specific ideal thread count is used.

5.5.4.14 int QwtPolarItem::rtti ( ) const [virtual]

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

The rtti value is useful for environments, where the runtime type information is disabled and it is not possible to do a dynamic_cast<...>.

Returns

rtti value

See Also

RttiValues

Reimplemented in QwtPolarGrid, QwtPolarCurve, QwtPolarSpectrogram, and QwtPolarMarker.

5.5.4.15 void QwtPolarItem::setItemAttribute (ItemAttribute *attribute*, bool *on* = true)

Toggle an item attribute

Parameters

attribute	Attribute type
on	true/false

See Also

testItemAttribute(), ItemAttribute

5.5.4.16 void QwtPolarItem::setLegendIconSize (const QSize & *size*)

Set the size of the legend icon

The default setting is 8x8 pixels

Parameters

size	Size

See Also

legendIconSize(), legendIcon()

5.5.4.17 void QwtPolarItem::setRenderHint (RenderHint *hint,* bool *on* = `true`)

Toggle an render hint

Parameters

hint	Render hint
on	true/false

See Also

testRenderHint(), RenderHint

5.5.4.18 void QwtPolarItem::setRenderThreadCount (uint *numThreads*)

On multi core systems rendering of certain plot item (f.e QwtPolarSpectrogram) can be done in parallel in several threads.

The default setting is set to 1.

Parameters

numThreads	Number of threads to be used for rendering. If numThreads is set to 0, the system specific ideal thread count is used.

The default thread count is 1 (= no additional threads)

5.5.4.19 void QwtPolarItem::setTitle (const QString & *title*)

Set a new title

Parameters

title	Title

See Also

title()

5.5.4.20 void QwtPolarItem::setTitle (const QwtText & *title*)

Set a new title

Parameters

title	Title

See Also

title()

5.5.4.21 void QwtPolarItem::setVisible (bool *on*) [virtual]

Show/Hide the item

Parameters

on	Show if true, otherwise hide

See Also

isVisible(), show(), hide()

5.5.4.22 void QwtPolarItem::setZ (double *z*)

Set the z value.

Plot items are painted in increasing z-order.

Parameters

z	Z-value

See Also

z(), QwtPolarItemDict::itemList()

5.5.4.23 bool QwtPolarItem::testItemAttribute (ItemAttribute *attribute*) const

Test an item attribute

Parameters

attribute	Attribute type

Returns

true/false

See Also

setItemAttribute(), ItemAttribute

5.5.4.24 bool QwtPolarItem::testRenderHint (RenderHint *hint*) const

Test a render hint

Parameters

hint	Render hint

Returns

true/false

See Also

setRenderHint(), RenderHint

5.5.4.25 const QwtText & QwtPolarItem::title ( ) const

Returns

Title of the item

See Also

setTitle()

5.5.4.26 void QwtPolarItem::updateScaleDiv (const QwtScaleDiv & *azimuthScaleDiv*, const QwtScaleDiv & *radialScaleDiv*, const QwtInterval & *interval*) `[virtual]`

Update the item to changes of the axes scale division.

Update the item, when the axes of plot have changed. The default implementation does nothing, but items that depend on the scale division (like QwtPolarGrid()) have to reimplement updateScaleDiv()

Parameters

azimuthScaleDiv	Scale division of the azimuth-scale
radialScaleDiv	Scale division of the radius-axis
interval	The interval of the radius-axis, that is visible on the canvas

See Also

QwtPolarPlot::updateAxes()

Reimplemented in QwtPolarGrid.

5.5.4.27 double QwtPolarItem::z ( ) const

Plot items are painted in increasing z-order.

Returns

Z value

See Also

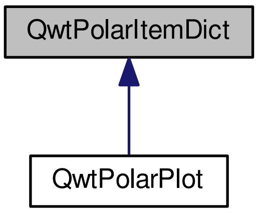
setZ(), QwtPolarItemDict::itemList()

5.6 QwtPolarItemDict Class Reference

A dictionary for polar plot items.

`#include <qwt_polar_itemdict.h>`

Inheritance diagram for QwtPolarItemDict:



Public Member Functions

- QwtPolarItemDict ()
- ∼QwtPolarItemDict ()
- void setAutoDelete (bool)
- bool autoDelete () const
- const QwtPolarItemList & itemList () const

    A QwtPolarItemList of all attached plot items.
- void detachItems (int rtti=QwtPolarItem::Rtti_PolarItem, bool autoDelete=true)

Protected Member Functions

- void insertItem (QwtPolarItem ∗)
- void removeItem (QwtPolarItem ∗)

5.6.1 Detailed Description

A dictionary for polar plot items.

QwtPolarItemDict organizes polar plot items in increasing z-order. If autoDelete() is enabled, all attached items will be deleted in the destructor of the dictionary.

See Also

QwtPolarItem::attach(), QwtPolarItem::detach(), QwtPolarItem::z()

5.6.2 Constructor & Destructor Documentation

5.6.2.1 QwtPolarItemDict::QwtPolarItemDict ( ) [explicit]

Constructor

Auto deletion is enabled.

See Also

setAutoDelete, attachItem

5.6.2.2 QwtPolarItemDict::∼QwtPolarItemDict ( )

Destructor

If autoDelete is on, all attached items will be deleted

See Also

setAutoDelete, autoDelete, attachItem

5.6.3 Member Function Documentation

5.6.3.1 bool QwtPolarItemDict::autoDelete ( ) const

Returns

true if auto deletion is enabled

See Also

setAutoDelete, attachItem

5.6.3.2 void QwtPolarItemDict::detachItems (int *rtti* = **QwtPolarItem::Rtti_PolarItem**, bool *autoDelete* = `true`)

Detach items from the dictionary

Parameters

rtti	In case of QwtPolarItem::Rtti_PlotItem detach all items otherwise only those items of the type rtti.
autoDelete	If true, delete all detached items

5.6.3.3 void QwtPolarItemDict::insertItem (**QwtPolarItem** ∗ *item*) `[protected]`

Insert a plot item

Parameters

item	PlotItem

See Also

removeItem()

5.6.3.4 const QwtPolarItemList & QwtPolarItemDict::itemList ( ) const

A QwtPolarItemList of all attached plot items.

Returns

List of all attached plot items.

Note

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.

5.6.3.5 void QwtPolarItemDict::removeItem (**QwtPolarItem** ∗ *item*) `[protected]`

Remove a plot item

Parameters

item	PlotItem

See Also

insertItem()

5.6.3.6 void QwtPolarItemDict::setAutoDelete (bool *autoDelete*)

En/Disable Auto deletion

If Auto deletion is on all attached plot items will be deleted in the destructor of QwtPolarItemDict. The default value is on.

See Also

autoDelete, attachItem

5.7 QwtPolarLayout Class Reference

Layout class for QwtPolarPlot.

```
#include <qwt_polar_layout.h>
```

Public Types

- enum Option { IgnoreScrollbars = 0x01, IgnoreFrames = 0x02, IgnoreTitle = 0x04, IgnoreLegend = 0x08 }

  Options to configure the plot layout engine.
- typedef QFlags< Option > Options

  Options to configure the plot layout engine.

Public Member Functions

- QwtPolarLayout ()

  Constructor.
- virtual ∼QwtPolarLayout ()

  Destructor.
- void setLegendPosition (QwtPolarPlot::LegendPosition pos, double ratio)

  Specify the position of the legend.
- void setLegendPosition (QwtPolarPlot::LegendPosition pos)

  Specify the position of the legend.
- QwtPolarPlot::LegendPosition legendPosition () const
- void setLegendRatio (double ratio)
- double legendRatio () const
- virtual void activate (const QwtPolarPlot ∗, const QRectF &rect, Options options=0)

  Recalculate the geometry of all components.
- virtual void invalidate ()
- const QRectF & titleRect () const
- const QRectF & legendRect () const
- const QRectF & canvasRect () const

Protected Member Functions

- QRectF layoutLegend (Options options, QRectF &) const

5.7.1 Detailed Description

Layout class for QwtPolarPlot.

Organizes the geometry for the different QwtPolarPlot components. It is used by the QwtPolar widget to organize its internal widgets or by QwtPolarRnderer to render its content to a QPaintDevice like a QPrinter, QPixmap/QImage or QSvgRenderer.

5.7.2 Member Enumeration Documentation

5.7.2.1 enum **QwtPolarLayout::Option**

Options to configure the plot layout engine.

Enumerator

- ***IgnoreScrollbars*** Ignore the dimension of the scrollbars.
- ***IgnoreFrames*** Ignore all frames.
- ***IgnoreTitle*** Ignore the title.
- ***IgnoreLegend*** Ignore the legend.

5.7.3 Member Function Documentation

5.7.3.1 void QwtPolarLayout::activate (const **QwtPolarPlot** ∗ *plot,* const QRectF & *boundingRect,* **Options** *options* = 0) [virtual]

Recalculate the geometry of all components.

Parameters

plot	Plot to be layout
boundingRect	Rect where to place the components
options	Options

See Also

invalidate(), titleRect(), legendRect(), canvasRect()

5.7.3.2 const QRectF & QwtPolarLayout::canvasRect ( ) const

Returns

Geometry for the canvas

See Also

activate(), invalidate()

5.7.3.3 void QwtPolarLayout::invalidate ( ) [virtual]

Invalidate the geometry of all components.

See Also

activate()

5.7.3.4 QRectF QwtPolarLayout::layoutLegend (**Options** *options,* QRectF & *rect*) const [protected]

Find the geometry for the legend

Parameters

options	Options how to layout the legend
rect	Rectangle where to place the legend

Returns

Geometry for the legend

5.7.3.5 QwtPolarPlot::LegendPosition QwtPolarLayout::legendPosition ( ) const

Returns

Position of the legend

See Also

setLegendPosition(), QwtPolarPlot::setLegendPosition(), QwtPolarPlot::legendPosition()

5.7.3.6 double QwtPolarLayout::legendRatio ( ) const

Returns

The relative size of the legend in the plot.

See Also

setLegendPosition()

5.7.3.7 const QRectF & QwtPolarLayout::legendRect ( ) const

Returns

Geometry for the legend

See Also

activate(), invalidate()

5.7.3.8 void QwtPolarLayout::setLegendPosition (QwtPolarPlot::LegendPosition *pos*, double *ratio*)

Specify the position of the legend.

Parameters

pos	The legend's position.
ratio	Ratio between legend and the bounding rect of title, canvas and axes. The legend will be shrinked 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

QwtPolarPlot::setLegendPosition()

5.7.3.9 void QwtPolarLayout::setLegendPosition (QwtPolarPlot::LegendPosition *pos*)

Specify the position of the legend.

Parameters

pos	The legend's position. Valid values are `QwtPolarPlot::LeftLegend`, `QwtPolarPlot::RightLegend`, `QwtPolarPlot::TopLegend`, `QwtPolarPlot::BottomLegend`.

See Also

QwtPolarPlot::setLegendPosition()

5.7.3.10 void QwtPolarLayout::setLegendRatio (double *ratio*)

Specify the relative size of the legend in the plot

Parameters

ratio	Ratio between legend and the bounding rect of title, canvas and axes. The legend will be shrinked 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.

5.7.3.11 const QRectF & QwtPolarLayout::titleRect ( ) const

Returns

Geometry for the title

See Also

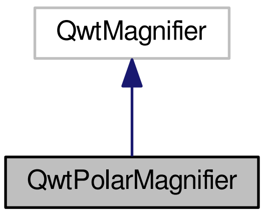
activate(), invalidate()

5.8 QwtPolarMagnifier Class Reference

QwtPolarMagnifier provides zooming, by magnifying in steps.

`#include <qwt_polar_magnifier.h>`

Inheritance diagram for QwtPolarMagnifier:



Public Member Functions

- QwtPolarMagnifier (QwtPolarCanvas ∗)
- virtual ∼QwtPolarMagnifier ()

Destructor.

- void setUnzoomKey (int key, int modifiers)
- void getUnzoomKey (int &key, int &modifiers) const
- QwtPolarPlot ∗ plot ()
- const QwtPolarPlot ∗ plot () const
- QwtPolarCanvas ∗ canvas ()
- const QwtPolarCanvas ∗ canvas () const

Protected Member Functions

- virtual void rescale (double factor)
- void unzoom ()

  Unzoom the plot widget.
- virtual void widgetKeyPressEvent (QKeyEvent ∗)

5.8.1 Detailed Description

QwtPolarMagnifier provides zooming, by magnifying in steps.

Using QwtPlotMagnifier a plot can be zoomed in/out in steps using keys, the mouse wheel or moving a mouse button in vertical direction.

Together with QwtPolarPanner it is possible to implement an individual navigation of the plot canvas.

See Also

QwtPolarPanner, QwtPolarPlot, QwtPolarCanvas

5.8.2 Constructor & Destructor Documentation

5.8.2.1 QwtPolarMagnifier::QwtPolarMagnifier (**QwtPolarCanvas** ∗ *canvas*) [explicit]

Constructor

Parameters

canvas	Plot canvas to be magnified

5.8.3 Member Function Documentation

5.8.3.1 QwtPolarCanvas ∗ QwtPolarMagnifier::canvas ( )

Returns

Observed plot canvas

5.8.3.2 const QwtPolarCanvas ∗ QwtPolarMagnifier::canvas ( ) const

Returns

Observed plot canvas

5.8.3.3 void QwtPolarMagnifier::getUnzoomKey (int & *key,* int & *modifiers*) const

Returns

Key, and modifiers that are used for unzooming

Parameters

key	Key code
modifiers	Modifiers

See Also

setUnzoomKey(), QwtPolarPlot::unzoom()

5.8.3.4 QwtPolarPlot ∗ QwtPolarMagnifier::plot ( )

Returns

Observed plot

5.8.3.5 const QwtPolarPlot ∗ QwtPolarMagnifier::plot ( ) const

Returns

observed plot

5.8.3.6 void QwtPolarMagnifier::rescale (double *factor*) [protected], [virtual]

Zoom in/out the zoomed area

Parameters

factor	A value $<$ 1.0 zooms in, a value $>$ 1.0 zooms out.

5.8.3.7 void QwtPolarMagnifier::setUnzoomKey (int *key,* int *modifiers*)

Assign key and modifiers, that are used for unzooming The default combination is Qt::Key_Home + Qt::NoModifier.

Parameters

key	Key code
modifiers	Modifiers

See Also

getUnzoomKey(), QwtPolarPlot::unzoom()

5.8.3.8 void QwtPolarMagnifier::widgetKeyPressEvent (QKeyEvent ∗ *event*) [protected], [virtual]

Handle a key press event for the observed widget.

Parameters

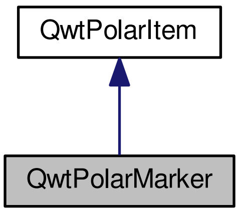
event	Key event

5.9 QwtPolarMarker Class Reference

A class for drawing markers.

`#include <qwt_polar_marker.h>`

Inheritance diagram for QwtPolarMarker:



Public Member Functions

- QwtPolarMarker ()

    Sets alignment to Qt::AlignCenter, and style to NoLine.
- virtual ∼QwtPolarMarker ()

    Destructor.
- virtual int rtti () const
- void setPosition (const QwtPointPolar &)

    Change the position of the marker.
- QwtPointPolar position () const
- void setSymbol (const QwtSymbol ∗s)

    Assign a symbol.
- const QwtSymbol ∗ symbol () const
- void setLabel (const QwtText &)

    Set the label.
- QwtText label () const
- void setLabelAlignment (Qt::Alignment)

    Set the alignment of the label.
- Qt::Alignment labelAlignment () const
- virtual void draw (QPainter ∗painter, const QwtScaleMap &azimuthMap, const QwtScaleMap &radialMap, const QPointF &pole, double radius, const QRectF &canvasRect) const
- virtual QwtInterval boundingInterval (int scaleId) const

Additional Inherited Members

5.9.1 Detailed Description

A class for drawing markers.

A marker can be a a symbol, a label or a 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 alignment refers to the center point of the marker, which means, for example, that the label would be painted left above the center point if the alignment was set to AlignLeft|AlignTop.

5.9.2 Member Function Documentation

5.9.2.1 QwtInterval QwtPolarMarker::boundingInterval (int *scaleId*) const [virtual]

Interval, that is necessary to display the item This interval can be useful for operations like clipping or autoscaling

Parameters

scaleId	Scale index

Returns

bounding interval (== position)

See Also

position()

Reimplemented from QwtPolarItem.

5.9.2.2 void QwtPolarMarker::draw (QPainter ∗ *painter,* const QwtScaleMap & *azimuthMap,* const QwtScaleMap & *radialMap,* const QPointF & *pole,* double *radius,* const QRectF & *canvasRect*) const [virtual]

Draw the marker

Parameters

painter	Painter
azimuthMap	Maps azimuth values to values related to 0.0, M_2PI
radialMap	Maps radius values into painter coordinates.
pole	Position of the pole in painter coordinates
radius	Radius of the complete plot area in painter coordinates
canvasRect	Contents rect of the canvas in painter coordinates

Implements QwtPolarItem.

5.9.2.3 QwtText QwtPolarMarker::label ( ) const

Returns

the label

See Also

setLabel()

5.9.2.4 Qt::Alignment QwtPolarMarker::labelAlignment ( ) const

Returns

the label alignment

See Also

setLabelAlignment()

5.9.2.5 QwtPointPolar QwtPolarMarker::position ( ) const

Returns

Position of the marker

5.9.2.6 int QwtPolarMarker::rtti ( ) const [virtual]

Returns

QwtPolarItem::Rtti_PlotMarker

Reimplemented from QwtPolarItem.

5.9.2.7 void QwtPolarMarker::setLabel (const QwtText & *label*)

Set the label.

Parameters

label	label text

See Also

label()

5.9.2.8 void QwtPolarMarker::setLabelAlignment (Qt::Alignment *align*)

Set the alignment of the label.

The alignment determines where the label is drawn relative to the marker's position.

Parameters

align	Alignment. A combination of AlignTop, AlignBottom, AlignLeft, AlignRight, AlignCenter, Algn-HCenter, AlignVCenter.

See Also

labelAlignment()

5.9.2.9 void QwtPolarMarker::setSymbol (const QwtSymbol $*$ *symbol*)

Assign a symbol.

Parameters

symbol	New symbol

See Also

symbol()

5.9.2.10 const QwtSymbol $*$ QwtPolarMarker::symbol ( ) const

Returns

the symbol

See Also

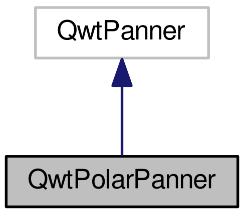
setSymbol(), QwtSymbol

5.10 QwtPolarPanner Class Reference

QwtPolarPanner provides panning of a polar plot canvas.

```
#include <qwt_polar_panner.h>
```

Inheritance diagram for QwtPolarPanner:



Public Member Functions

- QwtPolarPanner (QwtPolarCanvas ∗)

    Create a plot panner for a polar plot canvas.
- virtual ∼QwtPolarPanner ()

    Destructor.
- QwtPolarPlot ∗ plot ()
- const QwtPolarPlot ∗ plot () const
- QwtPolarCanvas ∗ canvas ()
- const QwtPolarCanvas ∗ canvas () const

Protected Slots

- virtual void movePlot (int dx, int dy)

Protected Member Functions

- virtual void widgetMousePressEvent (QMouseEvent ∗)

5.10.1 Detailed Description

QwtPolarPanner provides panning of a polar plot canvas.

QwtPolarPanner is a panner for a QwtPolarCanvas, that adjusts the visible area after dropping the canvas on its new position.

Together with QwtPolarMagnifier individual ways of navigating on a QwtPolarPlot widget can be implemented easily.

See Also

QwtPolarMagnifier

5.10.2 Member Function Documentation

5.10.2.1 QwtPolarCanvas ∗ QwtPolarPanner::canvas ( )

Returns

observed plot canvas

5.10.2.2 const QwtPolarCanvas * QwtPolarPanner::canvas ( ) const

Returns

observed plot canvas

5.10.2.3 void QwtPolarPanner::movePlot (int *dx*, int *dy*) `[protected]`, `[virtual]`, `[slot]`

Adjust the zoomed area according to dx/dy

Parameters

dx	Pixel offset in x direction
dy	Pixel offset in y direction

See Also

QwtPanner::panned(), QwtPolarPlot::zoom()

5.10.2.4 QwtPolarPlot * QwtPolarPanner::plot ( )

Returns

observed plot

5.10.2.5 const QwtPolarPlot * QwtPolarPanner::plot ( ) const

Returns

observed plot

5.10.2.6 void QwtPolarPanner::widgetMousePressEvent (QMouseEvent * *event*) `[protected]`, `[virtual]`

Block panning when the plot zoom factor is >= 1.0.

Parameters

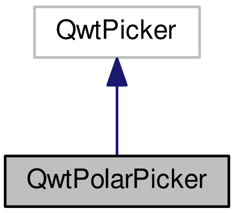
event	Mouse event

5.11 QwtPolarPicker Class Reference

QwtPolarPicker provides selections on a plot canvas.

```
#include <qwt_polar_picker.h>
```

Inheritance diagram for QwtPolarPicker:



Signals

- void selected (const QwtPointPolar &pos)
- void selected (const QVector< QwtPointPolar > &points)
- void appended (const QwtPointPolar &pos)
- void moved (const QwtPointPolar &pos)

Public Member Functions

- QwtPolarPicker (QwtPolarCanvas ∗)

  Create a polar plot picker.
- virtual ∼QwtPolarPicker ()

  Destructor.
- QwtPolarPicker (RubberBand rubberBand, DisplayMode trackerMode, QwtPolarCanvas ∗)
- QwtPolarPlot ∗ plot ()
- const QwtPolarPlot ∗ plot () const
- QwtPolarCanvas ∗ canvas ()
- const QwtPolarCanvas ∗ canvas () const
- virtual QRect pickRect () const

Protected Member Functions

- QwtPointPolar invTransform (const QPoint &) const
- virtual QwtText trackerText (const QPoint &) const
- virtual QwtText trackerTextPolar (const QwtPointPolar &) const

  Translate a position into a position string.
- virtual void move (const QPoint &)
- virtual void append (const QPoint &)
- virtual bool end (bool ok=true)

5.11.1 Detailed Description

QwtPolarPicker provides selections on a plot canvas.

QwtPolarPicker is a QwtPicker tailored for selections on a polar plot canvas.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 QwtPolarPicker::QwtPolarPicker (QwtPolarCanvas ∗ *canvas*) [explicit]

Create a polar plot picker.

Parameters

canvas	Plot canvas to observe, also the parent object

5.11.2.2 QwtPolarPicker::QwtPolarPicker (RubberBand *rubberBand*, DisplayMode *trackerMode*, QwtPolarCanvas ∗ *canvas*) [explicit]

Create a plot picker

Parameters

rubberBand	Rubberband style
trackerMode	Tracker mode
canvas	Plot canvas to observe, also the parent object

See Also

QwtPicker, QwtPicker::setSelectionFlags(), QwtPicker::setRubberBand(), QwtPicker::setTrackerMode QwtPolarPlot::autoReplot(), QwtPolarPlot::replot(), scaleRect()

5.11.3 Member Function Documentation

5.11.3.1 void QwtPolarPicker::append (const QPoint & *pos*) [protected], [virtual]

Append a point to the selection and update rubberband and tracker.

Parameters

pos	Additional point

See Also

isActive, begin(), end(), move(), appended()

Note

The appended(const QPoint &), appended(const QDoublePoint &) signals are emitted.

5.11.3.2 void QwtPolarPicker::appended (const QwtPointPolar & *pos*) [signal]

A signal emitted when a point has been appended to the selection

Parameters

pos	Position of the appended point.

See Also

append(). moved()

5.11.3.3 QwtPolarCanvas ∗ QwtPolarPicker::canvas ( )

Returns

Observed plot canvas

5.11.3.4 const QwtPolarCanvas ∗ QwtPolarPicker::canvas ( ) const

Returns

Observed plot canvas

5.11.3.5 bool QwtPolarPicker::end (bool *ok* = `true`) `[protected]`, `[virtual]`

Close a selection setting the state to inactive.

Parameters

ok	If true, complete the selection and emit selected signals otherwise discard the selection.

Returns

true if the selection is accepted, false otherwise

5.11.3.6 QwtPointPolar QwtPolarPicker::invTransform (const QPoint & *pos*) const [protected]

Translate a point from widget into plot coordinates

Parameters

pos	Point in widget coordinates of the plot canvas

Returns

Point in plot coordinates

See Also

transform(), canvas()

5.11.3.7 void QwtPolarPicker::move (const QPoint & *pos*) [protected], [virtual]

Move the last point of the selection

Parameters

pos	New position

See Also

isActive, begin(), end(), append()

Note

The moved(const QPoint &), moved(const QDoublePoint &) signals are emitted.

5.11.3.8 void QwtPolarPicker::moved (const QwtPointPolar & *pos*) [signal]

A signal emitted whenever the last appended point of the selection has been moved.

Parameters

pos	Position of the moved last point of the selection.

See Also

move(), appended()

5.11.3.9 QRect QwtPolarPicker::pickRect ( ) const [virtual]

Returns

Bounding rectangle of the region, where picking is supported.

5.11.3.10 QwtPolarPlot ∗ QwtPolarPicker::plot ( )

Returns

Plot widget, containing the observed plot canvas

5.11.3.11 const QwtPolarPlot ∗ QwtPolarPicker::plot ( ) const

Returns

Plot widget, containing the observed plot canvas

5.11.3.12 void QwtPolarPicker::selected (const QwtPointPolar & *pos*) `[signal]`

A signal emitted in case of selectionFlags() & PointSelection.

Parameters

pos	Selected point

5.11.3.13 void QwtPolarPicker::selected (const QVector< QwtPointPolar > & *points*) `[signal]`

A signal emitting the selected points, at the end of a selection.

Parameters

points	Selected points

5.11.3.14 QwtText QwtPolarPicker::trackerText (const QPoint & *pos*) const `[protected]`, `[virtual]`

Translate a pixel position into a position string

Parameters

pos	Position in pixel coordinates

Returns

Position string

5.11.3.15 QwtText QwtPolarPicker::trackerTextPolar (const QwtPointPolar & *pos*) const `[protected]`, `[virtual]`

Translate a position into a position string.

In case of HLineRubberBand the label is the value of the y position, in case of VLineRubberBand the value of the x position. Otherwise the label contains x and y position separated by a ',' .

The format for the double to string conversion is "%.4f".

Parameters

pos	Position

Returns

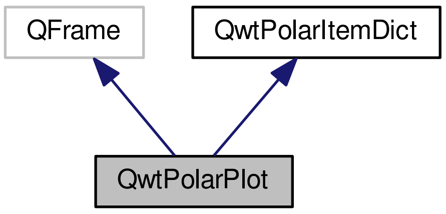
Position string

5.12 QwtPolarPlot Class Reference

A plotting widget, displaying a polar coordinate system.

```
#include <qwt_polar_plot.h>
```

Inheritance diagram for QwtPolarPlot:



Public Types

- enum LegendPosition {
  LeftLegend, RightLegend, BottomLegend, TopLegend,
  ExternalLegend }

Public Slots

- virtual void replot ()

  Redraw the plot.
- void autoRefresh ()

  Replots the plot if QwtPlot::autoReplot() is `true`*.*
- void setAzimuthOrigin (double)

  Change the origin of the azimuth scale.

Signals

- void itemAttached (QwtPolarItem ∗plotItem, bool on)
- void legendDataChanged (const QVariant &itemInfo, const QList< QwtLegendData > &data)
- void layoutChanged ()

Public Member Functions

- QwtPolarPlot (QWidget ∗parent=NULL)
- QwtPolarPlot (const QwtText &title, QWidget ∗parent=NULL)
- virtual ∼QwtPolarPlot ()

  Destructor.
- void setTitle (const QString &)
- void setTitle (const QwtText &)
- QwtText title () const
- QwtTextLabel ∗ titleLabel ()
- const QwtTextLabel ∗ titleLabel () const
- void setAutoReplot (bool tf=true)

  Set or reset the autoReplot option.
- bool autoReplot () const
- void setAutoScale (int scaleId)

Enable autoscaling.
- bool hasAutoScale (int scaleId) const
- void setScaleMaxMinor (int scaleId, int maxMinor)
- int scaleMaxMinor (int scaleId) const
- int scaleMaxMajor (int scaleId) const
- void setScaleMaxMajor (int scaleId, int maxMajor)
- QwtScaleEngine ∗ scaleEngine (int scaleId)
- const QwtScaleEngine ∗ scaleEngine (int scaleId) const
- void setScaleEngine (int scaleId, QwtScaleEngine ∗)
- void setScale (int scaleId, double min, double max, double step=0)

  Disable autoscaling and specify a fixed scale for a selected scale.
- void setScaleDiv (int scaleId, const QwtScaleDiv &)

  Disable autoscaling and specify a fixed scale for a selected scale.
- const QwtScaleDiv ∗ scaleDiv (int scaleId) const

  Return the scale division of a specified scale.
- QwtScaleDiv ∗ scaleDiv (int scaleId)

  Return the scale division of a specified scale.
- QwtScaleMap scaleMap (int scaleId, double radius) const
- QwtScaleMap scaleMap (int scaleId) const
- void updateScale (int scaleId)
- double azimuthOrigin () const
- void zoom (const QwtPointPolar &, double factor)

  Translate and in/decrease the zoom factor.
- void unzoom ()
- QwtPointPolar zoomPos () const
- double zoomFactor () const
- QwtPolarCanvas ∗ canvas ()
- const QwtPolarCanvas ∗ canvas () const
- void setPlotBackground (const QBrush &c)

  Set the background of the plot area.
- const QBrush & plotBackground () const
- virtual void drawCanvas (QPainter ∗, const QRectF &) const
- void insertLegend (QwtAbstractLegend ∗, LegendPosition=RightLegend, double ratio=-1.0)

  Insert a legend.
- QwtAbstractLegend ∗ legend ()
- const QwtAbstractLegend ∗ legend () const
- void updateLegend ()
- void updateLegend (const QwtPolarItem ∗)
- QwtPolarLayout ∗ plotLayout ()
- const QwtPolarLayout ∗ plotLayout () const
- QwtInterval visibleInterval () const
- QRectF plotRect () const
- QRectF plotRect (const QRectF &) const

  Calculate the bounding rect of the plot area.
- int plotMarginHint () const
- virtual QVariant itemToInfo (QwtPolarItem ∗) const

  Build an information, that can be used to identify a plot item on the legend.
- virtual QwtPolarItem ∗ infoToItem (const QVariant &) const

  Identify the plot item according to an item info object, that has bee generated from itemToInfo().

Protected Member Functions

- virtual bool event (QEvent ∗)

  Qt event handler.
- virtual void resizeEvent (QResizeEvent ∗)

  Resize and update internal layout.
- virtual void updateLayout ()

  Rebuild the layout.
- virtual void drawItems (QPainter ∗painter, const QwtScaleMap &radialMap, const QwtScaleMap &azimuthMap, const QPointF &pole, double radius, const QRectF &canvasRect) const

Friends

- class **QwtPolarItem**

5.12.1 Detailed Description

A plotting widget, displaying a polar coordinate system.

An unlimited number of plot items can be displayed on its canvas. Plot items might be curves (QwtPolarCurve), markers (QwtPolarMarker), the grid (QwtPolarGrid), or anything else derived from QwtPolarItem.

The coordinate system is defined by a radial and a azimuth scale. The scales at the axes can be explicitely set (QwtScaleDiv), or are calculated from the plot items, using algorithms (QwtScaleEngine) which can be configured separately for each axis. Autoscaling is supported for the radial scale.

In opposite to QwtPlot the scales might be different from the view, that is displayed on the canvas. The view can be changed by zooming - f.e. by using QwtPolarPanner or QwtPolarMaginfier.

5.12.2 Member Enumeration Documentation

5.12.2.1 enum QwtPolarPlot::LegendPosition

Position of the legend, relative to the canvas.

See Also

insertLegend()

Enumerator

LeftLegend The legend will be left from the canvas.

RightLegend The legend will be right from the canvas.

BottomLegend The legend will be below the canvas.

TopLegend The legend will be between canvas and title.

ExternalLegend External means that only the content of the legend will be handled by QwtPlot, but not its geometry. This might be interesting if an application wants to have a legend in an external window (or on the canvas).

Note

The legend is not painted by QwtPolarRenderer

5.12.3 Constructor & Destructor Documentation

5.12.3.1 QwtPolarPlot::QwtPolarPlot (QWidget ∗ *parent* = `NULL`) `[explicit]`

Constructor

Parameters

parent	Parent widget

5.12.3.2 QwtPolarPlot::QwtPolarPlot (const QwtText & *title*, QWidget ∗ *parent* = `NULL`)

Constructor

Parameters

title	Title text
parent	Parent widget

5.12.4 Member Function Documentation

5.12.4.1 bool QwtPolarPlot::autoReplot ( ) const

Returns

true if the autoReplot option is set.

5.12.4.2 double QwtPolarPlot::azimuthOrigin ( ) const

The azimuth origin is the angle where the azimuth scale shows the value 0.0.

Returns

Origin of the azimuth scale

See Also

setAzimuthOrigin()

5.12.4.3 QwtPolarCanvas ∗ QwtPolarPlot::canvas ( )

Returns

the plot's canvas

5.12.4.4 const QwtPolarCanvas ∗ QwtPolarPlot::canvas ( ) const

Returns

the plot's canvas

5.12.4.5 void QwtPolarPlot::drawCanvas (QPainter ∗ *painter*, const QRectF & *canvasRect*) const `[virtual]`

Redraw the canvas.

Parameters

painter	Painter used for drawing
canvasRect	Contents rect of the canvas

5.12.4.6 void QwtPolarPlot::drawItems (QPainter ∗ *painter*, const QwtScaleMap & *azimuthMap*, const QwtScaleMap & *radialMap*, const QPointF & *pole*, double *radius*, const QRectF & *canvasRect*) const `[protected]`, `[virtual]`

Redraw the canvas items.

Parameters

painter	Painter used for drawing
azimuthMap	Maps azimuth values to values related to 0.0, M_2PI
radialMap	Maps radius values into painter coordinates.
pole	Position of the pole in painter coordinates
radius	Radius of the complete plot area in painter coordinates
canvasRect	Contents rect of the canvas in painter coordinates

5.12.4.7 bool QwtPolarPlot::event (QEvent * *e*) `[protected]`, `[virtual]`

Qt event handler.

Handles QEvent::LayoutRequest and QEvent::PolishRequest

Parameters

e	Qt Event

Returns

True, when the event was processed

5.12.4.8 bool QwtPolarPlot::hasAutoScale (int *scaleId*) const

Returns

`true` if autoscaling is enabled

Parameters

scaleId	Scale index

See Also

setAutoScale()

5.12.4.9 QwtPolarItem * QwtPolarPlot::infoToItem (const QVariant & *itemInfo*) const `[virtual]`

Identify the plot item according to an item info object, that has bee generated from itemToInfo().

The default implementation simply tries to unwrap a QwtPlotItem pointer:

```
if ( itemInfo.canConvert<QwtPlotItem *>() )
    return qvariant_cast<QwtPlotItem *>( itemInfo );
```

Parameters

itemInfo	Plot item

Returns

A plot item, when successful, otherwise a NULL pointer.

See Also

itemToInfo()

5.12.4.10 void QwtPolarPlot::insertLegend (QwtAbstractLegend ∗ *legend*, QwtPolarPlot::LegendPosition *pos* = RightLegend, double *ratio* = `-1.0`)

Insert a legend.

If the position legend is `QwtPolarPlot::LeftLegend` or `QwtPolarPlot::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.

If pos != QwtPolarPlot::ExternalLegend the plot widget will become parent of the legend. It will be deleted when the plot is deleted, or another legend is set with insertLegend().

Parameters

legend	Legend
pos	The legend's position. For top/left position the number of colums will be limited to 1, otherwise it will be set to unlimited.
ratio	Ratio between legend and the bounding rect of title, canvas and axes. The legend will be shrinked 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(), QwtPolarLayout::legendPosition(), QwtPolarLayout::setLegendPosition()

5.12.4.11 void QwtPolarPlot::itemAttached (QwtPolarItem ∗ *plotItem*, bool *on*) `[signal]`

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

Parameters

plotItem	Plot item
on	Attached/Detached

5.12.4.12 QVariant QwtPolarPlot::itemToInfo (QwtPolarItem ∗ *plotItem*) const `[virtual]`

Build an information, that can be used to identify a plot item on the legend.

The default implementation simply wraps the plot item into a QVariant object. When overloading itemToInfo() usually infoToItem() needs to reimplemeted too.

```
QVariant itemInfo;
qVariantSetValue( itemInfo, plotItem );
```

Parameters

plotItem	Plot item

See Also

infoToItem()

5.12.4.13 void QwtPolarPlot::layoutChanged ( ) `[signal]`

A signal that is emitted, whenever the layout of the plot has been recalculated.

5.12.4.14 QwtAbstractLegend ∗ QwtPolarPlot::legend ( )

Returns

the plot's legend

See Also

insertLegend()

5.12.4.15 const QwtAbstractLegend ∗ QwtPolarPlot::legend ( ) const

Returns

the plot's legend

See Also

insertLegend()

5.12.4.16 void QwtPolarPlot::legendDataChanged (const QVariant & *itemInfo*, const QList< QwtLegendData > & *data*) [signal]

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

Parameters

itemInfo	Info about a plot, build from itemToInfo()

See Also

itemToInfo(), infoToItem(), QwtAbstractLegend::updateLegend()

5.12.4.17 const QBrush & QwtPolarPlot::plotBackground ( ) const

Returns

plot background brush

See Also

plotBackground(), plotArea()

5.12.4.18 QwtPolarLayout ∗ QwtPolarPlot::plotLayout ( )

Returns

Layout, responsible for the geometry of the plot components

5.12.4.19 const QwtPolarLayout ∗ QwtPolarPlot::plotLayout ( ) const

Returns

Layout, responsible for the geometry of the plot components

5.12.4.20 int QwtPolarPlot::plotMarginHint ( ) const

Returns

Maximum of all item margin hints.

See Also

QwtPolarItem::marginHint()

5.12.4.21 QRectF QwtPolarPlot::plotRect ( ) const

The plot area depends on the size of the canvas and the zoom parameters.

Returns

Bounding rect of the plot area

5.12.4.22 QRectF QwtPolarPlot::plotRect (const QRectF & *canvasRect*) const

Calculate the bounding rect of the plot area.

The plot area depends on the zoom parameters.

Parameters

canvasRect	Rectangle of the canvas

Returns

Rectangle for displaying 100% of the plot

5.12.4.23 void QwtPolarPlot::replot ( ) [virtual], [slot]

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

setAutoReplot()

Warning

Calls canvas()->repaint, take care of infinite recursions

5.12.4.24 const QwtScaleDiv ∗ QwtPolarPlot::scaleDiv (int *scaleId*) const

Return the scale division of a specified scale.

scaleDiv(scaleId)->lBound(), scaleDiv(scaleId)->hBound() are the current limits of the scale.

Parameters

scaleId	Scale index

Returns

Scale division

See Also

QwtScaleDiv, setScaleDiv(), setScale()

5.12.4.25 QwtScaleDiv ∗ QwtPolarPlot::scaleDiv (int *scaleId*)

Return the scale division of a specified scale.

scaleDiv(scaleId)->lBound(), scaleDiv(scaleId)->hBound() are the current limits of the scale.

Parameters

scaleId	Scale index

Returns

Scale division

See Also

QwtScaleDiv, setScaleDiv(), setScale()

5.12.4.26 QwtScaleEngine ∗ QwtPolarPlot::scaleEngine (int *scaleId*)

Returns

Scale engine for a specific scale

Parameters

scaleId	Scale index

See Also

setScaleEngine()

5.12.4.27 const QwtScaleEngine ∗ QwtPolarPlot::scaleEngine (int *scaleId*) const

Returns

Scale engine for a specific scale

Parameters

scaleId	Scale index

See Also

setScaleEngine()

5.12.4.28 QwtScaleMap QwtPolarPlot::scaleMap (int *scaleId,* double *radius*) const

Build a scale map

The azimuth map translates between the scale values and angles from [0.0, 2 ∗ PI[. The radial map translates scale values into the distance from the pole.

Parameters

scaleId	Scale index
radius	Radius of the plot are in pixels

Returns

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

See Also

QwtScaleMap, transform(), invTransform()

5.12.4.29 QwtScaleMap QwtPolarPlot::scaleMap (int *scaleId*) const

Build a scale map

The azimuth map translates between the scale values and angles from [0.0, 2 ∗ PI[. The radial map translates scale values into the distance from the pole. The radial map is calculated from the current geometry of the canvas.

Parameters

scaleId	Scale index

Returns

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

See Also

QwtScaleMap, transform(), invTransform()

5.12.4.30 int QwtPolarPlot::scaleMaxMajor (int *scaleId*) const

Returns

the maximum number of major ticks for a specified axis

Parameters

scaleId	Scale index

See Also

setScaleMaxMajor()

5.12.4.31 int QwtPolarPlot::scaleMaxMinor (int *scaleId*) const

Returns

the maximum number of minor ticks for a specified axis

Parameters

scaleId	Scale index

See Also

setScaleMaxMinor()

5.12.4.32 void QwtPolarPlot::setAutoReplot (bool *enable* = `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

enable	`true` or `false`. Defaults to `true`.

See Also

replot()

5.12.4.33 void QwtPolarPlot::setAutoScale (int *scaleId*)

Enable autoscaling.

This member function is used to switch back to autoscaling mode after a fixed scale has been set. Autoscaling calculates a useful scale division from the bounding interval of all plot items with the QwtPolarItem::AutoScale attribute.

Autoscaling is only supported for the radial scale and enabled as default.

Parameters

scaleId	Scale index

See Also

hasAutoScale(), setScale(), setScaleDiv(), QwtPolarItem::boundingInterval()

5.12.4.34 void QwtPolarPlot::setAzimuthOrigin (double *origin*) `[slot]`

Change the origin of the azimuth scale.

The azimuth origin is the angle where the azimuth scale shows the value 0.0. The default origin is 0.0.

Parameters

origin	New origin

See Also

azimuthOrigin()

5.12.4.35 void QwtPolarPlot::setPlotBackground (const QBrush & *brush*)

Set the background of the plot area.

The plot area is the circle around the pole. It's radius is defined by the radial scale.

Parameters

brush	Background Brush

See Also

plotBackground(), plotArea()

5.12.4.36 void QwtPolarPlot::setScale (int *scaleId*, double *min*, double *max*, double *stepSize* = `0`)

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

Parameters

scaleId	Scale index
min	
max	minimum and maximum of the scale
stepSize	Major step size. If `step == 0`, the step size is calculated automatically using the maxMajor setting.

See Also

setScaleMaxMajor(), setAutoScale()

5.12.4.37 void QwtPolarPlot::setScaleDiv (int *scaleId*, const QwtScaleDiv & *scaleDiv*)

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

Parameters

scaleId	Scale index
scaleDiv	Scale division

See Also

setScale(), setAutoScale()

5.12.4.38 void QwtPolarPlot::setScaleEngine (int *scaleId*, QwtScaleEngine ∗ *scaleEngine*)

Change the scale engine for an axis

Parameters

scaleId	Scale index
scaleEngine	Scale engine

See Also

axisScaleEngine()

5.12.4.39 void QwtPolarPlot::setScaleMaxMajor (int *scaleId*, int *maxMajor*)

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

Parameters

scaleId	Scale index
maxMajor	maximum number of major steps

See Also

scaleMaxMajor()

5.12.4.40 void QwtPolarPlot::setScaleMaxMinor (int *scaleId*, int *maxMinor*)

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

Parameters

scaleId	Scale index
maxMinor	maximum number of minor steps

See Also

scaleMaxMajor()

5.12.4.41 void QwtPolarPlot::setTitle (const QString & *title*)

Change the plot's title

Parameters

title	New title

5.12.4.42 void QwtPolarPlot::setTitle (const QwtText & *title*)

Change the plot's title

Parameters

title	New title

5.12.4.43 QwtText QwtPolarPlot::title ( ) const

Returns

the plot's title

5.12.4.44 QwtTextLabel ∗ QwtPolarPlot::titleLabel ( )

Returns

the plot's title

5.12.4.45 const QwtTextLabel ∗ QwtPolarPlot::titleLabel ( ) const

Returns

the plot's titel label.

5.12.4.46 void QwtPolarPlot::unzoom ( )

Unzoom the plot

See Also

zoom()

5.12.4.47 void QwtPolarPlot::updateLegend ( )

Emit legendDataChanged() for all plot item

See Also

QwtPlotItem::legendData(), legendDataChanged()

5.12.4.48 void QwtPolarPlot::updateLegend (const **QwtPolarItem** ∗ *plotItem*)

Emit legendDataChanged() for a plot item

Parameters

plotItem	Plot item

See Also

QwtPlotItem::legendData(), legendDataChanged()

5.12.4.49 void QwtPolarPlot::updateScale (int *scaleId*)

Rebuild the scale

Parameters

scaleId	Scale index

5.12.4.50 QwtInterval QwtPolarPlot::visibleInterval ( ) const

Returns

Bounding interval of the radial scale that is visible on the canvas.

5.12.4.51 void QwtPolarPlot::zoom (const QwtPointPolar & *zoomPos,* double *zoomFactor*)

Translate and in/decrease the zoom factor.

In zoom mode the zoom position is in the center of the canvas. The radius of the circle depends on the size of the plot canvas, that is devided by the zoom factor. Thus a factor $<$ 1.0 zoom in.

Setting an invalid zoom position disables zooming.

Parameters

zoomPos	Center of the translation
zoomFactor	Zoom factor

See Also

unzoom(), zoomPos(), zoomFactor()

5.12.4.52 double QwtPolarPlot::zoomFactor ( ) const

Returns

Zoom factor

See Also

zoom(), zoomPos()

5.12.4.53 QwtPointPolar QwtPolarPlot::zoomPos ( ) const

Returns

Zoom position

See Also

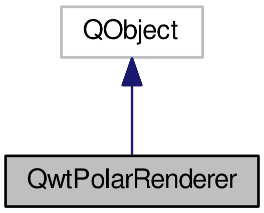
zoom(), zoomFactor()

5.13 QwtPolarRenderer Class Reference

Renderer for exporting a polar plot to a document, a printer or anything else, that is supported by QPainter/QPaint-Device.

`#include <qwt_polar_renderer.h>`

Inheritance diagram for QwtPolarRenderer:



Public Member Functions

- QwtPolarRenderer (QObject ∗parent=NULL)
- virtual ∼QwtPolarRenderer ()

  Destructor.
- void renderDocument (QwtPolarPlot ∗, const QString &format, const QSizeF &sizeMM, int resolution=85)
- void renderDocument (QwtPolarPlot ∗, const QString &title, const QString &format, const QSizeF &sizeMM, int resolution=85)
- void renderTo (QwtPolarPlot ∗, QPrinter &) const

  Render the plot to a QPrinter.
- void renderTo (QwtPolarPlot ∗, QPaintDevice &) const

  Render the plot to a `QPaintDevice`.
- virtual void render (QwtPolarPlot ∗, QPainter ∗, const QRectF &rect) const

  Render the plot to a given rectangle (f.e QPrinter, QSvgRenderer)
- bool exportTo (QwtPolarPlot ∗, const QString &documentName, const QSizeF &sizeMM=QSizeF(200, 200), int resolution=85)

  Execute a file dialog and render the plot to the selected file.
- virtual void renderTitle (QPainter ∗, const QRectF &) const
- virtual void renderLegend (const QwtPolarPlot ∗, QPainter ∗, const QRectF &) const

5.13.1 Detailed Description

Renderer for exporting a polar plot to a document, a printer or anything else, that is supported by QPainter/QPaint-Device.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 QwtPolarRenderer::QwtPolarRenderer (QObject ∗ *parent* = `NULL`) `[explicit]`

Constructor

Parameters

parent	Parent object

5.13.3 Member Function Documentation

5.13.3.1 bool QwtPolarRenderer::exportTo (QwtPolarPlot ∗ *plot,* const QString & *documentName,* const QSizeF & *sizeMM* = `QSizeF( 200, 200 )`, int *resolution* = `85`)

Execute a file dialog and render the plot to the selected file.

The document will be rendered in 85 dpi for a size 30x30 cm

Parameters

plot	Plot widget
documentName	Default document name
sizeMM	Size for the document in millimeters.
resolution	Resolution in dots per Inch (dpi)

See Also

renderDocument()

5.13.3.2 void QwtPolarRenderer::render (QwtPolarPlot ∗ *plot,* QPainter ∗ *painter,* const QRectF & *plotRect*) const `[virtual]`

Render the plot to a given rectangle (f.e QPrinter, QSvgRenderer)

Parameters

plot	Plot widget to be rendered
painter	Painter
plotRect	Bounding rectangle for the plot

5.13.3.3 void QwtPolarRenderer::renderDocument (QwtPolarPlot ∗ *plot,* const QString & *fileName,* const QSizeF & *sizeMM,* int *resolution* = `85`)

Render a polar plot to a file

The format of the document will be autodetected from the suffix of the filename.

Parameters

plot	Plot widget
fileName	Path of the file, where the document will be stored
sizeMM	Size for the document in millimeters.
resolution	Resolution in dots per Inch (dpi)

5.13.3.4 void QwtPolarRenderer::renderDocument (QwtPolarPlot ∗ *plot,* const QString & *fileName,* const QString & *format,* const QSizeF & *sizeMM,* int *resolution* = `85`)

Render a plot to a file

Supported formats are:

- pdf
- ps
- svg

- all image formats supported by Qt, see QImageWriter::supportedImageFormats()

Parameters

plot	Plot widget
fileName	Path of the file, where the document will be stored
format	Format for the document
sizeMM	Size for the document in millimeters.
resolution	Resolution in dots per Inch (dpi)

See Also

renderTo(), render(), QwtPainter::setRoundingAlignment()

5.13.3.5 void QwtPolarRenderer::renderLegend (const QwtPolarPlot ∗ *plot,* QPainter ∗ *painter,* const QRectF & *rect*) const [virtual]

Render the legend into a given rectangle.

Parameters

plot	Plot widget
painter	Painter
rect	Bounding rectangle

5.13.3.6 void QwtPolarRenderer::renderTitle (QPainter ∗ *painter,* const QRectF & *rect*) const [virtual]

Render the title into a given rectangle.

Parameters

painter	Painter
rect	Bounding rectangle

5.13.3.7 void QwtPolarRenderer::renderTo (QwtPolarPlot ∗ *plot,* QPrinter & *printer*) const

Render the plot to a QPrinter.

This function renders the contents of a QwtPolarPlot instance to `QPaintDevice` object. The size is derived from the printer metrics.

Parameters

plot	Plot to be rendered
printer	Printer to paint on

See Also

renderDocument(), render(), QwtPainter::setRoundingAlignment()

5.13.3.8 void QwtPolarRenderer::renderTo (QwtPolarPlot ∗ *plot,* QPaintDevice & *paintDevice*) const

Render the plot to a `QPaintDevice`.

This function renders the contents of a QwtPolarPlot instance to `QPaintDevice` object. The target rectangle is derived from its device metrics.

Parameters

plot	Plot to be rendered
paintDevice	device to paint on, f.e a QImage

See Also

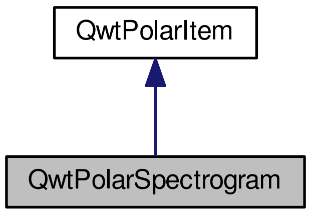
renderDocument(), render(), QwtPainter::setRoundingAlignment()

5.14 QwtPolarSpectrogram Class Reference

An item, which displays a spectrogram.

`#include <qwt_polar_spectrogram.h>`

Inheritance diagram for QwtPolarSpectrogram:



Public Types

- enum PaintAttribute { ApproximatedAtan = 0x01 }
- typedef QFlags< PaintAttribute > PaintAttributes

  Paint attributes.

Public Member Functions

- QwtPolarSpectrogram ()

  Constructor.
- virtual ∼QwtPolarSpectrogram ()

  Destructor.
- void setData (QwtRasterData ∗data)
- const QwtRasterData ∗ data () const
- void setColorMap (QwtColorMap ∗)
- const QwtColorMap ∗ colorMap () const
- void setPaintAttribute (PaintAttribute, bool on=true)
- bool testPaintAttribute (PaintAttribute) const
- virtual int rtti () const
- virtual void draw (QPainter ∗painter, const QwtScaleMap &azimuthMap, const QwtScaleMap &radialMap, const QPointF &pole, double radius, const QRectF &canvasRect) const
- virtual QwtInterval boundingInterval (int scaleId) const

Protected Member Functions

- virtual QImage renderImage (const QwtScaleMap &azimuthMap, const QwtScaleMap &radialMap, const Q-PointF &pole, const QRect &rect) const

  Render an image from the data and color map.
- virtual void renderTile (const QwtScaleMap &azimuthMap, const QwtScaleMap &radialMap, const QPointF &pole, const QPoint &imagePos, const QRect &tile, QImage ∗image) const

  Render a sub-rectangle of an image.

5.14.1 Detailed Description

An item, which displays a spectrogram.

A spectrogram displays threedimenional data, where the 3rd dimension (the intensity) is displayed using colors. The colors are calculated from the values using a color map.

See Also

QwtRasterData, QwtColorMap

5.14.2 Member Enumeration Documentation

5.14.2.1 enum QwtPolarSpectrogram::PaintAttribute

Attributes to modify the drawing algorithm. The default setting disables ApproximatedAtan

See Also

setPaintAttribute(), testPaintAttribute()

Enumerator

ApproximatedAtan Use qwtFastAtan2 instead of atan2 for translating widget into polar coordinates.

5.14.3 Member Function Documentation

5.14.3.1 QwtInterval QwtPolarSpectrogram::boundingInterval (int *scaleId*) const `[virtual]`

Interval, that is necessary to display the item This interval can be useful for operations like clipping or autoscaling

Parameters

scaleId	Scale index

Returns

bounding interval (== position)

See Also

position()

Reimplemented from QwtPolarItem.

5.14.3.2 const QwtColorMap ∗ QwtPolarSpectrogram::colorMap ( ) const

Returns

Color Map used for mapping the intensity values to colors

See Also

setColorMap()

5.14.3.3 const QwtRasterData ∗ QwtPolarSpectrogram::data ( ) const

Returns

Spectrogram data

See Also

setData()

5.14.3.4 void QwtPolarSpectrogram::draw (QPainter ∗ *painter*, const QwtScaleMap & *azimuthMap*, const QwtScaleMap & *radialMap*, const QPointF & *pole*, double *radius*, const QRectF & *canvasRect*) const [virtual]

Draw the spectrogram

Parameters

painter	Painter
azimuthMap	Maps azimuth values to values related to 0.0, M_2PI
radialMap	Maps radius values into painter coordinates.
pole	Position of the pole in painter coordinates
radius	Radius of the complete plot area in painter coordinates
canvasRect	Contents rect of the canvas in painter coordinates

Implements QwtPolarItem.

5.14.3.5 QImage QwtPolarSpectrogram::renderImage (const QwtScaleMap & *azimuthMap*, const QwtScaleMap & *radialMap*, const QPointF & *pole*, const QRect & *rect*) const [protected], [virtual]

Render an image from the data and color map.

The area is translated into a rect of the paint device. For each pixel of this rect the intensity is mapped into a color.

Parameters

azimuthMap	Maps azimuth values to values related to 0.0, M_2PI
radialMap	Maps radius values into painter coordinates.
pole	Position of the pole in painter coordinates
rect	Target rectangle of the image in painter coordinates

Returns

A QImage::Format_Indexed8 or QImage::Format_ARGB32 depending on the color map.

See Also

QwtRasterData::intensity(), QwtColorMap::rgb(), QwtColorMap::colorIndex()

5.14.3.6 void QwtPolarSpectrogram::renderTile (const QwtScaleMap & *azimuthMap*, const QwtScaleMap & *radialMap*, const QPointF & *pole*, const QPoint & *imagePos*, const QRect & *tile*, QImage ∗ *image*) const `[protected]`, `[virtual]`

Render a sub-rectangle of an image.

renderTile() is called by renderImage() to render different parts of the image by concurrent threads.

Parameters

azimuthMap	Maps azimuth values to values related to 0.0, M_2PI
radialMap	Maps radius values into painter coordinates.
pole	Position of the pole in painter coordinates
imagePos	Top/left position of the image in painter coordinates
tile	Sub-rectangle of the tile in painter coordinates
image	Image to be rendered

See Also

setRenderThreadCount()

Note

renderTile needs to be reentrant

5.14.3.7 int QwtPolarSpectrogram::rtti ( ) const [virtual]

Returns

QwtPolarItem::Rtti_PolarSpectrogram

Reimplemented from QwtPolarItem.

5.14.3.8 void QwtPolarSpectrogram::setColorMap (QwtColorMap * *colorMap*)

Change the color map

Often it is useful to display the mapping between intensities and colors as an additional plot axis, showing a color bar.

Parameters

colorMap	Color Map

See Also

colorMap(), QwtScaleWidget::setColorBarEnabled(), QwtScaleWidget::setColorMap()

5.14.3.9 void QwtPolarSpectrogram::setData (QwtRasterData * *data*)

Set the data to be displayed

Parameters

data	Spectrogram Data

See Also

data()

Warning

QwtRasterData::initRaster() is called each time before the image is rendered, but without any useful parameters. Also QwtRasterData::rasterHint() is not used.

5.14.3.10 void QwtPolarSpectrogram::setPaintAttribute (PaintAttribute *attribute*, bool *on* = `true`)

Specify an attribute how to draw the curve

Parameters

attribute	Paint attribute
on	On/Off

See Also

testPaintAttribute()

5.14.3.11 bool QwtPolarSpectrogram::testPaintAttribute (PaintAttribute *attribute*) const

Parameters

attribute	Paint attribute

Returns

True, when attribute has been set

See Also

setPaintAttribute()

Index